Automi e Linguaggi Formali – Appello 24/6/2022

Primo appello – Seconda Parte

Esercizio 1 (12 punti): Simulazione TM con stack di nastri

Teorema

Qualsiasi macchina di Turing con stack di nastri può essere simulata da una macchina di Turing standard.

Dimostrazione

Definizione TM con stack di nastri: Una TM con stack M_stack ha operazioni aggiuntive:

- **Push**: salva l'intero nastro corrente nello stack
- **Pop**: ripristina l'ultimo nastro salvato (LIFO)
- Se stack vuoto durante Pop → stato di rifiuto

Costruzione della simulazione usando TM multinastro:

Usiamo una TM a 3 nastri M_sim = (Q', Σ, Γ', δ', q_0', q_accept, q_reject):

Nastro 1: Nastro di lavoro corrente (simula il nastro principale di M_stack) **Nastro 2:** Stack dei nastri salvati
Nastro 3: Nastro ausiliario per operazioni

Codifica dello stack su Nastro 2:

```
⊢ [nastro₁] # [nastro₂] # ... # [nastroₖ] ⊣
```

dove [nastro_i] rappresenta l'i-esimo nastro salvato.

Simulazione delle operazioni:

Operazione Push:

1. Copia il contenuto completo del Nastro 1
2. Vai alla fine del Nastro 2 (cerca ⊣)

3. Scrivi # [contenuto_nastro1] prima di ⊣
4. Aggiorna il puntatore della fine dello stack

Operazione Pop:

1. Controlla se lo stack è vuoto (solo ⊢⊣ su Nastro 2)
   - Se vuoto → vai a q_reject
2. Trova l'ultimo nastro salvato (ultimo # prima di ⊣)
3. Copia il contenuto nel Nastro 1
4. Rimuovi l'entry dallo stack (cancella fino al # precedente)
5. Ripristina la posizione della testina (codificata nel nastro salvato)

Simulazione transizioni normali: Per ogni δ_stack(q, a) = (q', b, d) di M_stack:

- δ'(q, a, _, _) = (q', b, d, S, S) dove S = Stay

Codifica posizione testina: Ogni nastro salvato include un marcatore ◊ per la posizione della testina:

```
[a₁a₂◊a₃a₄...] indica testina su a₃
```

Algoritmo di simulazione dettagliato:

```
Inizializzazione:
- Nastro 1: input
- Nastro 2: ⊢⊣ (stack vuoto)
- Nastro 3: vuoto

Per ogni transizione di M_stack:
1. Se transizione normale:
   - Simula direttamente su Nastro 1

2. Se Push:
   - Marca posizione corrente testina su Nastro 1
   - Copia tutto su Nastro 3
   - Appendi a Nastro 2: # [contenuto_con_marca]

3. Se Pop:
   - Se stack vuoto → q_reject
   - Estrai ultimo elemento da Nastro 2
   - Sovrascrivi Nastro 1 con contenuto estratto
   - Ripristina posizione testina dalla marca
```

Correttezza:

Lemma 1 (Invariante): Ad ogni passo, la configurazione di M_sim codifica univocamente la configurazione di M_stack + stack.

Lemma 2 (Preservazione): Ogni transizione di M_stack è simulata correttamente da una sequenza finita di transizioni di M_sim.

Lemma 3 (Terminazione): M_sim termina nello stesso stato di M_stack.

Complessità:

- Se M_stack termina in tempo $T(n)$ usando spazio $S(n)$
- M_sim termina in tempo $O(S^2(n) \cdot T(n))$ usando spazio $O(S^2(n))$
- Il fattore quadratico deriva dalle operazioni di copia dei nastri

Conclusione: Le TM con stack di nastri non aumentano il potere computazionale oltre le TM standard. ■

Esercizio 2 (12 punti): Forty-Two è indecidibile

Teorema

Il linguaggio Forty-Two = {⟨M, w⟩ | M termina la computazione su w avendo solo 42 sul nastro} è indecidibile.

Dimostrazione

Dimostriamo per riduzione da **HALT_TM** = {⟨M, w⟩ | M termina su w}, che è indecidibile.

Costruzione della riduzione:

Data un'istanza ⟨M_0, w_0⟩ di HALT_TM, costruiamo ⟨M', w'⟩ per Forty-Two.

Costruzione di M':

```
M' = "Su input x:
1. Ignora completamente l'input x
2. Simula M₀ su w₀
3. Se M₀ termina (accetta o rifiuta):
  a. Cancella tutto il nastro
  b. Scrivi 42 sul nastro
  c. Termina nello stato di accettazione
4. Se M₀ non termina su w₀:
  - M' non termina"
```

Scelta di w': Poniamo w' = ε (stringa vuota).

Verifica della correttezza:

Caso 1: ⟨M_0, w_0⟩ ∈ HALT_TM

- M_0 termina su w_0 (in uno stato qualsiasi)
- M' simula M_0 e quando M_0 termina:
  - Cancella il nastro
  - Scrive 42
  - Termina
- Quindi M' termina su w' = ε con solo 42 sul nastro
- ⟹ ⟨M', ε⟩ ∈ Forty-Two

Caso 2: ⟨M_0, w_0⟩ ∉ HALT_TM

- M_0 non termina su w_0
- M' simula M_0 e non termina mai
- Quindi M' non termina su w'
- $\Longrightarrow \langle M', \varepsilon \rangle \notin$ Forty-Two

Computabilità della riduzione:

La funzione f: $\langle M_0, w_0 \rangle \mapsto \langle M', \varepsilon \rangle$ è computabile:

1. Dato $\langle M_0, w_0 \rangle$, costruiamo algoritmicamente M'
2. M' incorpora hard-coded M_0 e w_0 nella sua descrizione
3. Restituiamo $\langle M', \varepsilon \rangle$

Conseguenza:

Se Forty-Two fosse decidibile, allora esisterebbe un algoritmo D:

```
D(⟨M, w⟩) = {
  accetta se M termina su w con solo 42 sul nastro
  rifiuta altrimenti
}
```

Allora potremmo decidere HALT_TM:

```
A(⟨M₀, w₀⟩) = {
  1. Calcola f(⟨M₀, w₀⟩) = ⟨M', ε⟩
  2. Esegui D(⟨M', ε⟩)
  3. Restituisci lo stesso risultato
}
```

Ma questo contraddirebbe l'indecidibilità di HALT_TM.

Conclusione: Forty-Two è indecidibile. ■

Esercizio 3 (12 punti): List-Coloring

Definizione

List-Coloring = {⟨G, L⟩ | esiste una colorazione $c_1, \ldots, c_n$ degli n vertici tale che $c_v \in L(v)$ per ogni vertice v}

dove G = (V, E) è un grafo e L(v) è la lista di colori ammissibili per il vertice v.

Parte (a): List-Coloring è in NP

Teorema

List-Coloring ∈ NP.

Dimostrazione

Certificato: Una funzione di colorazione $c: V \to \mathbb{N}$.

Algoritmo di verifica V:

```
Input: ⟨G, L⟩, certificato c
1. Controlla che |c| = |V| (tempo O(|V|))
2. Per ogni vertice v ∈ V:
   - Verifica che c(v) ∈ L(v) (tempo O(|L(v)|))
3. Per ogni arco (u, v) ∈ E:
   - Verifica che c(u) ≠ c(v) (tempo O(1))
4. Se tutti i controlli passano: accetta
5. Altrimenti: rifiuta
```

Analisi della complessità:

- Passo 2: $O(|V| \cdot \max_v |L(v)|) = O(|V|^2)$ nel caso peggiore
- Passo 3: $O(|E|)$
- Totale: $O(|V|^2 + |E|)$ = tempo polinomiale

Correttezza:

- **Completezza:** Se ⟨G, L⟩ ∈ List-Coloring, allora esiste una colorazione valida c che sarà accettata da V
- **Soundness:** Se V accetta c, allora c è una colorazione valida e ⟨G, L⟩ ∈ List-Coloring

Quindi List-Coloring ∈ NP. ■

Parte (b): List-Coloring è NP-hard

Teorema

List-Coloring è NP-hard.

Dimostrazione

Dimostriamo per riduzione da **3-Color**, che è NP-completo.

3-Color: Dato un grafo G, è possibile colorare i vertici con 3 colori {1, 2, 3} tale che vertici adiacenti abbiano colori diversi?

Costruzione della riduzione:

Dato un grafo $G = (V, E)$ per 3-Color, costruiamo un'istanza $\langle G', L' \rangle$ per List-Coloring:

Grafo G': $G' = G$ (stesso grafo)

Liste di colori L': Per ogni vertice $v \in V$, poniamo $L'(v) = \{1, 2, 3\}$

Verifica della correttezza:

Direzione (⇒): Se G è 3-colorabile

- Esiste una colorazione valida $c: V \to \{1, 2, 3\}$ per G
- La stessa colorazione c soddisfa:
  - $c(v) \in L'(v) = \{1, 2, 3\}$ per ogni $v \in V$ ✓
  - $c(u) \neq c(v)$ per ogni $(u, v) \in E$ ✓
- Quindi $\langle G', L' \rangle \in$ List-Coloring

Direzione (⇐): Se $\langle G', L' \rangle \in$ List-Coloring

- Esiste una colorazione $c: V \to \mathbb{N}$ tale che:
  - $c(v) \in L'(v) = \{1, 2, 3\}$ per ogni $v \in V$
  - $c(u) \neq c(v)$ per ogni $(u, v) \in E$
- Quindi $c: V \to \{1, 2, 3\}$ è una 3-colorazione valida di G
- G è 3-colorabile

Computabilità della riduzione:

La trasformazione $G \mapsto \langle G, L' \rangle$ è chiaramente polinomiale:

- $G' = G$: $O(1)$
- Costruzione L': $O(|V|)$ per assegnare $\{1, 2, 3\}$ a ogni vertice

Conclusione:

3-Color $\leq_p$ List-Coloring

Poiché 3-Color è NP-completo, List-Coloring è NP-hard.

Risultato finale: Combinando (a) e (b), **List-Coloring è NP-completo**. ■

Note tecniche

Osservazione 1: La riduzione da 3-Color è molto naturale - List-Coloring generalizza il problema classico di colorazione.

Osservazione 2: Il problema rimane NP-hard anche con restrizioni specifiche sulle liste (ad esempio, se ogni lista ha dimensione fissa $k \geq 3$).

Osservazione 3: La dimostrazione di NP-hardness può essere estesa da k-Color per qualsiasi $k \geq 3$, rendendo List-Coloring NP-hard sotto varie parametrizzazioni.