Analisi Formale delle SRTM (Save-Restore Turing Machines)

(a) Definizione Formale della Funzione di Transizione di una SRTM

Definizione della SRTM

Una **Save-Restore Turing Machine (SRTM)** è una settupla: $M = (Q, \Sigma, \Gamma, \delta, q_0, q\_accept, q\_reject)$

Dove:

- Q: insieme finito di stati
- Σ: alfabeto di input ($\Sigma \subseteq \Gamma$)
- Γ: alfabeto del nastro
- $q_0 \in Q$: stato iniziale
- $q\_accept, q\_reject \in Q$: stati di accettazione e rifiuto
- δ: funzione di transizione estesa

Configurazioni e Stack di Save

Configurazione: Una configurazione di M è una tripla (q, w, i) dove:

- $q \in Q$: stato corrente
- $w \in \Gamma^*$: contenuto del nastro
- $i \in \mathbb{N}$: posizione della testina

Stack di configurazioni salvate: $S \in (Q \times \Gamma^* \times \mathbb{N})^*$

Funzione di Transizione Estesa

La funzione di transizione δ è definita come:

$\delta: Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\} \times \{NONE, SAVE, RESTORE\})$

Dove l'ultimo componente rappresenta l'operazione speciale da eseguire:

- **NONE**: nessuna operazione speciale
- **SAVE**: salva la configurazione corrente
- **RESTORE**: ripristina l'ultima configurazione salvata

Semantica delle Operazioni

Configurazione istantanea estesa: (q, w, i, S)

Relazione di transizione $\vdash$:

1. **Transizione normale** (op = NONE):

```
(q, w, i, S) ⊢ (q', w', i', S)
```

dove (q', a', dir, NONE) ∈ δ(q, w[i]), w' = w[1..i-1] + a' + w[i+1..], i' = i + dir_offset

2. **Operazione SAVE**:

```
(q, w, i, S) ⊢ (q', w', i', S')
```

dove (q', a', dir, SAVE) ∈ δ(q, w[i]), S' = S :: (q, w, i), w' e i' come sopra

3. **Operazione RESTORE** (con S ≠ ε):

```
(q, w, i, S :: (q_saved, w_saved, i_saved)) ⊢ (q_saved, w_saved, i_saved, S)
```

indipendentemente da δ(q, w[i])

4. **Operazione RESTORE** (con S = ε):

```
(q, w, i, ε) ⊢ (q, w, i, ε)
```

(nessun effetto se non ci sono configurazioni salvate)

(b) Equivalenza con le Macchine di Turing Standard

Teorema di Equivalenza

Teorema: Le SRTM riconoscono esattamente la classe dei linguaggi Turing-riconoscibili.

Dimostrazione:

Parte 1: TM ⊆ SRTM

Lemma: Ogni linguaggio riconosciuto da una TM può essere riconosciuto da una SRTM.

Dimostrazione: Triviale per estensione. Una TM è una SRTM che non usa mai le operazioni SAVE/RESTORE. □

Parte 2: SRTM ⊆ TM

Lemma: Ogni linguaggio riconosciuto da una SRTM può essere riconosciuto da una TM standard.

Dimostrazione per simulazione:

Sia M = (Q, Σ, Γ, δ, q_0, q_accept, q_reject) una SRTM.

Costruiamo una TM standard M' che simula M:

Struttura del nastro di M':

```
#[contenuto_originale]#[config_1]#[config_2]#...#[config_k]#
```

Dove:

- Il primo segmento contiene il nastro originale di M
- I segmenti successivi contengono le configurazioni salvate nello stack

Codifica delle configurazioni: Una configurazione (q, w, i) è codificata come: `state_q$w_1w_2...w_n$pos_i`

Algoritmo di simulazione:

1. **Inizializzazione**:
   - Copia l'input nel primo segmento
   - Inizializza puntatori per tracciare segmenti
2. **Simulazione passo-passo**:
   - **Per transizione normale**: aggiorna solo il primo segmento
   - **Per SAVE**:
     - Codifica configurazione corrente
     - Aggiungi nuovo segmento alla fine del nastro
     - Aggiorna contatori
   - **Per RESTORE**:
     - Se stack non vuoto: decodifica ultima configurazione salvata
     - Ripristina primo segmento e stato
     - Rimuovi ultimo segmento dallo stack
3. **Gestione degli stati**:

```
Q' = Q ∪ {stati_ausiliari_per_simulazione}
```

Complessità della simulazione:

- Tempo: polinomiale nel tempo di M (overhead per gestione stack)
- Spazio: lineare nello spazio di M plus spazio per configurazioni salvate

Descrizione Implementativa delle Operazioni

Implementazione SAVE:

```
Procedura SAVE():
1. Scansiona il nastro per determinare i bounds del contenuto
2. Naviga alla fine del nastro (dopo tutti i segmenti di stack)
3. Scrivi un nuovo separatore #
4. Codifica e scrivi: stato_corrente$contenuto_nastro$posizione_testina
5. Aggiorna contatore delle configurazioni salvate
6. Torna alla posizione originale nel primo segmento
```

Implementazione RESTORE:

```
Procedura RESTORE():
1. Se contatore_configurazioni = 0: return (nessun effetto)
2. Naviga all'ultimo segmento di configurazione
3. Decodifica: stato, contenuto_nastro, posizione
4. Cancella l'ultimo segmento dal nastro
5. Sovrascrivi il primo segmento con contenuto_nastro decodificato
6. Imposta stato = stato_decodificato
7. Imposta posizione_testina = posizione_decodificata
8. Decrementa contatore_configurazioni
```

Ottimizzazione: Per evitare costi eccessivi di spostamento, si può usare una rappresentazione più efficiente con puntatori bidezionali o liste linkate simulate.

Conclusione

Le SRTM sono computazionalmente equivalenti alle TM standard. L'aggiunta delle operazioni SAVE/RESTORE fornisce maggiore flessibilità nella programmazione di algoritmi (specialmente per backtracking), ma non aumenta la potenza computazionale fondamentale.

Implicazione pratica: Questo risultato conferma la robustezza della Chiesa-Turing thesis: anche estensioni apparentemente potenti del modello di calcolo non superano i limiti fondamentali della computabilità.