Lezione 11 – modelli di calcolo

Lezioni del 27/03/2025

PascalMinimo e macchine di Turing

- Obiettivo di questa lezione è dimostrare il teorema inverso del teorema 3.5, ossia il seguente
- **Teorema 3.6**: Per ogni macchina di Turing deterministica T di tipo riconoscitore ad un nastro esiste un programma $\mathcal{A}_T$ scritto in accordo alle regole del linguaggio PascalMinimo tale che: per ogni parola x, se T(x) termina nello stato finale $q_F \in \{q_A, q_R\}$ allora l'esecuzione di $\mathcal{A}_T$ con input x restituisce q_F in output e se l'esecuzione di $\mathcal{A}_T$ con input x restituisce q_A in output allora T(x) termina nello stato finale q_A

PascalMinimo e macchine di Turing

- Il nostro obiettivo sarà raggiunto progettando un algoritmo in PascalMinimo che simula la macchina di Turing Universale
  - lo trovate nella dispensa 3, nelle ultime 3 righe a pag. 9, a pag. 11, e nella Tabella 3.3 a pag. 12
  - è poco più di un esercizio
- Utilizziamo questo esercizio per dimostrare il Teorema 3.6
  - che nella dispensa è dimostrato in maniera leggermente diversa (anche se la tecnica è la stessa)
- Infatti, se progettiamo un programma $\mathcal{U}$ in PascalMinimo che simula la macchina di Turing Universale
  - ossia, tale che l'esecuzione di $\mathcal{U}$ con input la descrizione di una macchina di Turing T e un input x per T simula la computazione U(T,x)
- otteniamo un programma capace di simulare qualunque macchina di Turing

PascalMinimo e macchine di Turing

- **Teorema 3.6**: Per ogni macchina di Turing deterministica T di tipo riconoscitore ad un nastro esiste un programma $\mathcal{A}_T$ scritto in accordo alle regole del linguaggio PascalMinimo tale che: per ogni parola x, se T(x) termina nello stato finale $q_F \in \{q_A, q_R\}$ allora l'esecuzione di $\mathcal{A}_T$ con input x restituisce q_F in output e se l'esecuzione di $\mathcal{A}_T$ con input x restituisce q_A in output allora T(x) termina nello stato finale q_A
- Dimostriamo questo teorema progettando un programma $\mathcal{U}$ che si comporta come la macchina Universale:
  - utilizzando opportune strutture dati per rappresentare le quintuple di una generica macchina di Turing , e il suo stato iniziale, e i suoi stati finali
  - e altre opportune strutture dati per rappresentare un input di quella generica macchina,
  - fornendo in input ad $\mathcal{U}$ le descrizioni di una data macchina T e di un suo dato input x (in accordo alle strutture dati utilizzate)
  - l'esecuzione di $\mathcal{U}$ sul suo input restituisce un output che corrisponde allo stato in cui terminerebbe la computazione T(x)
    - o che non termina qualora T(x) non terminasse

PascalMinimo e macchine di Turing

- Progettiamo un programma $\mathcal{U}$ che si comporta come la macchina Universale:
  - per memorizzare le quintuple della macchina T che si vuole simulare, utilizziamo i 5 array Q1, S1, S2, Q2, M:
    - e usiamo i valori -1, 0, 1 per rappresentare i movimenti della testina 'sinistra', 'ferma', 'destra'
  - se la **i**-esima quintupla di T è **〈 q , a, b, q' , sinistra〉** , allora avremo
    Q1[i] =q, S1[i] = a, S2[i] = b, Q2[i] = q', M[i] = -1
    - e analogamente per i movimenti della testina 'ferma' e 'destra'
    - Q1[i] memorizza lo stato in cui si deve trovare la macchina per eseguire la quintupla i, Q2[i] memorizza lo stato in cui deve entrare la macchina dopo aver eseguito la quintupla i, e analogamente per S1[i], S2[i] e M[i]
  - rappresentiamo il nastro di T mediante l'array N, che, per semplicità, ammettiamo possa avere anche indici negativi
    - ad esempio, N[-4]: tanto il PascalMinimo ce lo stiamo inventando...
- A questo punto, vediamo il programma, nel prossimo lucido
  - **che trovate (spiegato nel dettaglio) anche sulla dispensa!**

Un programma che simula U

```
Input:  stringa x1x2...xn memorizzata nell'array N, con N[i] = xi per i = 1,...,n,
        array Q, Σ, Q1, S1, S2, Q2, M descritti nel testo,
        q0, qA, qR.
1       q ← q0;
2       t ← 1;
3       primaCella ← 1;
4       ultimaCella ← n;
5       while (q ≠ qA ∧ q ≠ qR) do begin
6               j ← 1;
7               trovata ← falso;
8               while (j ≤ k ∧ trovata = falso) do
9                       if (q = Q1[j] ∧ N[t] = S1[j]) then trovata ← vero;
10                      else j ← j + 1;
11              if (trovata = vero) then begin
12                      N[t] ← S2[j];
13                      q ← Q2[j];
14                      t ← t + M[j];
15                      if (t < primaCella) then begin
16                              primaCella ← t;
17                              N[t] ← □;
18                      end
19                      if (t > ultimaCella) then begin
20                              ultimaCella ← t;
21                              N[t] ← □;
22                      end
23              end
24              else q ← qR;
25      end
26      Output: q
```

In input viene fornita la descrizione di T (negli array Q_1, S_1, S_2, Q_2 e M e nelle variabili q_0, q_A e q_R) e del suo input (nell'array N)

q è la variabile in cui memorizziamo lo stato interno di T
e t è la variabile in cui memorizziamo la posizione della testina di T ad ogni passo della computazione

Il programma consiste in un loop while che termina quando viene raggiunto uno stato finale

NB: k è il numero di quintuple della macchina T che si vuole simulare

Un programma che simula U

```
5   while (q ≠ q_A ∧ q ≠ q_R) do begin
6       j ← 1;
7       trovata ← falso;
8       while (j ≤ k ∧ trovata = falso) do
9           if (q = Q_1[j] ∧ N[t] = S_1[j]) then trovata ← vero;
10          else j ← j + 1;
11      if (trovata = vero) then begin
12          N[t] ← S_2[j];
13          q ← Q_2[j];
14          t ← t + M[j];
15          if (t < primaCella) then begin
16              primaCella ← t;
17              N[t] ← □;
18          end
19          if (t > ultimaCella) then begin
20              ultimaCella ← t;
21              N[t] ← □;
22          end
23      end
24      else q ← q_R;
25  end
```

vengono esaminate ordinatamente, dalla prima (j=1) all'ultima (j=k), le quintuple di T fino a quando:

ne viene trovata una che può essere eseguita (quando $Q_1[j]=q$ e $S_1[j] = N[t]$) e in questo caso si pone trovata = true

oppure non ne viene trovata alcuna che può essere eseguita (quando j = k+1)

Un programma che simula U

```
3        primaCella ← 1;
4        ultimaCella ← n;
         ...
11       if (trovata = vero) then begin
12           N[t] ← S2[j];
13           q ← Q2[j];
14           t ← t + M[j];
15           if (t < primaCella) then begin
16               primaCella ← t;
17               N[t] ← □;
18           end
19           if (t > ultimaCella) then begin
20               ultimaCella ← t;
21               N[t] ← □;
22           end
23       end
```

in questo modo memorizziamo gli indici dell'array N che individuano, rispettivamente, la cella più a sinistra e la cella più a destra della porzione di nastro non blank di T

Se la quintupla da eseguire è stata trovata,
la si esegue: si aggiorna l'elemento N[t],
si aggiorna lo stato interno q,
si muove la testina (t + M[t])

se, eseguendo la quintupla, la testina di T viene spostata su una cella a sinistra della porzione di nastro sinora utilizzata (t < primaCella)
oppure su una cella a destra della porzione di nastro sinora utilizzata (t > ultimaCella)
allora primaCella o ultimaCella devono essere aggiornate

Un programma che simula U

```
5     while (q ≠ q_A ∧ q ≠ q_R) do begin
6         j ← 1;
7         trovata ← falso;
8         while (j ≤ k ∧ trovata = falso) do
9             if (q = Q_1[j] ∧ N[t] = S_1[j]) then trovata ← vero;
10            else j ← j + 1;
11        if (trovata = vero) then begin
                ...
23        end
24        else q ← q_R;
25    end
26    Output: q
```

se non viene trovata alcuna quintupla da eseguire, si porta la macchina nello stato di rigetto

Ricordate quello che avevamo detto a proposito di un insieme di quintuple **non** *completo?*

Macchina non deterministica in PascalMinimo

- Guardiamo insieme ora l'algoritmo in PascalMinimo che simula una macchina di Turing non deterministica (che trovate al paragrafo 3.4)
  - meno facile rispetto alla simulazione della macchina Universale...
- L'algoritmo implementa in PascalMinimo la coda di rondine con ripetizioni che abbiamo descritto informalmente nel corso della Lezione 4, e che dimostra il Teorema 2.1 (Dispensa 2):
  - inizializza un contatore i a 1
  - simula ***tutte*** le computazioni deterministiche di i istruzioni
    - se una di esse accetta, allora accetta
    - altrimenti; se tutte rigettano, allora rigetta
  - se al passo precedente non hai terminato (ossia, nessuna computazione di i passi ha accettato e almeno una di esse non ha rigettato), allora incrementa il valore di i e ripeti il passo precedente

Macchina non deterministica in PascalMinimo

- e tutto ciò si traduce in PascalMinimo nel programma seguente

```
Input: stringa x1x2...xn memorizzata nell'array N, con N[i] = xi per i = 1,...,n.
       Il programma utilizza anche le seguenti costanti: q0, qA, qR e inoltre
       gli array Q, Σ, Q1, S1, S2,Q2, M descritti nel testo,
1      q ← q0;
2      i ← 1;
3      primaCella ← 1;
4      ultimaCella ← n;
5      while (q ≠ qA ∧ q ≠ qR) do begin
6          q ← simulaRicorsivo(q0, 1, N, i);
7          i ← i + 1;
8      end
9      Output: q
```

primaCella e ultimaCella sono variabili globali

- ove la simulazione di tutte le computazioni deterministiche è eseguita dall'invocazione della funzione ricorsiva `simulaRicorsivo`(q_0 , 1, N , i)
  - i cui parametri sono: lo stato interno (q_0), la posizione della testina (1) e il contenuto del nastro (N) della macchina non deterministica quando ha inizio la simulazione, e la lunghezza (i) delle computazioni da simulare

Macchina non deterministica in PascalMinimo

- Ecco lo schema della funzione ricorsiva:

```
Q simulaRicorsivo(Q q, int t, Σ[ ] N, int i)
begin
    if (i = 0) then ρ ← q;
    else begin
        per ogni quintupla che puoi eseguire a partire dallo stato globale in cui ti trovi
            eseguila e fai partire tutte le la simulazioni delle computazioni lunghe i-1
                (invocazione ricorsiva di simulaRicorsivo) :
                se una di esse restituisce qA (ossia, accetta) allora termina
                    le simulazioni con ρ = qA
                altrimenti, se almeno una simulazione non restituisce qR (ossia, non rigetta)
                    allora poni rigetto ← falso per ricordartelo
                altrimenti, se tutte le simulazioni restituiscono qR (ossia, rigettano)
                    allora avrai ρ = qR
    end
    if (ρ ≠ qA e rigetto = falso) then  ρ ← q0;
    return ρ
end
```

```
    Q funzione simulaRicorsivo(Q q, int t, Σ[ ] N, int i)
1   begin
2       if (i = 0) then ρ ← q;
            // la precedente istruzione gestisce il caso in cui la ricorsione termina
3       else begin
4           j ← 1;
5           rigetto ← vero;
6           while (j ≤ k ∧ q ≠ q_A) do begin
7               while (j ≤ k ∧ [ q ≠ Q_1[j] ∨ N[t] ≠ S_1[j] ]) do j ← j + 1;
8               if (j = k + 1) then ρ ← q_R;
                    // nessuna (ulteriore) quintupla da eseguire è stata trovata
9               else begin
                    // ora i primi due elementi di p_j sono q e N[t]
10                  N[t] ← S_2[j];
11                  if (t + M[j] < primaCella) then begin
12                      primaCella ← t + M[j];
13                      N[t + M[j]] ← □;
14                  end
15                  if (t + M[j] > ultimaCella) then begin
16                      ultimaCella ← t + M[j];
17                      N[t + M[j]] ← □;
18                  end
19                  ρ ← simulaRicorsivo(Q_2[j], t + M[j], N, i − 1);
20                  if (ρ = q_A) then q ← q_A;
21                  else if (ρ ≠ q_R) then rigetto ← falso;
22                  N[t] ← S_1[j];
                        // lo stato di N viene ripristinato alla situazione precedente l'esecuzione
                        della quintupla p_j
23                  j ← j + 1;
                        // si predispone ad iniziare la ricerca di una nuova quintupla da eseguire:
                        tale ricerca avrà luogo solo se nessuna computazione deterministica
                        precedente (iniziata da una quintupla p_h con h < j) ha accettato
24              end;
25          end;
26          if (ρ ≠ q_A ∧ rigetto = falso) then ρ ← q_0;
                // la precedente istruzione gestisce il caso in cui il ciclo while alle linee 6-25
                termina senza aver trovato lo stato q_A e senza poter decidere circa il rigetto;
                si osservi che, se ρ ≠ q_A e rigetto = vero, allora tutte le quintuple eseguite
                hanno portato a rigettare e, quindi, ρ = q_R
27      end;
28      return: ρ;
29  end
```

primaCella e ultimaCella sono variabili globali