Lezione 7 – modelli di calcolo

Lezioni del 26-27/03/2024

Ri-facciamo il punto

- Siamo partiti cercando di capire come risolvere automaticamente i problemi
- E abbiamo studiato la soluzione proposta da Alan Turing che, partendo dalla sua analisi del processo di soluzione è arrivato a definire un modello di calcolo: la Macchina di Turing
  - che è un linguaggio per descrivere algoritmi
  - e ogni macchina di Turing è un algoritmo
- Poi, abbiamo introdotto i concetti di linguaggi decidibili e accettabili, e di funzioni calcolabili
  - che corrispondono, informalmente, ai problemi che sappiamo risolvere con la Macchina di Turing
- considerando, così, implicitamente, la possibilità che possano esistere linguaggi non decidibili – o, persino, non accettabili (uhmm…) – e funzioni non calcolabili
  - la possibilità che esistano problemi irrisolvibili – con la Macchina di Turing

A questo punto

- Ma, ammesso (e non concesso!) che esista un linguaggio non decidibile / non accettabile, oppure una funzione non calcolabile, non sarà forse possibile decidere / accettare quel linguaggio, oppure calcolare quella funzione, con un altro modello di calcolo?
- In fondo, cos'ha di tanto speciale questa Macchina di Turing????
- APERTA PARENTESI. Togliamoci subito un dente: chi mi dimostra che "esiste un linguaggio non decidibile se e soltanto se esiste un linguaggio non accettabile" ?
  - suggerimento: guardate i teoremi della scorsa lezione: ce n'è uno che…
- CHIUSA PARENTESI
- Torniamo alla nostra questione: esiste un modello di calcolo più potente della Macchina di di Turing? Che "sa risolvere più problemi"?

Modelli di calcolo

- Siamo al paragrafo 3.3 della dispensa 3
- Dove si dice che sono stati definiti un sacco di modelli di calcolo
  - che, guarda caso, sono tutti basati sullo stesso concetto di "operazione elementare" utilizzato da Turing
  - perché esso corrisponde proprio all'umano modo di calcolare
- Ebbene: per tutti i modelli di calcolo definiti fino ad ora, è stato dimostrato che sanno "risolvere" tutti e soli i problemi che possono essere "risolti" mediante la Macchina di Turing
  - non uno di più, non uno di meno
- ossia, tutti i modelli di calcolo introdotti sino ad ora sono ***Turing-equivalenti***
- Viene quasi da pensare che un modello di calcolo più potente della Macchina di Turing non esista
  - posto che esso consideri "operazione elementare" una operazione che possa essere eseguita "a mente" da un umano medio!

La tesi di Church-Turing

- Questa tesi assume che non esista un modello di calcolo più potente della Macchina di Turing: dato un qualunque altro modello di calcolo $\mathcal{M}$,
  - se un linguaggio L è decidibile/accettabile nel modello $\mathcal{M}$ allora L è decidibile/accettabile nel modello Macchina di Turing
  - se una funzione f è calcolabile nel modello $\mathcal{M}$ allora f è calcolabile nel modello Macchina di Turing
  - e viceversa
- Purché $\mathcal{M}$ sia un modello" ragionevole"
  - ossia, sia basato sul concetto di *operazione elementare* del quale abbiamo parlato diffusamente
  - perché, se definiamo un modello di calcolo che disponga dell'unica operazione elementare "qualunque sia il problema, qualunque sia l'istanza del problema, trova la soluzione di quell'istanza",…
  - beh, è ovvio che questo modello è più potente della macchina di Turing!
  - Ma non è mica tanto realistico (nel senso che dalla Macchina di Turing sono nati i calcolatori, ma è difficile che nascano macchine reali che corrispondano a questo modello)

La tesi di Church-Turing

- Dunque:

  è calcolabile tutto e solo ciò che può essere calcolato dalla Macchina di Turing

- Attenzione: è una tesi, non è un teorema!
  - Non è mai stata dimostrata!
  - E sembra difficile riuscire a dimostrarla: sembra difficile riuscire a prevedere i modelli di calcolo che potrebbero essere definiti nel futuro…
  - Tuttavia, sembra poco probabile riuscire a progettare un modello di calcolo che non la soddisfi
  - e, non dimentichiamolo, tutti i modelli di calcolo esistenti la soddisfano
  - infatti, è generalmente accettata!

Il modello di calcolo PascalMinimo

- È un linguaggio di programmazione – perché ogni linguaggio di programmazione è un modello di calcolo!
- Dispone di tutte le istruzioni "tipiche" dei linguaggi di programmazione
  - istruzione di assegnazione: a ← b
  - istruzione condizionale **if** … **then** ... **else**
  - istruzioni di loop **while** (…) **do** e **for** (...)
  - funzioni
  - istruzioni per l'input e l'output
  - ecc. ecc.
- E dispone di variabili semplici (intere, caratteri, ...) ma anche di variabili che corrispondono a collezioni di oggetti – insiemi e, soprattutto, **array**
  - se avete provato a risolvere un certo esercizio, questo dovrebbe dirvi qualcosa...
- E la descrizione di questo linguaggio la trovate a pag. 7 della dispensa 3

Il modello di calcolo PascalMinimo

- Nella dispensa 3, a partire da pag. 7, si accenna alla dimostrazione che il modello di calcolo PascalMinimo è equivalente alla Macchina di Turing

  - nel Teorema 3.5 si dà un'idea (grossolana) di come "trasformare" un programma in PascalMinimo in una macchina di Turing che "faccia le stesse cose"

  - nel Teorema 3.6 si dà un'idea (abbastanza precisa) di come "trasformare" una macchina di Turing in un programma in PascalMinimo che "faccia le stesse cose"

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.

- Sulla dispensa viene descritta l'idea della dimostrazione
  - in particolare, viene mostrato come "trasformare" un programma in PascalMinimo in una macchina di Turing di tipo trasduttore solo nel caso in cui il programma non utilizzi variabili strutturate
  - come, ad esempio, gli array

- Guardiamo questa idea di dimostrazione insieme
  - includendo nella nostra discussione anche gli array

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: sia $\mathcal{P}$ un programma in PascalMinimo, prima di mostrare come costruire una macchina di Turing T che ''si comporta'' come $\mathcal{P}$, scriviamo $\mathcal{P}$ in una forma opportuna, ossia
  - 1) utilizzando variabili ausiliarie, eliminiamo gli array eventualmente presenti nelle condizioni
    - ad esempio,''if (A[i]=1) then ... '' diventa ''ausilA = A[i]; if (ausilA=1) then ...''
    - naturalmente, potremo utilizzare la stessa variabile di appoggio in più punti del programma
    - in questo modo, gli array compaiono solo nelle istruzioni di assegnazione
  - 2) scriviamo una sola istruzione in ciascuna riga del programma
  - 3) numeriamo le righe del programma

Il modello di calcolo PascalMinimo

- Idea della dimostrazione. Esempio: il seguente programma

Input: n valori interi positivi memorizzati nell'array A e un intero memorizzato nella variabile k

```
p ← 1; i ← 1;
 while (i ≤ n) do
        if (A[i] > p) then p ← A[i];
if (p > k) then ris ← k else ris ← p
output ris
```

- diventa:

```
1) p ← 1;
2) i ← 1;
3) while (i ≤ n) do  begin
4)     ausilA ← A[i];
5)     if (ausilA > p) then
6)             p ← A[i];
7) end
8) if (p > k) then ris ← k
9)     else ris ← p
10) output ris
```

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: sia $\mathcal{P}$ un programma in PascalMinimo, costruiamo T, multinastro a testine indipendenti
  - Oltre ai nastri input e output, T utilizza:
    - un nastro per ciascuna variabile (incluse le variabili di tipo array)
    - un nastro aggiuntivo per ogni variabile di tipo array sul quale memorizzare in unario l'indice dell'array al quale si vuole accedere
    - un nastro di lavoro per la valutazione delle espressioni e delle condizioni contenute nel programma.
  - I contenuti dei nastri sono codificati in binario (o in unario) con una sola eccezione: nei nastri che corrispondono a variabili di tipo array viene utilizzato un carattere speciale, '\$', come carattere separatore fra gli elementi dell'array

Il modello di calcolo PascalMinimo

- Idea della dimostrazione. Esempio: il seguente programma

Input: n valori interi positivi memorizzati nell'array A e un intero memorizzato nella variabile k

```
1) p ← 1;
2) i ← 1;
3) while (i ≤ n) do  begin
4)     ausilA ← A[i];
5)     if (ausilA > p) then
6)           p ← A[i];
7) end
8) if (p > k) then ris ← k
9)     else ris ← p
10) output ris
```

- La macchina corrispondente al programma utilizza 8 nastri:
  - 5 nastri per le variabili semplici: il nastro N_p per p, il nastro N_i per i, , il nastro N_{ausilA} per ausilA. il nastro N_k per k, il nastro N_{ris} per ris
  - un nastro N_A per la variabile di tipo array A e un nastro N_{indA} per l'indice con il quale accedere ad A
  - un nastro di lavoro

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: sia $\mathcal{P}$ un programma in PascalMinimo, costruiamo T assegnando uno stato interno di T ad ogni riga del programma che contenga una istruzione,
- Vedremo ora come costruire le quintuple di T:
  - la descrizione che presenteremo è <u>ad alto livello</u>, rappresentando lo stato di partenza, le azioni che devono essere compiute a partire da quello stato e lo stato in cui entra T dopo che quelle azioni sono state eseguite
  - ad esempio con

    $\langle$ q, **[copia N_b su N_a riavvolgendo le testine su N_b e N_a]**, q', ferme $\rangle$

    intendiamo che quando la macchina è nello stato q, indipendentemente da quello che viene letto sui nastri, T inizia ad eseguire una serie di quintuple che (utilizzando un certo numero di stati intermedi) copiano il contenuto del nastro N_b sul nastro N_a, poi riavvolgono le testine su quei nastri e infine, mantenendo ferme tutte le testine, portano T nello stato q'

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: sia $\mathcal{P}$ un programma in PascalMinimo, costruiamo T assegnando uno stato interno di T ad ogni riga del programma che contenga una istruzione,
- Vedremo ora come costruire le quintuple di T:
  - la descrizione che presenteremo è <u>ad alto livello</u>, rappresentando lo stato di partenza, le azioni che devono essere compiute a partire da quello stato e lo stato in cui entra T dopo che quelle azioni sono state eseguite
  - invece con

    $\langle$ q, **[sul nastro N_c è scritto xxx]**, q', ferme $\rangle$

    intendiamo che quando la macchina è nello stato q, se sul nastro N_c trova scritto xxx entra nello stato q' senza muovere le testine
  - xxx può essere una parola (invece che un singolo carattere): in tal caso T deve eseguire una serie di quintuple per verificare che su N_c sia scritta la parola xxx

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione di assegnazione fra variabili semplici, dopo aver eseguito l'assegnazione T entra nello stato q_{i+1}
  - ESEMPIO: **21**) a =b;
    22) if (a > 5) then
  - diventa:
    ⟨ q_{21}, **[copia N_b su N_a e posiziona le testine sui primi simboli di N_b su N_a]**, q_{22}, ferme ⟩
    - assumiamo che quando T entra in uno stato corrispondente a un'istruzione (ad esempio, in q_{21}) le testine sono sempre posizionate sui simboli più a sinistra su ciascun nastro
    - **[copia N_b su N_a e posiziona le testine sui primi simboli di N_b su N_a] corrisponde, in effetti ad un insieme di quintuple: terminata la copia e riposizionate le testine a sinistra, T entra in q_{22}**

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione di assegnazione che coinvolge una variabile A di tipo array:
    - 1) viene copiato su N_{indA} in unario il valore dell'indice dell'elemento di A cui si vuole accedere,
    - 2) ci si posiziona su quell'elemento,
    - 3) si esegue l'assegnazione dopo la quale T entra nello stato q_{i+1}

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - ESEMPIO: 15) A[i] = b;
    16) ...
  - per semplificarci la vita, assumiamo che ogni elemento di A possa essere contenuto in un'unica cella del nastro N_A
    - in questo modo non utilizziamo il carattere separatore $ gli elementi dell'array sono scritti in celle contigue di N_A
  - allora, i tre punti descritti alla pagina precedente diventano

$\langle q_{15}$, **[copia N_i in unario su N_{indA} e riposiziona le testine a sinistra]**, q_{15}^1, ferme $\rangle$
$\langle q_{15}^1$, **[muovi a destra su N_{indA} e N_A sino al blank su N_{indA}]**, q_{15}^2, ferme $\rangle$
$\langle q_{15}^2$, **[copia N_b su N_A e riposiziona le testine a sinistra]**, q_{16}, ferme $\rangle$

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione di assegnazione che coinvolge una variabile A di tipo array: 19 viene copiato su N_{indA} in unario il valore dell'indice dell'elemento di A cui si vuole accedere, 2) ci si posiziona su quell'elemento, 3) si esegue l'assegnazione dopo la quale T entra nello stato q_{i+1}
  - ESEMPIO: **15**) A[i] = b;
    16) ...
  - per semplificarci la vita, abbiamo assunto che ogni elemento di A possa essere contenuto in un'unica cella del nastro N_A
  - questa assunzione può essere eliminata utilizzando il carattere separatore '$' su N_A (esercizio non proprio facile facile)

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione **if (*condizione*) then ...**, calcola *condizione* : se è vera T entra nello stato q_{i+1}, altrimenti entra nello stato corrispondente alo stato dell'istruzione da eseguire quando è falsa
  - ESEMPIO: **22**) if (a > 5) then
    23) c = a;
    24) ...
  - diventa:
    $\langle q_{22}$, **[calcola *condizione* sul nastro di lavoro scrivendovi il risultato]**, q^1_{22}, ferme $\rangle$
    $\langle q^1_{22}$, **[sul nastro di lavoro c'è scritto *vero*]**, q_{23}, ferme $\rangle$
    $\langle q^1_{22}$, **[sul nastro di lavoro c'è scritto *falso*]**, q_{24}, ferme $\rangle$

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: quando le quintuple che corrispondono a una istruzione sono terminate, T entra nello stato che corrisponde alla successiva istruzione che deve essere eseguita:
  - se la riga i contiene una istruzione **while (*condizione*) do...**, calcola *condizione* : se è vera T entra nello stato q_{i+1},*e poi di seguito fino all'ultima istruzione del loop che riporta T in* q_i, altrimenti entra nello stato corrispondente alo stato dell'istruzione da eseguire dopo il loop
  - ESEMPIO: **24**) while (a > 5) do begin
    25) a = a-1;
    26) . i = i+1; end
    27) ...
  - diventa:
    ⟨q_{24}, **[calcola** a > 5 **utilizzando** N_a **sul nastro di lavoro scrivendovi il risultato]**, q_{24}^1, ferme ⟩
    ⟨q_{24}^1, **[sul nastro di lavoro c'è scritto *vero*]**, q_{25}, ferme ⟩
    ⟨q_{25}, **[sottrai 1 al contenuto di** N_a**]**, q_{26}, ferme ⟩
    ⟨q_{26}, **[somma 1 al contenuto di** N_i**]**, q_{24}, ferme ⟩
    ⟨q_{24}^1, **[sul nastro di lavoro c'è scritto *falso*]**, q_{27}, ferme ⟩

Il modello di calcolo PascalMinimo

- **Teorema 3.5**: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo, esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.
- Idea della dimostrazione: dato $\mathcal{P}$ un programma in PascalMinimo, abbiamo costruito una macchina di Turing T
  - che utilizza tanti nastri quante sono le variabili in $\mathcal{P}$ (più i nastri indice degli array)
  - che dispone di un insieme di stati interni ''principali'' ciascuno dei quali corrisponde a una istruzione in $\mathcal{P}$
- descrivendo ad alto livello le sue quintuple
- poiché le quintuple di T simulano, passo passo, le istruzioni di $\mathcal{P}$, intuitivavamente quanto stabilito dall'asserto del teorema è vero
- ossia, abbiamo dimostrato *informalmente* il teorema
  - in effetti, abbiamo visto solo un' *idea della dimostrazione*

PascalMinimo e macchine di Turing

- Guardiamo ora l'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
  - facile: dovete solo prendere confidenza con le strutture dati (semplici array)
  - 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

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 $\mathcal{A}_T$ con input x restituisce q_F in output.
- 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 voi dovete studiare sulla dispensa!**

Un programma che simula U

Input: stringa $x_1 x_2 \ldots x_n$ memorizzata nell'array N, con $N[i] = x_i$ per $i = 1, \ldots, n$,
array Q, Σ, Q_1, S_1, S_2, Q_2, M descritti nel testo,
q_0, q_A, q_R.

$q \leftarrow q_0$;
$t \leftarrow 1$;
$primaCella \leftarrow 1$;
$ultimaCella \leftarrow n$;
while $(q \neq q_A \wedge q \neq q_R)$ **do begin**
    $j \leftarrow 1$;
    $trovata \leftarrow$ `falso`;
    while $(j \leq k \wedge trovata =$ `falso`$)$ **do**
        if $(q = Q_1[j] \wedge N[t] = S_1[j])$ **then** $trovata \leftarrow$ `vero`;
        else $j \leftarrow j + 1$;
    if $(trovata =$ `vero`$)$ **then begin**
        $N[t] \leftarrow S_2[j]$;
        $q \leftarrow Q_2[j]$;
        $t \leftarrow t + M[j]$;
        if $(t < primaCella)$ **then begin**
            $primaCella \leftarrow t$;
            $N[t] \leftarrow \square$;
        end
        if $(t > ultimaCella)$ **then begin**
            $ultimaCella \leftarrow t$;
            $N[t] \leftarrow \square$;
        end
    end
    else $q \leftarrow q_R$;
end
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

```
while (q ≠ q_A ∧ q ≠ q_R) do begin
        j ← 1;
        trovata ← falso;
        while (j ≤ k ∧ trovata = falso) do
                if (q = Q_1[j] ∧ N[t] = S_1[j]) then trovata ← vero;
                else j ← j + 1;
        if (trovata = vero) then begin
                N[t] ← S_2[j];
                q ← Q_2[j];
                t ← t + M[j];
                if (t < primaCella) then begin
                        primaCella ← t;
                        N[t] ← □;
                end
                if (t > ultimaCella) then begin
                        ultimaCella ← t;
                        N[t] ← □;
                end
        end
        else q ← q_R;
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

```
        primaCella ← 1;
        ultimaCella ← n;
        ...
            if (trovata = vero) then begin
                N[t] ← S2[j];
                q ← Q2[j];
                t ← t + M[j];
                if (t < primaCella) then begin
                    primaCella ← t;
                    N[t] ← □;
                end
                if (t > ultimaCella) then begin
                    ultimaCella ← t;
                    N[t] ← □;
                end
            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

while $(q \neq q_A \ \wedge\ q \neq q_R)$ **do begin**

  $j \leftarrow 1$;

  $trovata \leftarrow \texttt{falso}$;

  while $(j \leq k \wedge trovata = \texttt{falso})$ **do**

    if $(q = Q_1[j] \wedge N[t] = S_1[j])$ **then** $trovata \leftarrow \texttt{vero}$;

    else $j \leftarrow j + 1$;

  if $(trovata = \texttt{vero})$ **then begin**

    ...

  end

  $\boxed{\textbf{else } q \leftarrow q_R;}$

end

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 $x_1 x_2 \dots x_n$ memorizzata nell'array N, con $N[i] = x_i$ per $i = 1, \dots, n$.
Il programma utilizza anche le seguenti costanti: q_0, q_A, q_R e inoltre gli array Q, Σ, Q_1, S_1, S_2,Q_2, M descritti nel testo,

$i \leftarrow 1;$
$primaCella \leftarrow 1;$
$ultimaCella \leftarrow n;$
while $(q \neq q_A \ \wedge \ q \neq q_R)$ **do begin**
    $q \leftarrow$ `simulaRicorsivo`$(q_0, 1, N, i);$
    $i \leftarrow i + 1;$
end
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 $\rho \leftarrow$ 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 q_A (ossia, accetta) allora termina
                    le simulazioni con $\rho = q_A$
                altrimenti, se almeno una simulazione **non** restituisce q_R (ossia, non rigetta)
                    allora poni rigetto ← falso per ricordartelo
                altrimenti, se **tutte** le simulazioni restituiscono q_R (ossia, rigettano)
                    allora avrai $\rho = q_R$
    end
    if ($\rho \neq q_A$ e rigetto = falso) then $\rho \leftarrow q_0$;
    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