Computabilità
Appunti_09 (21/12/2022)

Mario Verdicchio
Università degli Studi di Bergamo
Anno Accademico 2022-2023

Problema dell'arresto

- Dato il codice di una MT e un input I, la computazione di MT su I termina in tempo finito?
- Questo problema non è decidibile, ossia non è risolvibile tramite una MT
- Lo dimostriamo per assurdo, supponendo che esista una macchina H che riesca, in tempo finito, a darci sempre una risposta (positiva o negativa)

La macchina H

- Sia C_{MT} la codifica del codice di una MT
- Si suppone che esista la seguente macchina:

$$H(C_{MT}, I) = \begin{cases} 0, \text{ se MT(I) termina} \\ 1, \text{ se MT(I) non termina} \end{cases}$$

- Sulla base della macchina H, costruiamo la macchina H'

La macchina H’

- Si costruisce la seguente macchina:

$$H'(C_{MT}) = \begin{cases} 0, \text{ se } MT(C_{MT}) \text{ termina, ossia } H(C_{MT}, C_{MT}) = 0 \\ 1, \text{ se } MT(C_{MT}) \text{ non termina, } H(C_{MT}, C_{MT}) = 1 \end{cases}$$

- La macchina H’ computa una funzione che è un caso particolare della funzione computata da H (C_{MT} è un valore particolare di I)

La macchina Z

- Sulla base di H', si costruisce la seguente macchina:

$$Z(C_{MT}) = \begin{cases} \text{non termina, se } H'(C_{MT}) = 0 \\ 0, \text{ se } H'(C_{MT}) = 1 \end{cases}$$

- Per la non-terminazione basta aggiungere istruzioni che facciano sì che, se H' restituisce 0 in output, la testina di Z vada, ad esempio, a destra all'infinito

Il paradosso

- Proviamo a calcolare $Z(C_Z)$
  a) $Z(C_Z)$ non termina $\Leftrightarrow$ $H'(C_{MT})= 0 \Leftrightarrow$ $Z(C_Z)$ termina
  b) $Z(C_Z) = 0 \Leftrightarrow H'(C_{MT})= 1 \Leftrightarrow Z(C_Z)$ non termina
- In ogni caso, si arriva a un assurdo che ci costringe a respingere l'assunzione che la macchina H esista
- La macchina H non può esistere, ossia il problema dell'arresto non è decidibile

Controesempio?

```
while (1) {
    print (0);
    }
```

NO: non è un controesempio perché si tratta di un caso specifico risolvibile algoritmicamente. Quello che non esiste è una MT che possa fare l'analisi di terminazione o meno per qualunque codice C_{MT} e input I.

Funzioni base e operazioni

- Funzioni base:
  - funzione zero, $Z:N \to N$, $Z(x)=0$ per ogni x
  - funzione successore, $s:N \to N$, $s(x)$ = il successore di x nell'enumerazione di N
  - proiezioni, $P^n_i:N^n \to N$, $P^n_i(x_1,\ldots,x_n)=x_i$
- Operazione di composizione:
  - date $\chi:N^k \to N$ e $\psi_1,\ldots,\psi_k:N^n \to N$
  - si ottiene per composizione la funzione $\phi:N^n \to N$ quando
    $\phi(x_1,\ldots,x_n) = \chi(\psi_1(x_1,\ldots,x_n),\ldots,\psi_1(x_1,\ldots,x_n))$
- Operazione di ricorsione:
  - date $\chi:N^n \to N$ e $\psi:N^{n+2} \to N$
  - si ottiene per ricorsione la funzione $\phi:N^{n+1} \to N$ quando

$$\begin{cases} \phi(x_1,\ldots,x_n,0) = \chi(x_1,\ldots,x_n) \\ \phi(x_1,\ldots,x_n,s(y)) = \psi(x_1,\ldots,x_n,y,\phi(x_1,\ldots,x_n,y)) \end{cases}$$

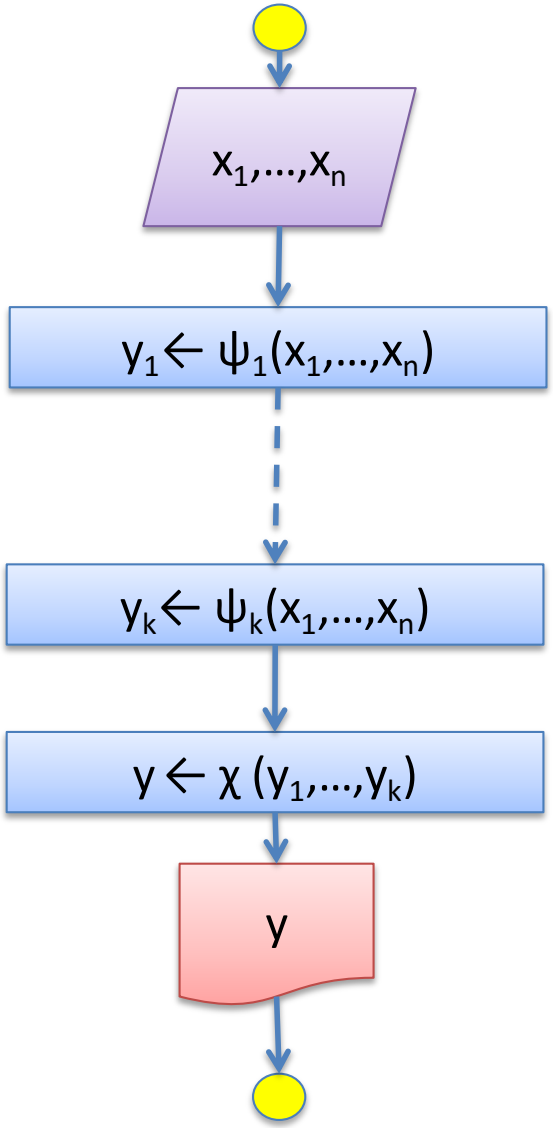
Funzioni ricorsive primitive (RP)

- Si dice ricorsiva primitiva una funzione che si ottiene dalle funzioni base applicando un numero finito di volte le operazioni di composizione e ricorsione
- RP è linsieme di tutte e sole le funzioni ricorsive primitive
- Una derivazione in RP è una sequenza di funzioni, ciascuna delle quali è una funzione base, oppure è stata ottenuta dalle funzioni precedenti tramite composizione o ricorsione
- L'ultima funzione in una derivazione in RP si dice RP-derivabile
- Una fuzione è in RP $\Leftrightarrow$ è RP-derivabile
- Esempio di derivazione dell'addizione in RP:
  1. $\phi_1(x) = P^1_1(x)$ [base]
  2. $\phi_2(x) = s(x)$ [base]
  3. $\phi_3(x,y,z) = P^3_3(x,y,z)$ [base]
  4. $\phi_4(x,y,z) = s(P^3_3(x,y,z))$ [composizione di 2 e 3]
  5. $\begin{cases} \phi_5(x,0) = P^1_1(x) \\ \phi_5(x,s(y)) = s(P^3_3(x,y,\phi_5(x,y))) \text{ [ricorsione di 1 e 4]} \end{cases}$

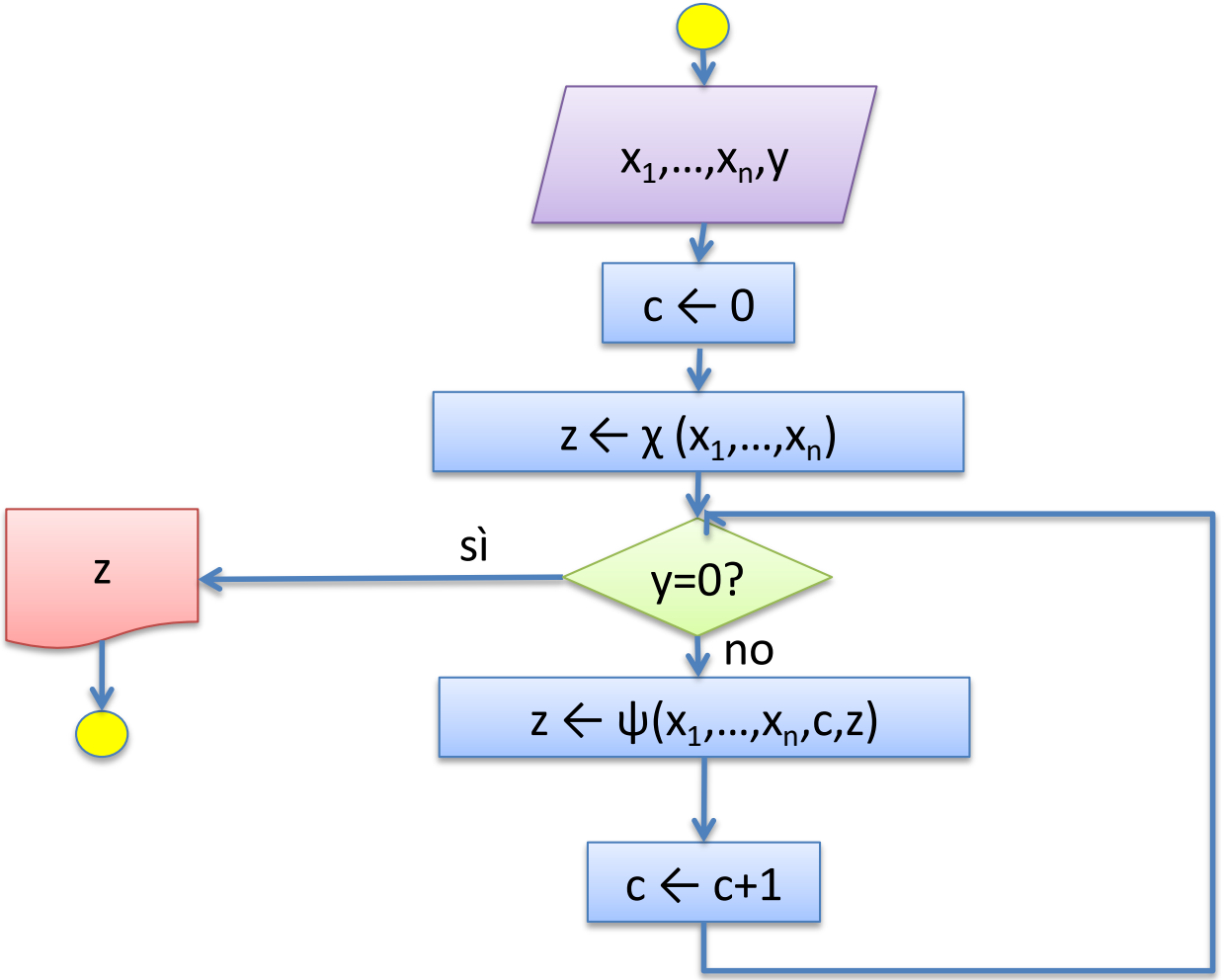
- $\phi_5(x,0) = P^1_1(x)$
  $\phi_5(x,s(y)) = s(P^3_3(x,y,\phi_5(x,y)))$
- $\phi_5(x,0) = x$
- $\phi_5\ (x,3) = \phi_5(x,s(2)) = s(P^3_3(x,2,\phi_5(x,2))) =$
  $s(\phi_5(x,2)) = \ s(s(\phi_5(x,1))) = s(s(s(\phi_5(x,0))))=$
  $s(s(s(x)))$
  ϕ_5 si comporta come +
  dal nostro punto di vista, DEFINIAMO +

come ϕ_5 dimostrando che + è in RP, cioè è una funzione ricorsiva primitiva

La composizione conserva la computabilità e la totalità delle funzioni



La ricorsione conserva la computabilità e la totalità delle funzioni



Le funzioni RP

- Le funzioni base sono computabili e totali
- Le operazioni conservano tali caratteristiche delle funzioni a cui sono applicate
- Perciò le funzioni RP sono computabili e totali

Funzioni computabili totali non RP

- Le funzioni RP (ad 1 argomento, senza perdere in generalità) ϕ_i sono enumerabili perché possiamo costruire una sequenza di funzioni ottenute da quelle base con 0, 1, 2, 3,… applicazioni delle operazioni di composizione e ricorsione
- Data tale sequenza, definiamo $f(x) = \phi_x(x)+1$
- f è computabile e totale
- Se supponiamo che f sia nella sequenza, ovvero che $f \equiv \phi_k$, calcoliamo f(k)
- Per la definizione di f, $f(k) = \phi_k(k)+1$
- Per la posizione di f nella sequenza, $f(k) = \phi_k(k)$
- Si conclude che $\phi_k(k)+1 = \phi_k(k)$, ossia, visto che ϕ_k è totale e quindi $\phi_k(k)$ è un valore definito, che 1 = 0
- Dobbiamo respingere l'ipotesi che f sia nella sequenza, ossia dobbiamo ammettere che esistono funzioni computabili totali che NON sono in RP

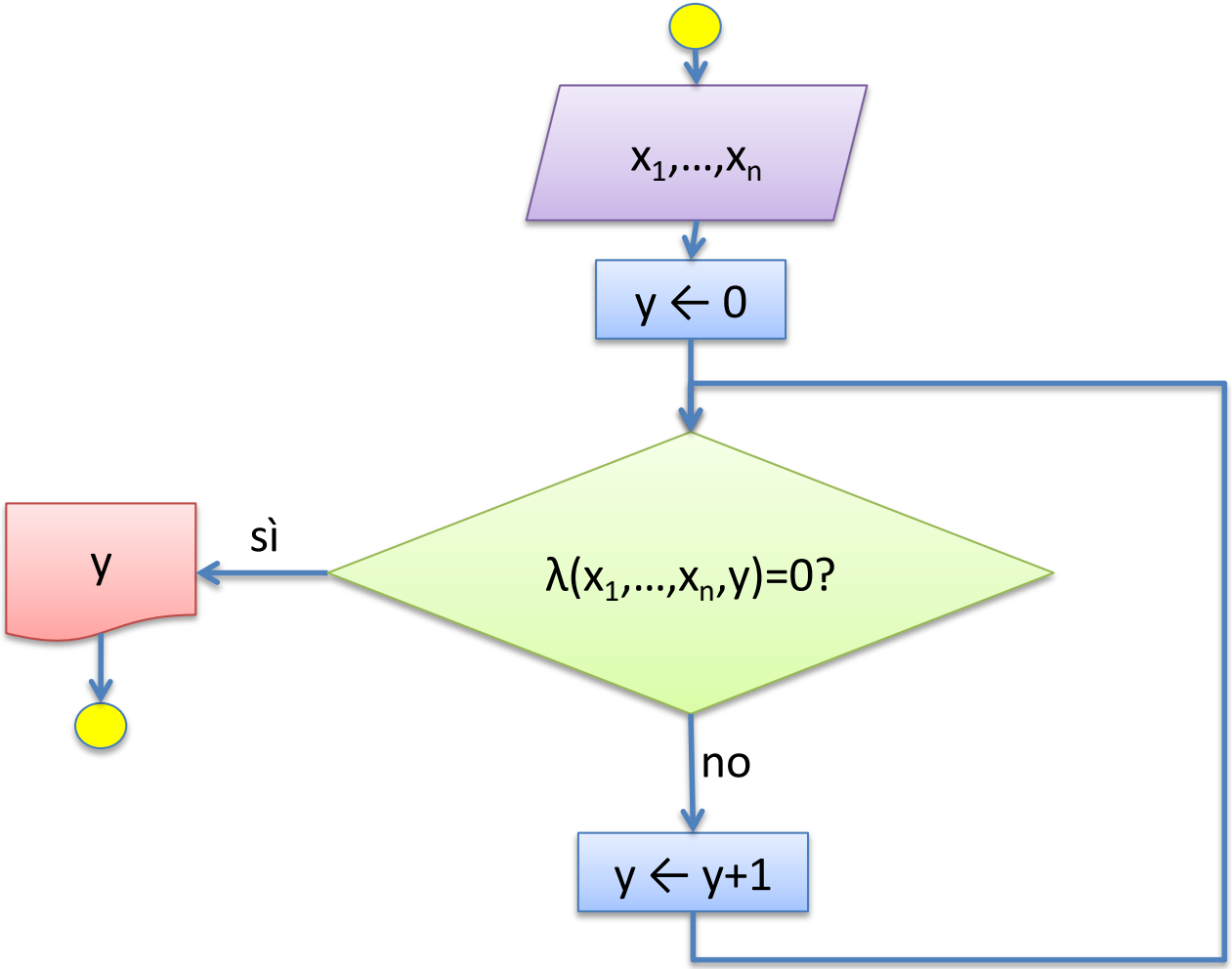
Nuova operazione: minimizzazione

- Operazione di minimizzazione:
  - data $\lambda:N^{n+1}\rightarrow N$
  - si ottiene per minimizzazione la funzione $\phi:N^{n}\rightarrow N$ quando

    $\phi(x_1,\ldots,x_n) = \mu y(\lambda(x_1,\ldots,x_n,y)=0)$,

    dove $\mu y(P)$ vuol dire “il più piccolo y tale che P”
- Funzioni definite tramite la minimizzazione possono essere parziali, perché, dati $x_1,\ldots,x_n$, non è garantita l’esistenza di un y tale che $\lambda(x_1,\ldots,x_n,y)=0$

La minimizzazione conserva la computabilità delle funzioni



Funzioni ricorsive generali (RG)

- Si dice ricorsiva generale una funzione che si ottiene dalle funzioni base applicando un numero finito di volte le operazioni di composizione, ricorsione, e minimizzazione
- RG è linsieme di tutte e sole le funzioni ricorsive generali
- Una derivazione in RG è una sequenza di funzioni, ciascuna delle quali è una funzione base, oppure è stata ottenuta dalle funzioni precedenti tramite composizione, ricorsione, o minimizzazione
- L'ultima funzione in una derivazione in RG si dice RG-derivabile
- Una fuzione è in RG $\Leftrightarrow$ è RG-derivabile
- Esempio di funzione in RG:

$$\phi(x,y)=\begin{cases} x/y, \text{ se } x \text{ è multiplo di } y \\ \perp, \text{ altrimenti} \end{cases}$$

1. $\phi_1(x) = pr(x)$ [funzione predecessore, definita per ricorsione da $z(x)$ e $P^1_1(x)$]
2. $\phi_2(x,y) = x \dot{-} y$ [differenza simmetrica, definita per ricorsione da $P^1_1(x)$ e $pr(x)$]
3. $\phi_3(x,y) = dist(x,y) = x \dot{-} y + y \dot{-} x$ [distanza, definita per composizione da + e $\dot{-}$]
4. $\phi_4(x,y) = \mu z(dist(z*y, x) = 0)$ [definita per minimizzazione da una composizione di * e dist]

- ϕ_4 è la funzione ϕ, ed è stata derivata da funzioni RP con operazioni di composizione, ricorsione, e minimizzazione, quindi è in RG

Dimostrazione

- $\phi(x,y)$= x/y, se x è multiplo di y
  $\perp$, altrimenti

1. $\phi_1(x) = pr(x)$ [funzione predecessore, definita per ricorsione da z(x) e $P^1_1(x)$]

$pr(0) = z(x)$
$pr(s(x)) = P^1_1(x)$

pr(x) è in RP (E QUINDI anche in RG)

2. $\phi_2(x,y) = x \dot{-} y$ [differenza simmetrica, definita per ricorsione da $P^1_1(x)$ e pr(x)]

$x \dot{-} 0 = P^1_1(x)$

$x \dot{-} s(y) = pr(x \dot{-} y)$

$\dot{-}$è in RP (E QUINDI anche in RG)

3. $\phi_3(x,y) = dist(x,y) = x \dot{-} y + y \dot{-} x$ [distanza, definita per composizione da + e $\dot{-}$]

dist è in RP (E QUINDI anche in RG)

4. $\phi_4(x,y) = \mu z(dist(z*y, x) = 0)$ [definita per minimizzazione da una composizione di * e dist]

quindi, ϕ_4 appartiene a RG (ϕ_4 coincide con la funzione ϕ)

$x * 0 = z(x)$
$x * s(y) = +(x,(x*y))$ [anche al moltiplicazione è in RP]

Enumerazione delle RG

- In questo caso non si conclude nulla sull'appartenenza di $f(x) = \phi_x(x)+1$ a RG o meno, perché l'uguaglianza $\phi_k(k)+1 = \phi_k(k)$ non porta ad alcun assurdo
- Semplicemente, dobbiamo ammettere che ϕ_k, se appartiene alla sequenza delle RG, non sia definita in k, il che è compatibile con la parzialità delle RG
- Il tutto è compatibile anche con l'ipotesi che f NON sia nella sequenza delle RG

Teorema (Turing, 1937)

- Una funzione f è RG ⇔ f è T-computabile
- (⇒)
  - si dimostra che le funzioni base sono T-computabili
  - e che le operazioni di composizione, ricorsione, e minimizzazione conservano la T-computabilità

La funzione zero è T-computabile

- La seguente macchina di Turing MTz infatti computa la funzione zero:

  q_0 I [ ] D q_0

  q_0 [ ] I C q_f

La funzione successore è T-computabile

- La seguente macchina di Turing MTs infatti computa la funzione successore:

  q_0 I I D q_0

  q_0 [ ] I C q_0

Le funzioni proiezione sono T-computabili

- La seguente macchina di Turing MTp^3_1, ad esempio, computa la funzione $P^3_1(x,y,z) = x$:

  q_1 I I D q_1

  q_1 s_0 s_0 D q_2

  q_2 I s_0 D q_2

  q_2 s_0 s_0 D q_3

  q_3 I s_0 D q_3

  q_3 s_0 s_0 C q_0

La composizione conserva la T-computabilità

- $\phi(x) = \chi(\psi(x))$
- Se χ e ψ sono T-computabili, lo è anche ϕ?
- Ossia: se esistono MTχ e MTψ, si riesce a costruire MTϕ?
- Sì: basta aggiungere le istruzioni di MTχ a quelle di MTψ, con i seguenti accorgimenti:
  - rinominare gli stati interni di MTχ per evitare sovrapposizione con gli stati di MTψ
  - aggiungere istruzioni che portino la testina della macchina nella posizione standard dopo che sono state eseguite le istruzioni di MTψ e prima che vengano eseguite quelle di MTχ

La ricorsione conserva la T-computabilità

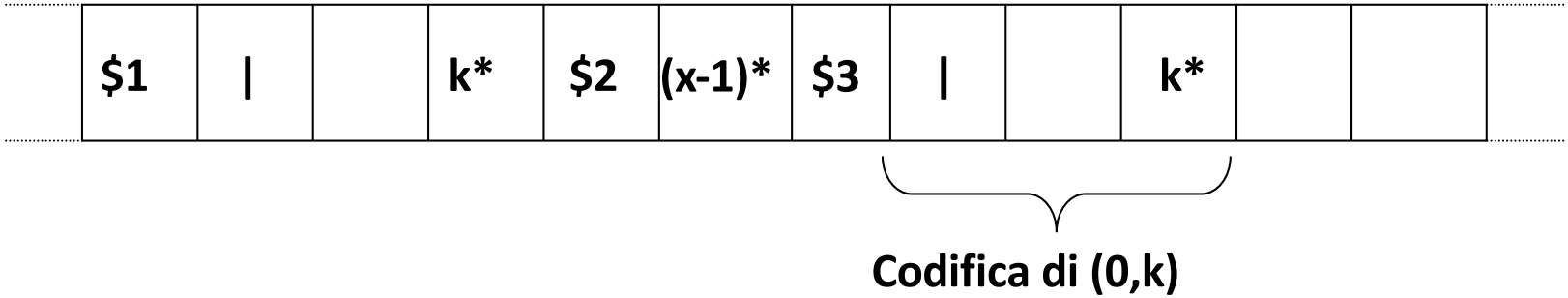
$$\begin{cases} \phi(0) = k \\ \phi(s(x)) = \psi(x,\phi(x)) \end{cases}$$

- Se ψ è T-computabile, lo è anche ϕ?
- Ossia: avendo a disposizione MT_ψ, si riesce a costruire MT_ϕ?
- Sì: viene mostrato nelle seguenti slide
- L'alfabeto include i simboli \$1, \$2, \$3 che sono usati come separatori

La ricorsione con MT

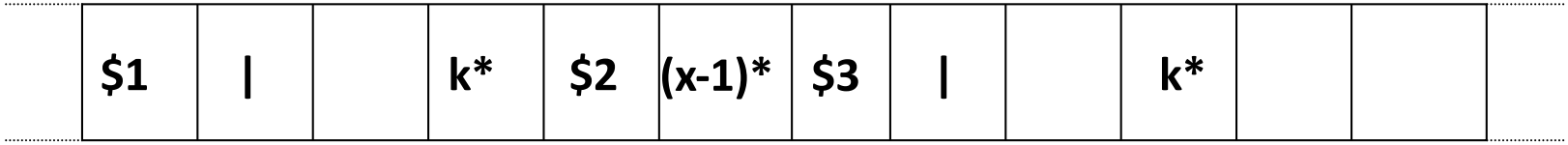
- Per calcolare $\phi(x)$, MT_ϕ inizia scrivendo sul nastro (k* rappresenta k+1 barre):

- MT_ϕ cancella una barra da x*
- Se ci sono zero barre tra $2 e $3, l'output è tra $1 e $2 (perché $\phi(0) = k$)
- Altrimenti MT_ϕ configura il nastro così:

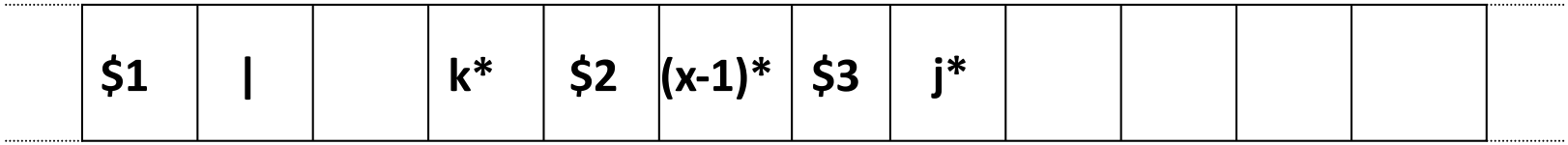


La ricorsione con MT

- MT_{ϕ} esegue il programma di MT_{ψ} sulla parte a destra di $3:



ottenendo:

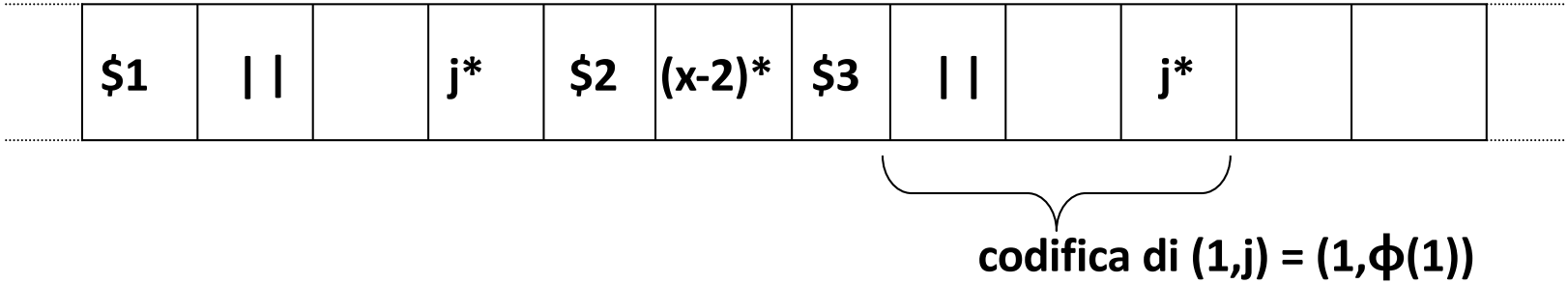


dove j* è la codifica di $j = \psi(0,k) = \psi(0,\phi(0)) = \phi(1)$

La ricorsione con MT

- MT_{ϕ} cancella una barra tra \$2 e \$3, se non ci sono più barre vuol dire che x=1 e quindi l'output è $j = \psi(0,k) = \phi(1)$, che si trova a destra di \$3:

- Altrimenti nuova configurazione e si ripete:



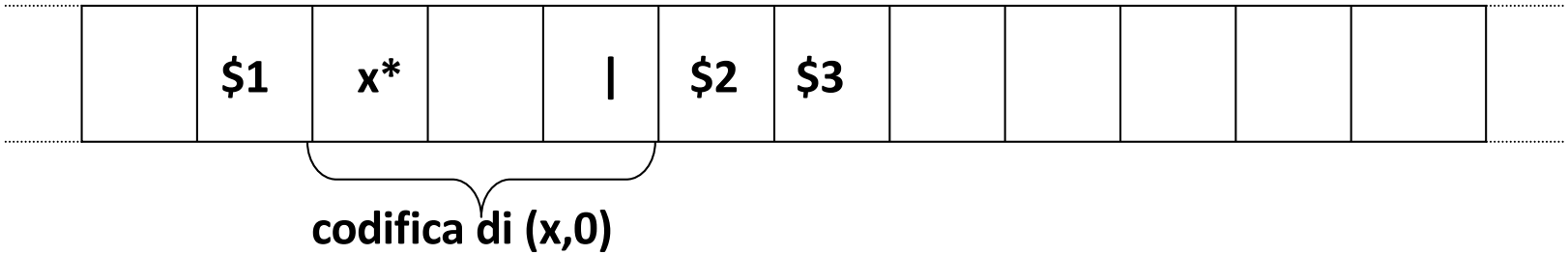
- Eseguendo MT_{ψ} a destra di \$3 si calcola $\psi(1,\phi(1)) = \phi(2)$...e così via fino a $\phi(x)$

La minimizzazione conserva la T-computabilità

- $\phi(x)$ = il più piccolo y: $\lambda(x,y)=0$
- Dobbiamo costruire MT_ϕ usando MT_λ
- MT_ϕ procede eseguendo iterativamente il codice di MTλ per calcolare $\lambda(x,0)$, $\lambda(x,1)$, $\lambda(x,2),\ldots,\lambda(x,y)$ e restituisce in output il primo y per cui $\lambda(x,y)=0$
- Se tale y non esiste MT_ϕ non si ferma mai (e infatti ϕ non è definita per quella x)

La minimizzazione con MT

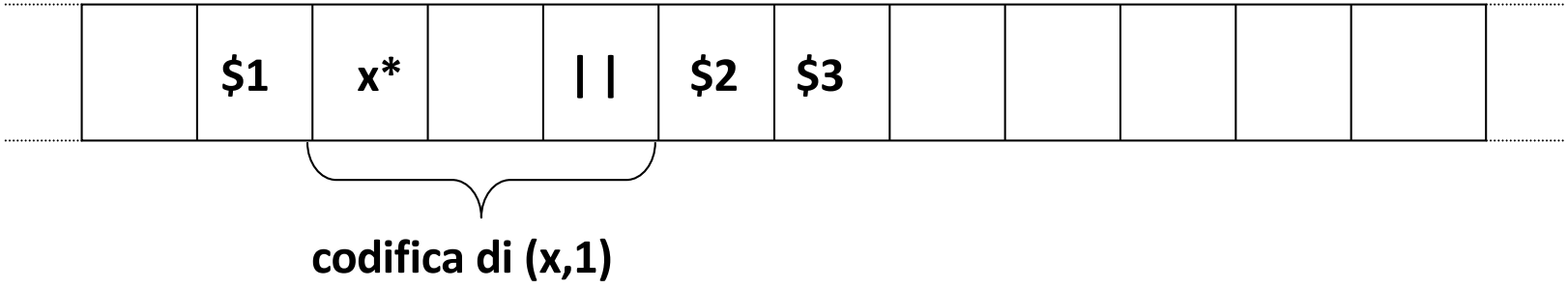
- MT_{ϕ} configura inizialmente il nastro così:



- Poi copia i dati tra $1 e $2 nello spazio tra $2 e $3 e lì usa il codice di MT_{λ} per calcolare λ(x,0)
- Se il risultato è zero cancella tutto il resto e lo lascia come output

La minimizzazione con MT

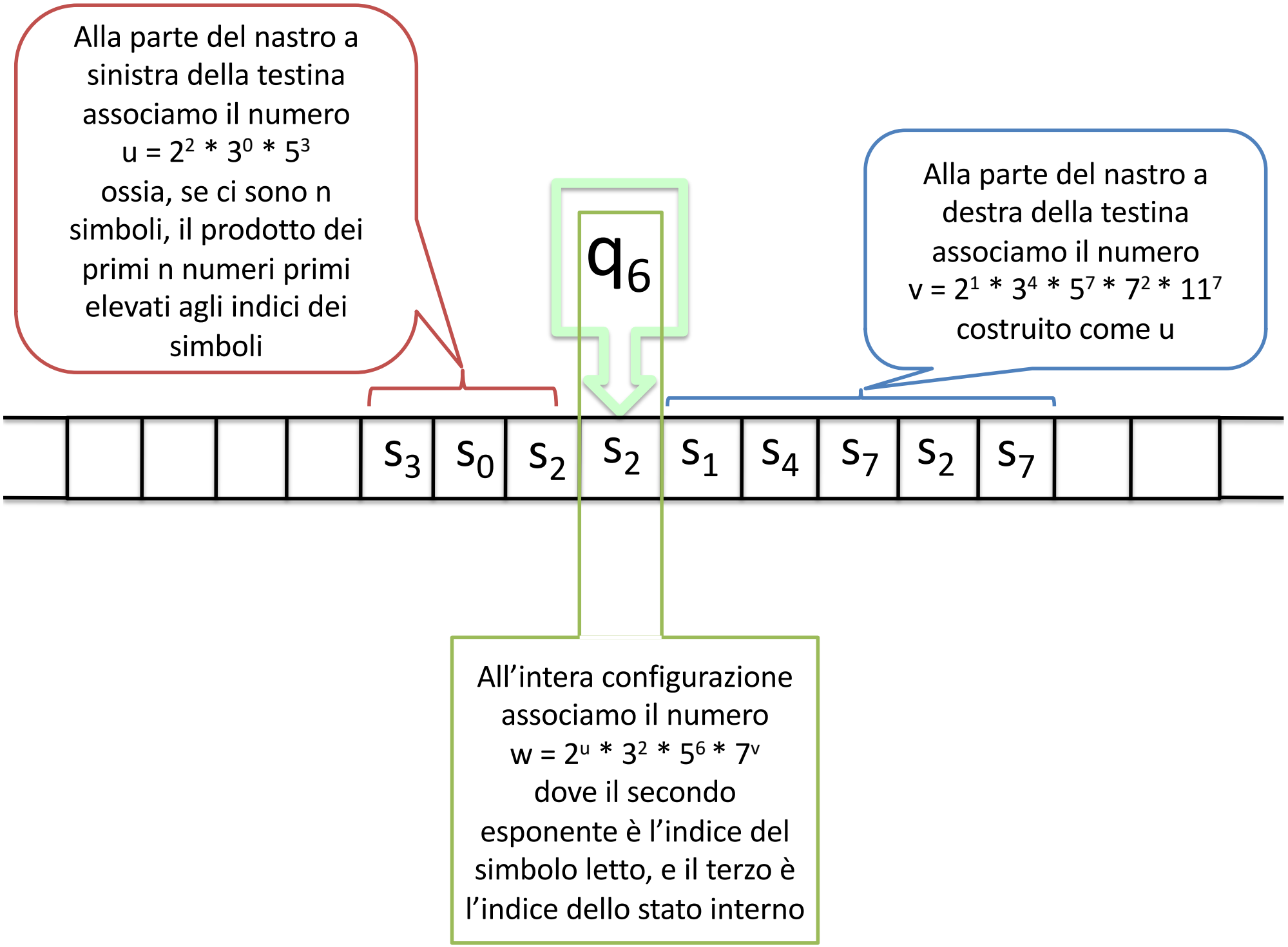
- Altrimenti MT_ϕ riconfigura il nastro così:



  e ripete tutto per calcolare λ(x,1)
- ...e così via fino a trovare (eventualmente) il primo y per cui λ(x,y)=0

Gödelizzazione delle configurazioni di una MT

- Prima di procedere alla dimostrazione (sommaria) di ($\Leftarrow$), introduciamo delle codifiche elaborate da Gödel che permettono di esprimere le computazioni delle MT come applicazioni di funzioni matematiche


Alla parte del nastro a sinistra della testina associamo il numero
$u = 2^2 * 3^0 * 5^3$
ossia, se ci sono n simboli, il prodotto dei primi n numeri primi elevati agli indici dei simboli
q_6
Alla parte del nastro a destra della testina associamo il numero
$v = 2^1 * 3^4 * 5^7 * 7^2 * 11^7$
costruito come u
s_3 s_0 s_2 s_2 s_1 s_4 s_7 s_2 s_7
All'intera configurazione associamo il numero
$w = 2^u * 3^2 * 5^6 * 7^v$
dove il secondo esponente è l'indice del simbolo letto, e il terzo è l'indice dello stato interno

Il programma di MT come funzione

- L'indice w è associato alla configurazione della MT in maniera univoca, grazie all'unicità della scomposizione di un numero in fattori primi
- Per mezzo delle sue istruzioni, la MT passa da una configurazione all'altra in modo deterministico: da w la MT non può fare altro che passare a w' (a meno che w non sia una configurazione finale)
- Ad ogni MT possiamo quindi associare una funzione:

$$\rho_{MT}(w)=\begin{cases} w' \text{ se } w \text{ è la codifica di una} \\ \quad \text{configurazione non finale} \\ w \quad \text{altrimenti} \end{cases}$$

Il determinismo di MT garantisce che ρ_{MT} sia una funzione

Si può dimostrare che qualunque sia la MT, ρ_{MT} è in RP

La computazione di un numero finito di passi con MT come funzione

- A partire da ρ_{MT}, definiamo θ_{MT} per ricorsione:

$$\begin{cases} \theta_{MT}(w,0) = w \\ \theta_{MT}(w,s(z)) = \rho_{MT}(\theta_{MT}(w,z)) \end{cases}$$

- Se w è la codifica di una configurazione w, $\theta_{MT}(w,z)$ è la codifica della configurazione che si ottiene dopo z istruzioni a partire da w
- Anche θ_{MT} è RP, perché definita per ricorsione da una funzione RP

f è T-computabile ⇒ f è RG

- Sia MT_f la MT che computa f
- Dato l'input x, lo si codifichi sul nastro di MT_f
- La computazione di MT_f è data dalla funzione RP $\theta_{MTf}(w,z)$, e con un'operazione di minimizzazione si ottiene (se esiste) il minimo numero z_0 di passi per cui, a partire dalla codifica della configurazione di partenza w_1, si ottiene una configurazione finale
- Calcolando $\theta_{MTf}(w_1,z_0)$ si ottiene la codifica della configurazione finale, e invertendo la gödelizzazione, si ricava la codifica dell'output sul nastro
- Decodificando l'output, si ottiene f(x)
- f è stata ottenuta componendo tutti i calcoli di cui sopra, fatti con funzioni RP e con una minimizzazione su una funzione RP, quindi f è RG

In conclusione:

- Teorema di Turing:
  Una funzione f è RG ⇔ f è T-computabile
- Tesi di Church:
  Una funzione è computabile ⇔ essa è RG
- Tesi di Church-Turing:

  Una funzione è computabile ⇔
  essa è T-computabile