Automi a pila e macchina di Turing

Nicola Caravaggi, Tommaso Dossi, Alessandro Fenu, Matteo Gori, Tommaso Lunghi, Lorenzo Picinelli e Carlo Rotolo

17 febbraio 2022

Indice

Indice

La pila

Prima di poter parlare di automi a pila è necessario introdurre questa semplice struttura dati. La pila, o stack, supporta le seguenti operazioni:

- Inserimento di un elemento in cima
- Lettura, modifica e rimozione dell'elemento in cima

La pila ha dimensione idealmente illimitata. Un automa a pila, o PDA, (dall'inglese Pushdown Automaton) consiste di un automa a stati finiti e una pila, in cui le transizioni coinvolgono anche quest'ultima.

Un esempio informale

Consideriamo il linguaggio $L = ww^R, w \in \Sigma^*$, formato dai palindromi di lunghezza pari. Non essendo un linguaggio regolare, non può essere riconosciuto da un normale automa a stati finiti, tuttavia un automa a pila è in grado di riconoscerlo, vediamo come.

Definizione formale

In termini formali, un PDA è definito da 7 elementi:
$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, dove:

- Q è l'insieme degli stati
- Σ è l'alfabeto dei simboli presi in input
- Γ è l'alfabeto, distinto da Σ, dei simboli dello stack
- δ è la funzione di transizione
- q_0 è lo stato iniziale
- Z_0 è l'unico simbolo inizialmente presente nello stack
- F è l'insieme degli stati finali

Le transizioni

A differenza di quanto succede in DFA e NFA, la funzione di transizione $\delta(q, \alpha, Z)$ ha 3 argomenti, dove:

- q è lo stato attuale
- $\alpha \in \Sigma \cup \{\epsilon\}$ è il simbolo in ingresso o ϵ.
- Z è il simbolo attualmente in cima allo stack Il risultato di una transizione è una o più coppie della forma (p, Y), dove p è uno stato e $Y \in \Gamma^*$ è la stringa di simboli che rimpiazza X in cima allo stack.

Descrizione istantanea

Per rappresentare meglio lo "stato" di un PDA, è utile introdurre una descrizione che contiene, oltre allo stato nell'automa finito, anche il contenuto dello stack e l'input rimanente.
La descrizione istantanea, o ID, di un PDA è quindi una terna q, w, γ, dove:

- q è lo stato
- $w \in \Sigma^*$ è l'input rimanente
- $\gamma \in \Gamma^*$ è il contenuto dello stack

Mosse nei PDA

Come negli automi a stati finiti, nei PDA esistono delle mosse. Supponiamo di avere $(p, \alpha) \in \delta(q, a, X)$, allora una mossa sarà:

$$(q, aw, X\beta) \vdash (p, w, \alpha\beta)$$

Dove il simbolo $\vdash$ indica che la seconda ID è raggiungibile dalla prima con una mossa. Inoltre, $\vdash^*$ la raggiungibilità di un'ID da un'altra tramite zero o più mosse, e la sequenza di ID su cui avvengono le mosse è detta *computazione*.

Inoltre, se $(q, x, \alpha) \vdash^* (p, y, \beta)$, allora:

$$(q, xw, \alpha\gamma) \vdash^* (p, yw, \beta\gamma) \forall w \in \Sigma^*, \gamma \in \Gamma^*$$

Questo risultato segue dal fatto che le mosse fatte non alterano né w né γ. Dei casi particolari di questo teorema sono due importanti principi per i PDA, che si ottengono rispettivamente sostituendo $\gamma = \epsilon$ e $w = \epsilon$, o, in modo meno formale:

- Se una computazione è valida in un PDA, allora lo è anche quella che si ottiene accodando la stessa scritta all'input di ogni ID
- Se una computazione è valida in un PDA, allora lo è anche quella che si ottiene aggiungendo gli stessi simboli in fondo allo stack di ogni ID

Un terzo principio afferma invece che se l'ultima ID di una computazione ha come input residuo una stringa non vuota, è possibile rimuoverla dall'input residuo di ogni ID della computazione e ottenere un'altra computazione valida, o, formalmente:
Se $(p, xw, \alpha) \vdash^* (q, yw, \beta)$, allora $(p, x, \alpha) \vdash^* (q, y, \beta)$.

Indice

Linguaggi accettati da un PDA

Un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ può accettare un linguaggio L in uno dei due seguenti modi:

- **Accettazione per stato finale:** per uno stato $q \in F$ e una stringa α, $L(P) = \{w | (q_0, w, Z_0) \vdash^* (q, \epsilon, \alpha)\}$, l'automa consuma la stringa w e arriva in uno stato finale q, non importa cosa rimane nello stack
- **Accettazione per stack vuoto:** per un qualsiasi stato $q \in Q$, $N(P) = \{w | (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)\}$, l'automa consuma la stringa w e svuota lo stack, non importa in quale stato si trovi

L'accettazione per stato finale e l'accettazione per stack vuoto sono equivalenti

Equivalenza tra le due accettazioni

In generale dato un PDA, i linguaggi che riconosce per stato finale sono diversi dai linguaggi che riconosce per stack vuoto

Dato un linguaggio L, esiste un PDA che riconosce L per stato finale se e solo se esiste un PDA che riconosce L per stack vuoto

Cosa significa che l'accettazione per stato finale e per stack vuoto sono equivalenti?

Da stack vuoto a stato finale

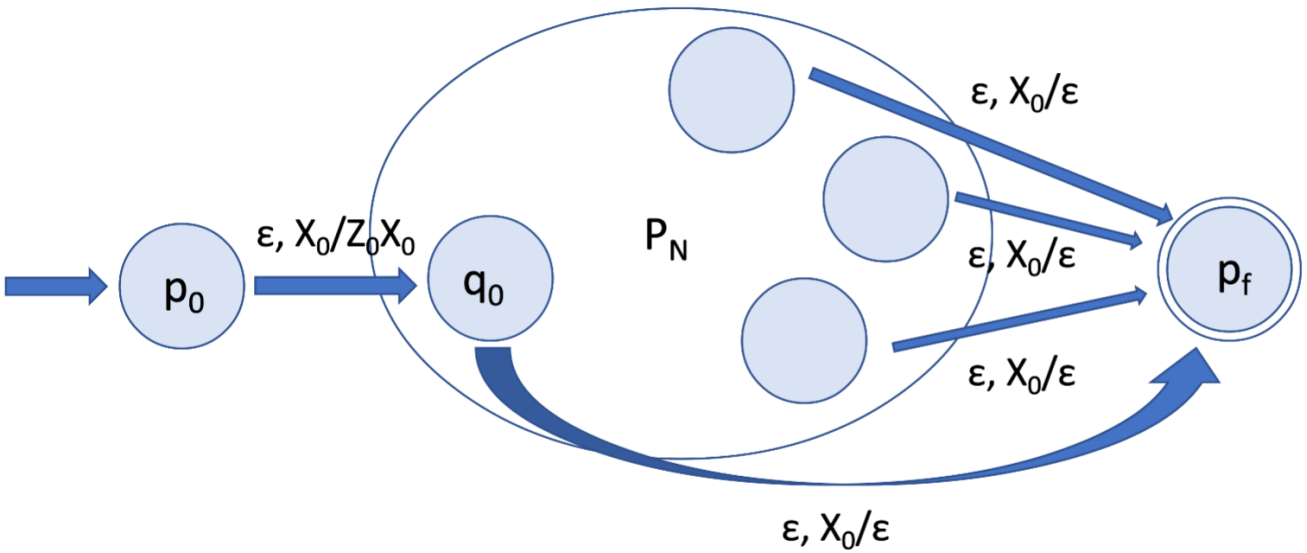
Sia $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$ un automa che riconosce il linguaggio L per stack vuoto, allora costruiamo un automa P_F che riconosce L per stato finale:

$$P_F = (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$$

In cui la funzione di transizione δ_F è definita come

1. $\delta_F(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$
2. $\delta_N(q, a, Y) \subseteq \delta_F(q, a, Y)$ per ogni $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$, $Y \in \Gamma$ e $(p_f, \epsilon) \in \delta_F(q, \epsilon, X_0)$ per ogni $q \in Q$

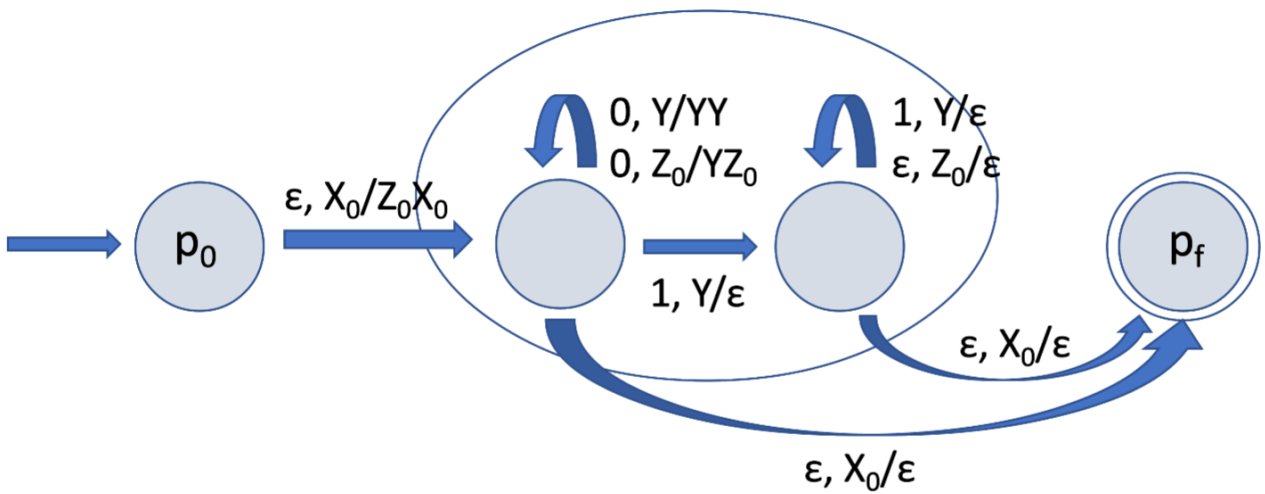
Da stack vuoto a stack finale



Abbiamo quindi mostrato che per ogni linguaggio L riconosciuto da un PDA per stack vuoto è possibile costruire un altro PDA che lo riconosce per stato finale

Esempio

Consideriamo il linguaggio $L = \{w \in \{0,1\}^* | w = 0^n 1^n, n \geq 1\}$, dato un PDA che lo riconosce per stack vuoto costruiamo $P_F = (Q \cup \{p_0, p_f\}, \{0,1\}, \{X_0, Z_0, Y\}, \delta_F, p_0, X_0, p_f)$ che lo riconosce per stato finale



Da stato finale a stack vuoto

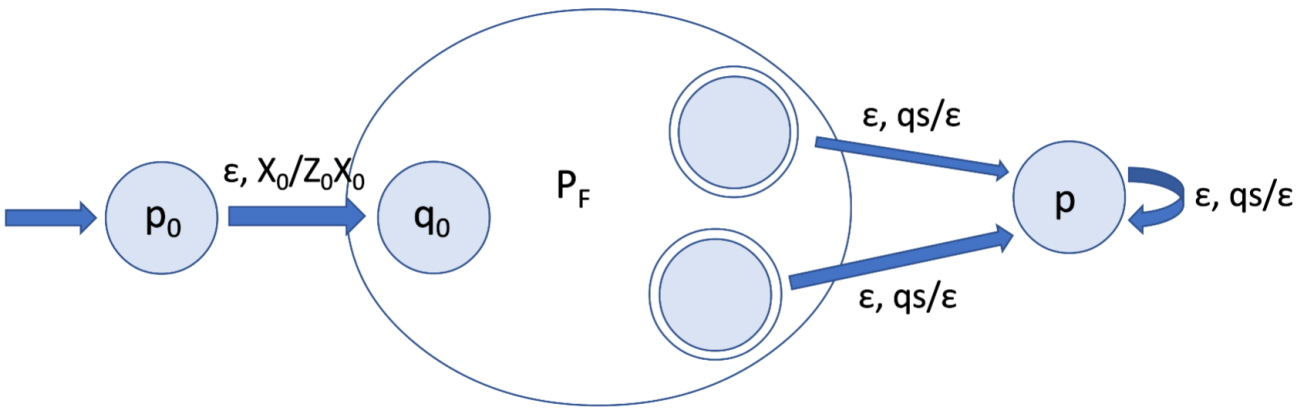
Sia $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$ un automa che riconosce il linguaggio L per stato finale, allora costruiamo un automa P_N che riconosce L per stack vuoto:

$$P_N = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$$

In cui la funzione di transizione δ_N è definita come

1. $\delta_N(p_0, \epsilon, X_0) = \{(q_0, Z_0 X_0)\}$
2. $\delta_F(q, a, Y) \subseteq \delta_N(q, a, Y)$ per ogni $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$, $Y \in \Gamma$
3. $(p, \epsilon) \in \delta_N(q, \epsilon, Y)$ per ogni $q \in F$, $Y \in \Gamma \cup \{X_0\}$
4. $\delta_N(p, \epsilon, Y) = \{(p, \epsilon)\}$ per ogni $Y \in \Gamma \cup \{X_0\}$

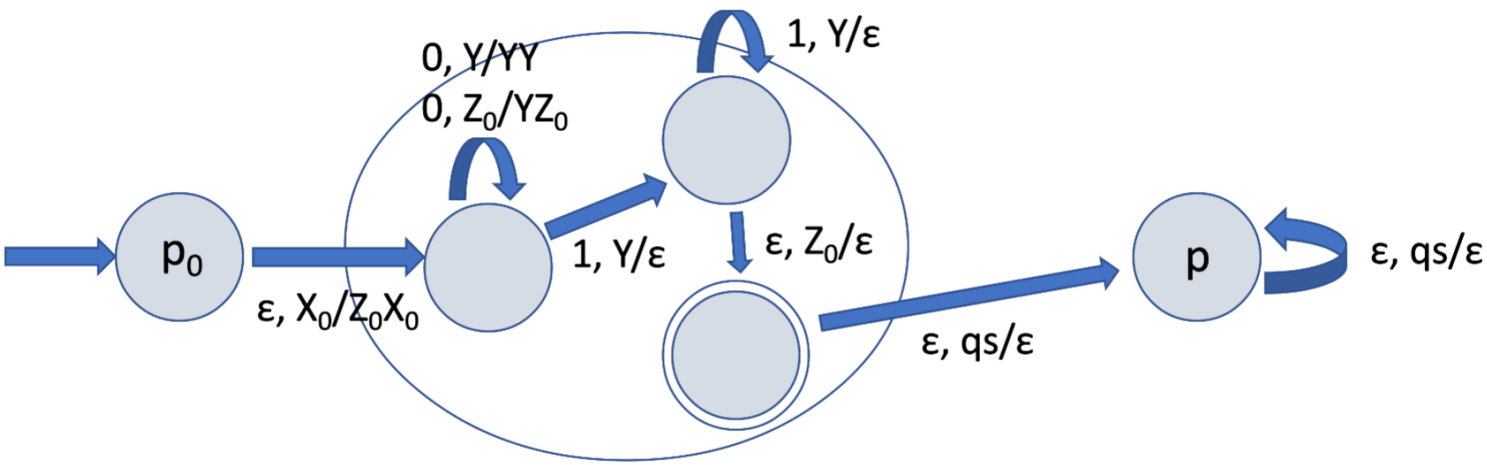
Da stato finale a stack vuoto



Abbiamo quindi mostrato che per ogni linguaggio L riconosciuto da un PDA per stato finale è possibile costruire un altro PDA che lo riconosce per stack vuoto

Esempio

Consideriamo il linguaggio $L = \{w \in \{0,1\}^* | w = 0^n 1^n, n \geq 1\}$, dato un PDA che lo riconosce per stato finale costruiamo $P_N = (Q \cup \{p_0, p\}, \{0,1\}, \{X_0, Z_0, Y\}, \delta_N, p_0, X_0)$ che lo riconosce per stack vuoto ("qs" indica un qualsiasi simbolo nello stack)



Indice

Equivalenza fra PDA e CFG

Adesso siamo pronti a descrivere la classe dei linguaggi accettati dai PDA. Nelle prossime slide dimostreremo infatti che coincidono con i linguaggi liberi dal contesto.
Per farlo sarà sufficiente mostrare le due implicazioni:

- data una CFG generica, è possibile costruire un PDA che accetti per stack vuoto lo stesso linguaggio;
- dato un PDA P generico, è possibile costruire una CFG che generi il linguaggio accettato per stack vuoto da P;

Dalle grammatiche ai PDA

Sia $G = (V, T, Q, S)$ una grammatica libera dal contesto; cercheremo di simulare le produzioni di G costruendo un PDA P con alfabeto di input T e alfabeto di stack $T \cup V$ che operi nel seguente modo:

- Se il simbolo in cima allo stack è una variabile, viene sviluppata seguendo una qualunque produzione di G;
- Se il simbolo in cima allo stack è un simbolo terminale, l'automa lo confronta col simbolo ricevuto da input; se sono uguali, prosegue nel riconoscimento della stringa eliminando la testa dello stack, altrimenti abbandona la diramazione;

Dalle grammatiche ai PDA

Formalmente, l'automa sarà così definito:

$$P = (\{q\}, T, T \cup V, \delta, q, S)$$

Dove la funzione di transizione è definita dalle regole

- Per ogni $A \in V$, $\delta(q, \varepsilon, A) = \{(q, \beta) | A \to \beta \in Q\}$
- Per ogni $a \in T$, $\delta(q, a, a) = \{(q, \varepsilon)\}$

Dalle grammatiche ai PDA

Il riconoscimento di una stringa w nel PDA procederà dunque nel seguente modo:

Dalle grammatiche ai PDA

Il riconoscimento di una stringa w nel PDA procederà dunque nel seguente modo:

- Per ogni produzione di G del tipo $S \to xA\alpha$, dove x è composta da soli terminali, applicando δ viene modificato lo stack sostituendo S con $xA\alpha$;

Dalle grammatiche ai PDA

Il riconoscimento di una stringa w nel PDA procederà dunque nel seguente modo:

- Per ogni produzione di G del tipo $S \rightarrow xA\alpha$, dove x è composta da soli terminali, applicando δ viene modificato lo stack sostituendo S con $xA\alpha$;
- Vengono confrontati i simboli in input con i terminali di x; se coincidono, applicando δ viene eliminato il terminale dallo stack, altrimenti il riconoscimento si ferma e viene abbandonato il ramo;

Dalle grammatiche ai PDA

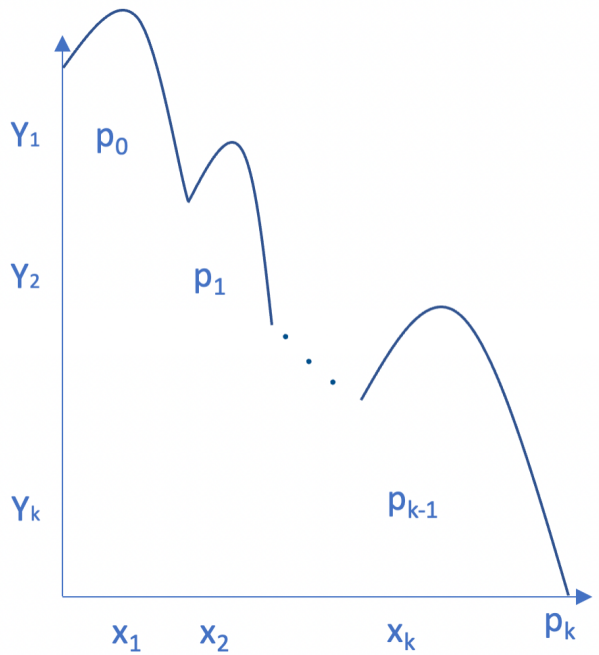
Il riconoscimento di una stringa w nel PDA procederà dunque nel seguente modo:

- Per ogni produzione di G del tipo $S \to xA\alpha$, dove x è composta da soli terminali, applicando δ viene modificato lo stack sostituendo S con $xA\alpha$;
- Vengono confrontati i simboli in input con i terminali di x; se coincidono, applicando δ viene eliminato il terminale dallo stack, altrimenti il riconoscimento si ferma e viene abbandonato il ramo;
- Se il riconoscimento prosegue fino a consumare per intero x, si replica quanto fatto in precedenza con tutte le produzioni del tipo $A \to x'A'\alpha'$. La stringa sarà accettata se in qualche diramazione lo stack risulterà a un certo punto completamente vuoto.

Dai PDA alle grammatiche

Concludiamo ora mostrando una costruzione esplicita di una CFG che generi lo stesso linguaggio riconosciuto per stack vuoto da un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$.

Nel grafico è rappresentato il riconoscimento di una stringa in P.



Dai PDA alle grammatiche

Definiamo le variabili di G simulando il riconoscimento:

- Includiamo per prima cosa il simbolo iniziale S.
- Per ogni $p, q \in Q$, $X \in \Gamma$, rappresentiamo l'eliminazione di X dallo stack con conseguente passaggio dallo stato p allo stato q con la variabile $[pXq]$.

Dai PDA alle grammatiche

Definiamo ora le produzioni di G:

- Per ogni stato p, includiamo la produzione $S \to [q_0 Z_0 p]$;
- Se $\delta(q, a, X)$ contiene $(r, Y_1 Y_2 \dots Y_k)$ allora includiamo per ogni k-upla di stati $r_1, \dots r_k$, la produzione $[qXr_k] \to a[rY_1 r_1][r_1 Y_2 r_2] \dots [r_{k-1} Y_k r_k]$.

Dai PDA alle grammatiche

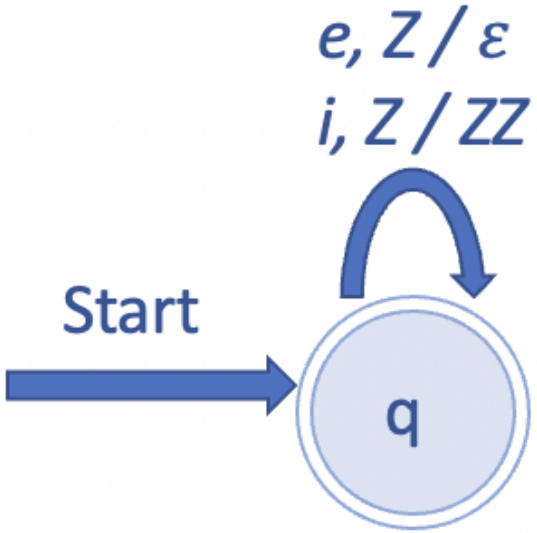
Definiamo ora le produzioni di G:

- Per ogni stato p, includiamo la produzione $S \to [q_0 Z_0 p]$;
- Se $\delta(q, a, X)$ contiene $(r, Y_1 Y_2 \dots Y_k)$ allora includiamo per ogni k-upla di stati $r_1, \dots r_k$, la produzione $[qXr_k] \to a[rY_1 r_1][r_1 Y_2 r_2] \dots [r_{k-1} Y_k r_k]$.

Il secondo tipo di produzione simula l'eliminazione del simbolo X dallo stack dopo la lettura da input di a, con conseguente passaggio da q a r_k. In questo modo G è in grado di generare, a partire da S, tutte le stringhe che portano a vuotare lo stack di P.

Esempio

Convertiamo in una grammatica libera dal contesto il seguente automa a pila, che accetta stringhe di if ed else sbilanciate a causa di un else finale in più



Esempio

Costruiamo le variabili di G:

- Includiamo il simbolo iniziale S, come descritto nella costruzione mostrata;
- Poiché l'automa ha un solo stato q e un solo simbolo di stack Z, l'unica altra variabile sarà $[qZq]$.

Esempio

Costruiamo le variabili di G:

- Includiamo il simbolo iniziale S, come descritto nella costruzione mostrata;
- Poiché l'automa ha un solo stato q e un solo simbolo di stack Z, l'unica altra variabile sarà $[qZq]$.

Le produzioni saranno invece:

- $S \to [qZq]$
- $[qZq] \to i[qZq][qZq]$
- $[qZq] \to e$

Semplificando quanto ottenuto, ricaviamo la grammatica
$S \to iSS \mid e$

Indice

PDA deterministici

Una classe importante di PDA è data da quelli deterministici, che possiamo presentare informalmente come la classe dei PDA in cui a ogni punto del riconoscimento non c'è alcuna possibilità di scelta fra mosse diverse.

PDA deterministici

Una classe importante di PDA è data da quelli deterministici, che possiamo presentare informalmente come la classe dei PDA in cui a ogni punto del riconoscimento non c'è alcuna possibilità di scelta fra mosse diverse.

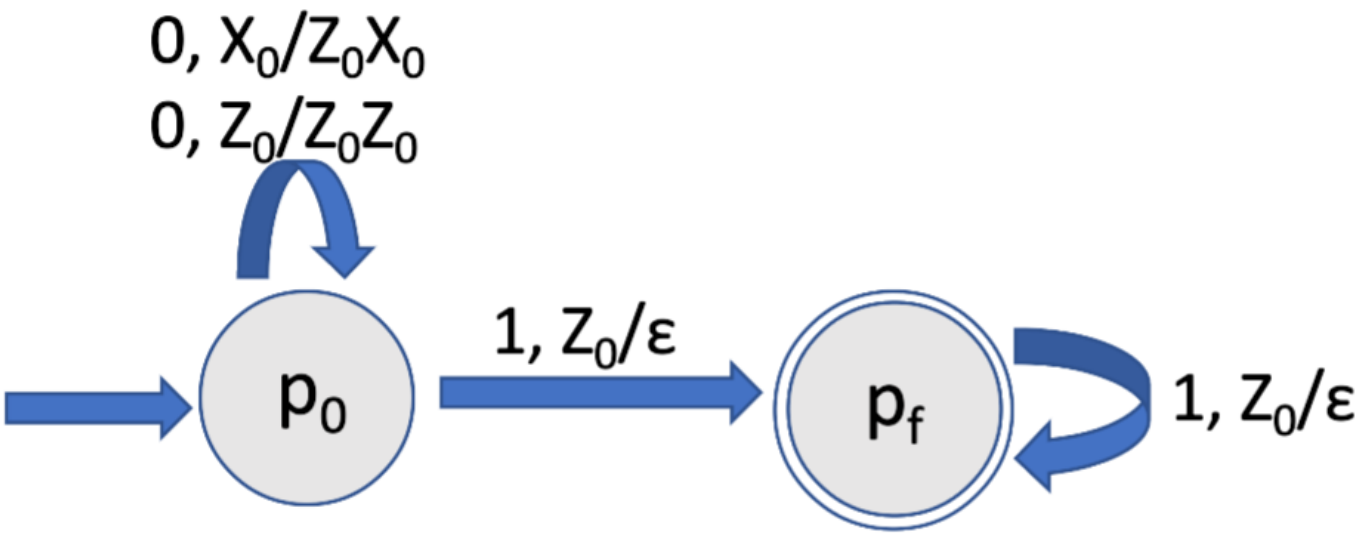
Definizione

Un PDA si dice *deterministico* se

- $\delta(q, a, X)$ ha al più un elemento per ogni $q \in Q$, $a \in \Sigma$ o $a = \varepsilon$, $X \in \Gamma$
- Se $\delta(q, a, X)$, con $a \in \Sigma$ contiene un elemento, allora $\delta(q, \varepsilon, X)$ è vuoto

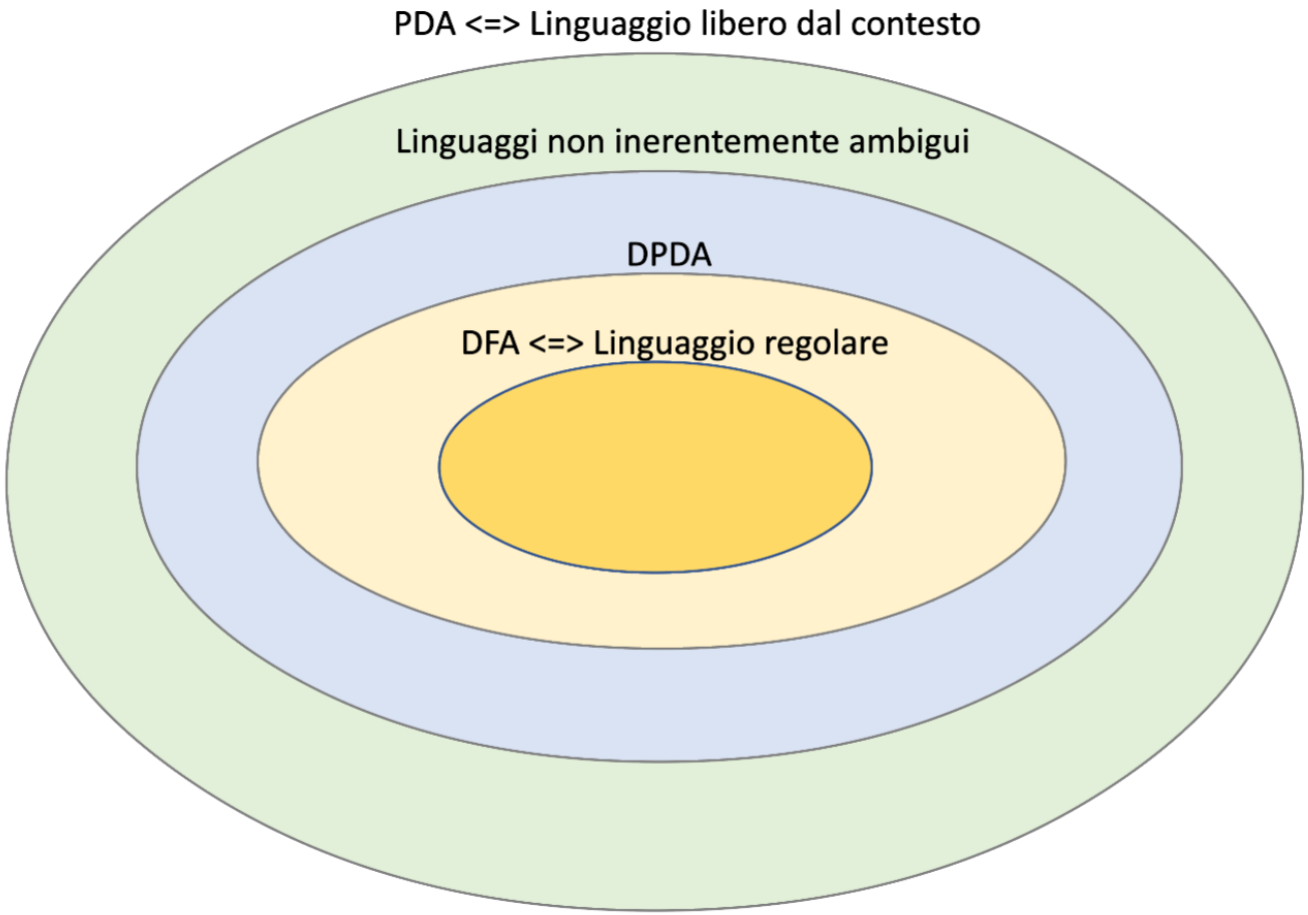
Esempio

Dato il linguaggio $L = \{w \in \{0,1\}^* | w = 0^n 1^m, n \geq m \geq 1\}$, il seguente è un DPDA che riconosce L



Gerarchia dei linguaggi riconosciuti da un DPDA

È possibile dimostrare le seguenti inclusioni tra linguaggi



Indice

Problemi indecidibili

Definizione

Un problema decisionale (ovvero che richiede una risposta `sì/no`) si dice *indecidibile* se non esiste alcun algoritmo che riesca a risolverlo.

Problemi indecidibili

Definizione

Un problema decisionale (ovvero che richiede una risposta `sì/no`) si dice *indecidibile* se non esiste alcun algoritmo che riesca a risolverlo.

N.B: Potrebbero anche esistere algoritmi in grado di risolvere alcuni casi specifici di un problema indecidibile: l'indecidibilità sta a significare che non è possibile trovare un algoritmo **generale**.

Problemi indecidibili

Definizione

Un problema decisionale (ovvero che richiede una risposta `sì/no`) si dice *indecidibile* se non esiste alcun algoritmo che riesca a risolverlo.

N.B: Potrebbero anche esistere algoritmi in grado di risolvere alcuni casi specifici di un problema indecidibile: l'indecidibilità sta a significare che non è possibile trovare un algoritmo **generale**.

Osserviamo subito che, dato che gli input dei problemi decisionali che vedremo sono delle stringhe costruite su un certo alfabeto Σ, i problemi sono in realtà dei linguaggi: il linguaggio di un problema P è

$$L = \{w \in \Sigma^* | \text{ se l'input è } w, \text{ la risposta al problema è } \texttt{sì}\}$$

Esistenza di problemi indecidibili

Problemi che non si possono risolvere esistono, purtroppo, ma non solo.
In effetti, questi costituiscono la maggior parte dei problemi.

Esistenza di problemi indecidibili

Problemi che non si possono risolvere esistono, purtroppo, ma non solo.
In effetti, questi costituiscono la maggior parte dei problemi.

Mostriamo che, anche limitandoci ad una precisa classe di problemi, non tutti sono risolvibili: per farlo, mostriamo che i problemi sono "più numerosi" degli algoritmi che possono risolverli.

Più formalmente, mostriamo che non esiste una funzione iniettiva dall'insieme dei possibili algoritmi all'insieme dei problemi considerati.

Dimostrazione

Sia $\mathbb{N}^{\mathbb{N}}$ l'insieme delle funzioni dai naturali in sé stessi.

Mostreremo che esistono funzioni in questo insieme tali che il problema di calcolarle in ogni valore è indecidibile.

Dimostrazione

Sia $\mathbb{N}^{\mathbb{N}}$ l'insieme delle funzioni dai naturali in sé stessi.

Mostreremo che esistono funzioni in questo insieme tali che il problema di calcolarle in ogni valore è indecidibile.

Sia ora T l'insieme dei file di testo, ovvero delle sequenze di caratteri di lunghezza finita. Chiaramente, i programmi, scritti in un qualche linguaggio di programmazione, che permettono di calcolare le funzioni in $\mathbb{N}^{\mathbb{N}}$ sono un sottoinsieme di T.

Dimostrazione

Usando un po' di teoria sulle cardinalità, otteniamo:

$$|\mathbb{N}^{\mathbb{N}}| \geq |2^{\mathbb{N}}| = |\mathcal{P}(\mathbb{N})|$$

Dimostrazione

Usando un po' di teoria sulle cardinalità, otteniamo:

$$|\mathbb{N}^{\mathbb{N}}| \geq |2^{\mathbb{N}}| = |\mathcal{P}(\mathbb{N})|$$

Sia T_i l'insieme dei file di testo di lunghezza i caratteri per ogni $i \in \mathbb{N}$.
Dato che l'insieme dei caratteri possibili è finito, T_i è finito per ogni i.

Dimostrazione

Usando un po' di teoria sulle cardinalità, otteniamo:

$$|\mathbb{N}^{\mathbb{N}}| \geq |2^{\mathbb{N}}| = |\mathcal{P}(\mathbb{N})|$$

Sia T_i l'insieme dei file di testo di lunghezza i caratteri per ogni $i \in \mathbb{N}$.
Dato che l'insieme dei caratteri possibili è finito, T_i è finito per ogni i.
Quindi, dato che un'unione numerabile di insiemi finiti è numerabile,

$$|T| = \left| \bigcup_{n=0}^{\infty} T_n \right| = |\mathbb{N}|$$

Dimostrazione

Ma allora, sfruttando la nota relazione $|\mathcal{P}(\mathbb{N})| > |\mathbb{N}|$, otteniamo:

$$|\mathbb{N}^{\mathbb{N}}| \geq |\mathcal{P}(\mathbb{N})| > |\mathbb{N}| = |T|$$

E ciò conclude. □

Il problema della fermata

Consideriamo il seguente problema:

Halting problem

Trovare un algoritmo che, dati un algoritmo e l'input su cui tale algoritmo lavora, determini se l'esecuzione termina in un numero finito di passi.

Il problema della fermata

Consideriamo il seguente problema:

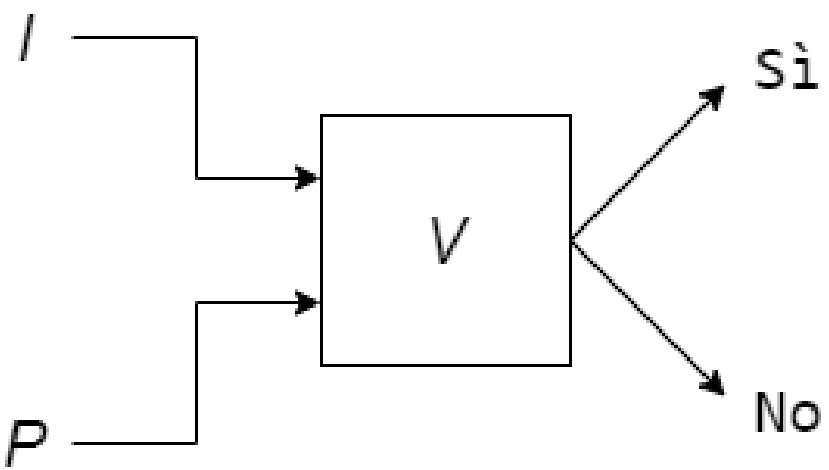
Halting problem

Trovare un algoritmo che, dati un algoritmo e l'input su cui tale algoritmo lavora, determini se l'esecuzione termina in un numero finito di passi.

In effetti, questo risulta essere impossibile, dato che il problema in questione risulta essere *indecidibile*.

Dimostrazione

Procediamo per assurdo: supponiamo che esista un algoritmo verificatore V che, dati in input un algoritmo P e l'input su cui agisce I, stampi `sì` e termini se P termina con l'input I, altrimenti stampi `no` e termini.



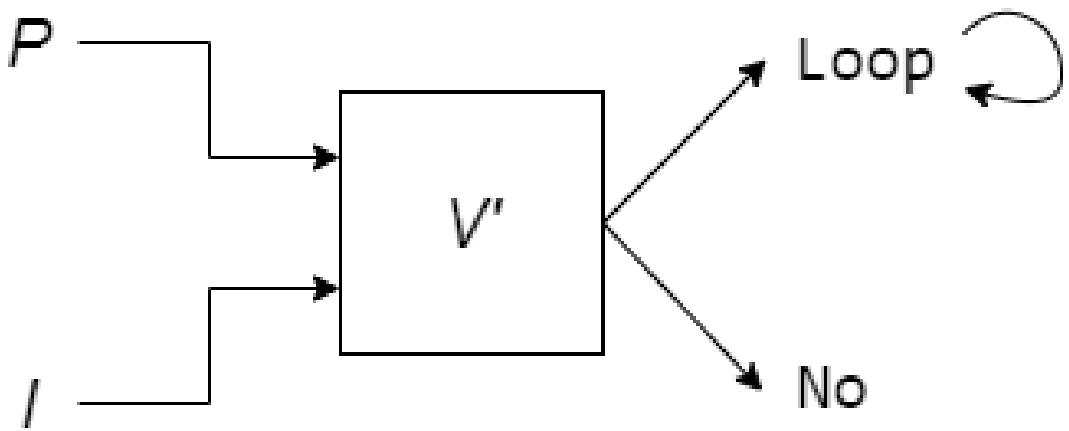
Dimostrazione

Ora, apportiamo delle modifiche successive a V, che possono essere fatte su ogni algoritmo:

Dimostrazione

Ora, apportiamo delle modifiche successive a V, che possono essere fatte su ogni algoritmo:

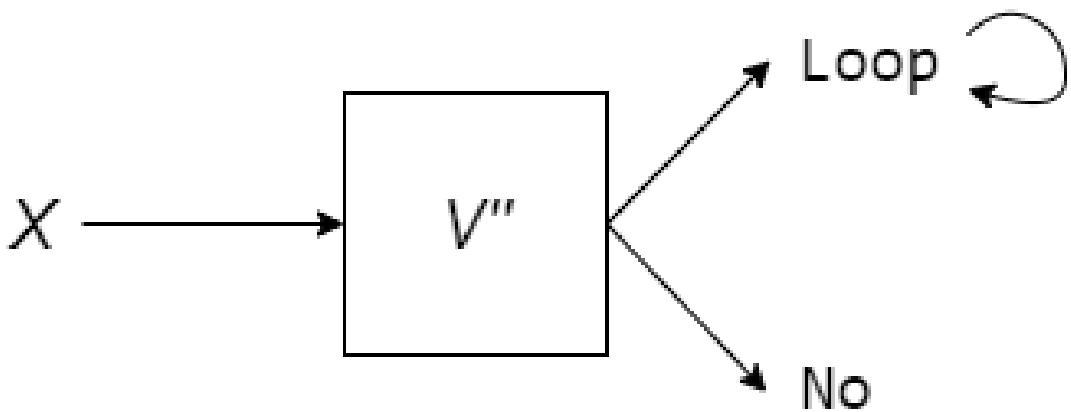
- Modifichiamo V in V' in modo che, se P non termina, invece di stampare `sì` entri in un loop infinito.



Dimostrazione

Ora, apportiamo delle modifiche successive a V, che possono essere fatte su ogni programma:

- Modifichiamo V in V' in modo che, se P non termina, invece di stampare `sì` entri in un loop infinito.
- Modifichiamo V' in V'' in modo che accetti un solo input X che funge sia da programma che da input del programma.



Dimostrazione

Ma cosa succede se poniamo $X = V''$ come input a V''?
Ci sono due possibilità:

Dimostrazione

Ma cosa succede se poniamo $X = V''$ come input a V''?
Ci sono due possibilità:

1. Se $X = V''$ termina l'esecuzione su sé stesso, allora V'' entra in un loop infinito, quindi V'' **non** termina se l'input è $X = V''$, assurdo.

Dimostrazione

Ma cosa succede se poniamo $X = V''$ come input a V''?
Ci sono due possibilità:

1. Se $X = V''$ termina l'esecuzione su sé stesso, allora V'' entra in un loop infinito, quindi V'' **non** termina se l'input è $X = V''$, assurdo.
2. Se $X = V''$ **non** termina l'esecuzione su sé stesso, allora V'' stampa `no` e termina, quindi V'' termina se l'input è $X = V''$, assurdo.

Dimostrazione

Ma cosa succede se poniamo $X = V''$ come input a V''?
Ci sono due possibilità:

1. Se $X = V''$ termina l'esecuzione su sé stesso, allora V'' entra in un loop infinito, quindi V'' **non** termina se l'input è $X = V''$, assurdo.
2. Se $X = V''$ **non** termina l'esecuzione su sé stesso, allora V'' stampa `no` e termina, quindi V'' termina se l'input è $X = V''$, assurdo.

In ogni caso raggiungiamo una contraddizione, quindi V'' non può esistere e, di conseguenza, neanche V. □

La riduzione

Abbiamo appena dimostrato che il problema della fermata è indecidibile, ma come si fa a dimostrare che altri problemi sono indecidibili?

Abbiamo due opzioni:

La riduzione

Abbiamo appena dimostrato che il problema della fermata è indecidibile, ma come si fa a dimostrare che altri problemi sono indecidibili?

Abbiamo due opzioni:

- Come abbiamo fatto con il problema della fermata, procedere per assurdo.

La riduzione

Abbiamo appena dimostrato che il problema della fermata è indecidibile, ma come si fa a dimostrare che altri problemi sono indecidibili?

Abbiamo due opzioni:

- Come abbiamo fatto con il problema della fermata, procedere per assurdo.
- Possiamo mostrare che se il nuovo problema fosse risolvibile, allora lo sarebbe anche il problema della fermata, producendo una contraddizione.

La riduzione

Abbiamo appena dimostrato che il problema della fermata è indecidibile, ma come si fa a dimostrare che altri problemi sono indecidibili?

Abbiamo due opzioni:

- Come abbiamo fatto con il problema della fermata, procedere per assurdo.
- Possiamo mostrare che se il nuovo problema fosse risolvibile, allora lo sarebbe anche il problema della fermata, producendo una contraddizione.

Questa seconda tecnica, molto efficace, è detta *riduzione* di un problema ad un altro.

La riduzione

In generale, dati un problema P, di cui vogliamo dimostrare l'indecidibilità, e un problema I, che sappiamo essere indecidibile, la tecnica della riduzione consiste di questi passaggi:

La riduzione

In generale, dati un problema P, di cui vogliamo dimostrare l'indecidibilità, e un problema I, che sappiamo essere indecidibile, la tecnica della riduzione consiste di questi passaggi:

1. Supporre per assurdo che esista un algoritmo V che verifichi il problema P, ovvero tale che, dato un input i_P, stabilisca se $i_P \in P$ [1]

[1] Ricordiamo che i problemi sono in realtà linguaggi

La riduzione

In generale, dati un problema P, di cui vogliamo dimostrare l'indecidibilità, e un problema I, che sappiamo essere indecidibile, la tecnica della riduzione consiste di questi passaggi:

1. Supporre per assurdo che esista un algoritmo V che verifichi il problema P, ovvero tale che, dato un input i_P, stabilisca se $i_P \in P$ [1]
2. Convertire gli input i_I del problema I in input i_P del problema P in modo che

$$i_I \in I \iff i_P \in P$$

[1] Ricordiamo che i problemi sono in realtà linguaggi

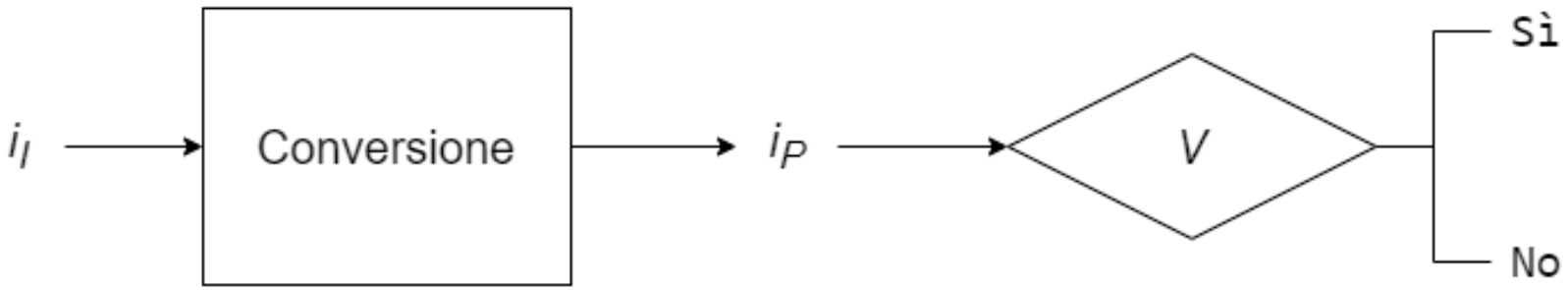
La riduzione

In generale, dati un problema P, di cui vogliamo dimostrare l'indecidibilità, e un problema I, che sappiamo essere indecidibile, la tecnica della riduzione consiste di questi passaggi:

1. Supporre per assurdo che esista un algoritmo V che verifichi il problema P, ovvero tale che, dato un input i_P, stabilisca se $i_P \in P$ [1]
2. Convertire gli input i_I del problema I in input i_P del problema P in modo che
$$i_I \in I \iff i_P \in P$$
3. Dato che V è in grado di decidere il problema P e il problema I è stato *ridotto* al problema P, concludiamo che P è decidibile e otteniamo la contraddizione desiderata.

[1] Ricordiamo che i problemi sono in realtà linguaggi

La riduzione



Schema della dimostrazione di indecidibilità per riduzione.

Esempio

Mostriamo che il seguente problema P è indecidibile:

Problema

Determinare se un determinato programma di input, scritto in linguaggio C, stampa come primi caratteri la stringa `Hello, world!`

Esempio

Mostriamo che il seguente problema P è indecidibile:

Problema

Determinare se un determinato programma di input, scritto in linguaggio C, stampa come primi caratteri la stringa `Hello, world!`

Per quanto abbiamo appena visto, è sufficiente trovare un modo di trasformare tutti gli input i_I di un problema indecidibile I in input i_P di P in modo che

$$i_I \in I \iff i_P \in P$$

Esempio

Usiamo come I il problema della fermata: dato un programma qualunque i_I, input di I, eseguiamo queste modifiche:

Esempio

Usiamo come I il problema della fermata: dato un programma qualunque i_I, input di I, eseguiamo queste modifiche:

1. Rimuoviamo tutte le istruzioni di output dal programma.

Esempio

Usiamo come I il problema della fermata: dato un programma qualunque i_I, input di I, eseguiamo queste modifiche:

1. Rimuoviamo tutte le istruzioni di output dal programma.
2. Aggiungiamo alla fine l'istruzione `printf("Hello, world!");`
   Chiamiamo il programma risultante i_P

Esempio

Usiamo come I il problema della fermata: dato un programma qualunque i_I, input di I, eseguiamo queste modifiche:

1. Rimuoviamo tutte le istruzioni di output dal programma.
2. Aggiungiamo alla fine l'istruzione `printf("Hello, world!");`
   Chiamiamo il programma risultante i_P

Dato che le istruzioni di output non influiscono sull'elaborazione, i_P termina se e solo se i_I termina e, dato che l'istruzione che abbiamo inserito alla fine è l'unica istruzione di output nel programma, i_P stampa `Hello, world!` se e solo se termina.

Esempio

Usiamo come I il problema della fermata: dato un programma qualunque i_I, input di I, eseguiamo queste modifiche:

1. Rimuoviamo tutte le istruzioni di output dal programma.
2. Aggiungiamo alla fine l'istruzione `printf("Hello, world!");`
   Chiamiamo il programma risultante i_P

Dato che le istruzioni di output non influiscono sull'elaborazione, i_P termina se e solo se i_I termina e, dato che l'istruzione che abbiamo inserito alla fine è l'unica istruzione di output nel programma, i_P stampa `Hello, world!` se e solo se termina.
Quindi, ricapitolando, i_P stampa `Hello, world!` se e solo se i_I termina, e da ciò concludiamo che P è indecidibile. □

Indice

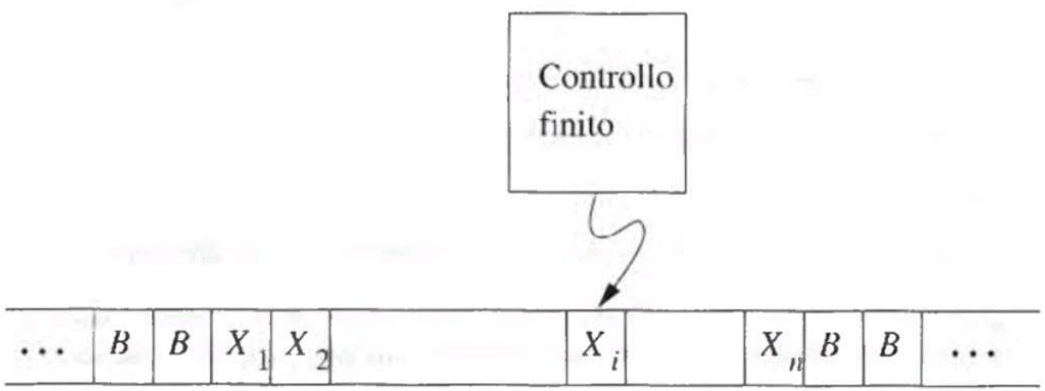
Motivazione

- Ad inizio '900 il matematico Hilbert si chiese se esistessero formule di calcolo della veridicità delle proposizioni del primo ordine sugli interi.
- Nel 1931 il matematico Gödel dimostrò che esistono proposizioni di questo tipo indecidibili, fornendo un esempio analogo a quello mostrato precedentemente.
- Si notò che tutti gli strumenti di calcolo di questi predicati risolvono la stessa classe di problemi, che si pensa essere quella delle funzioni ricorsive parziali. Questa supposizione è detta Tesi di Church-Turing.
- Turing nello stesso periodo aveva teorizzato la macchina che prende il suo nome, che è diventata di particolare importanza perché è possibile descrivere le varie configurazioni in modo molto semplice.

Definizione e notazione (informale)

Possiamo visualizzare una TM (Turing Machine) come una testina che riceve in input una stringa con finiti simboli non nulli (detta nastro). In particolare la testina si colloca inizialmente sul simbolo non nullo più a sinistra, successivamente segue delle istruzioni dipendenti esclusivamente dal proprio stato e dal simbolo che legge, per cui ad ogni mossa:

- La testina cambia di stato.
- La cella su cui è cambia di simbolo.
- La testina si sposta su una delle due celle adiacenti.



Definizione e notazione (formale)

Formalmente le mosse della testina sono determinate da una settupla del tipo $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ in cui:

- Q, q_0, F: sono rispettivamente l'insieme degli stati, lo stato iniziale e l'insieme degli stati accettanti.
- Σ, Γ, B: sono rispettivamente l'alfabeto di input, l'alfabeto di cui può fare uso M e il simbolo Blank associato alle celle vuote.
- δ: funzione di transizione tale che $\delta(q, X) = (p, Y, D)$ con $q, p \in Q$, $X, Y \in \Gamma$ e $D \in R, L$ con R=right e L=left. Indica che se nello stato q si legge X lo stato diventa p, il simbolo nella cella diventa Y e ci si sposta nella cella adiacente indicata.

Descrizioni istantanee

Per descrivere in modo conciso una configurazione si utilizza una formula del tipo $X_1 \dots X_{i-1} q X_i \dots X_n$, detta ID (Instantaneous Description), in cui:

- q precede la cella in cui si trova la testina.
- Gli X_i costituiscono i simboli tra quello non nullo più a sinistra e a destra ordinati, se la testina sta su una cella in questo intervallo.
- In alternativa la stringa viene prolungata nel verso in cui si trova la testina con i B mancanti; così q precederà X_1 o X_n a seconda dei casi.

È chiaro che in questo modo stiamo considerando tutte e sole le informazioni di cui necessitiamo.
Per indicare che la configurazione A con una mossa diventa B si usa $A \vdash B$, mentre per dire che prima o poi diventerà B si usa $A \vdash^* B$.

Il linguaggio $\{0^n1^n | n \geq 1\}$

Tale linguaggio è accettato dalla macchina

$$M_1 = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

con funzione di transizione

	Simbolo				
Stato	0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

M_1 "accoppia" gli 0 e 1 del nastro in ordine di comparizione scorrendo avanti e indietro e quando avviene qualcosa di non previsto la funzione non è definita.

Cosa accade a 0011 e 0010 in M_1?

Visualizziamo i cambiamenti delle due stringhe con i rispettivi ID:

- $q_0 0011 \vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y 1 \vdash q_2 X 0 Y 1 \vdash X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash X X q_2 Y Y \vdash X q_2 X Y Y \vdash X X q_0 Y Y \vdash X X Y q_3 Y \vdash X X Y Y q_3 B \vdash X X Y Y B q_4 B$
- $q_0 0010 \vdash X q_1 010 \vdash X 0 q_1 10 \vdash X q_2 0 Y 0 \vdash q_2 X 0 Y 0 \vdash X q_0 0 Y 0 \vdash X X q_1 Y 0 \vdash X X Y q_1 0 \vdash X X Y 0 q_1 B$

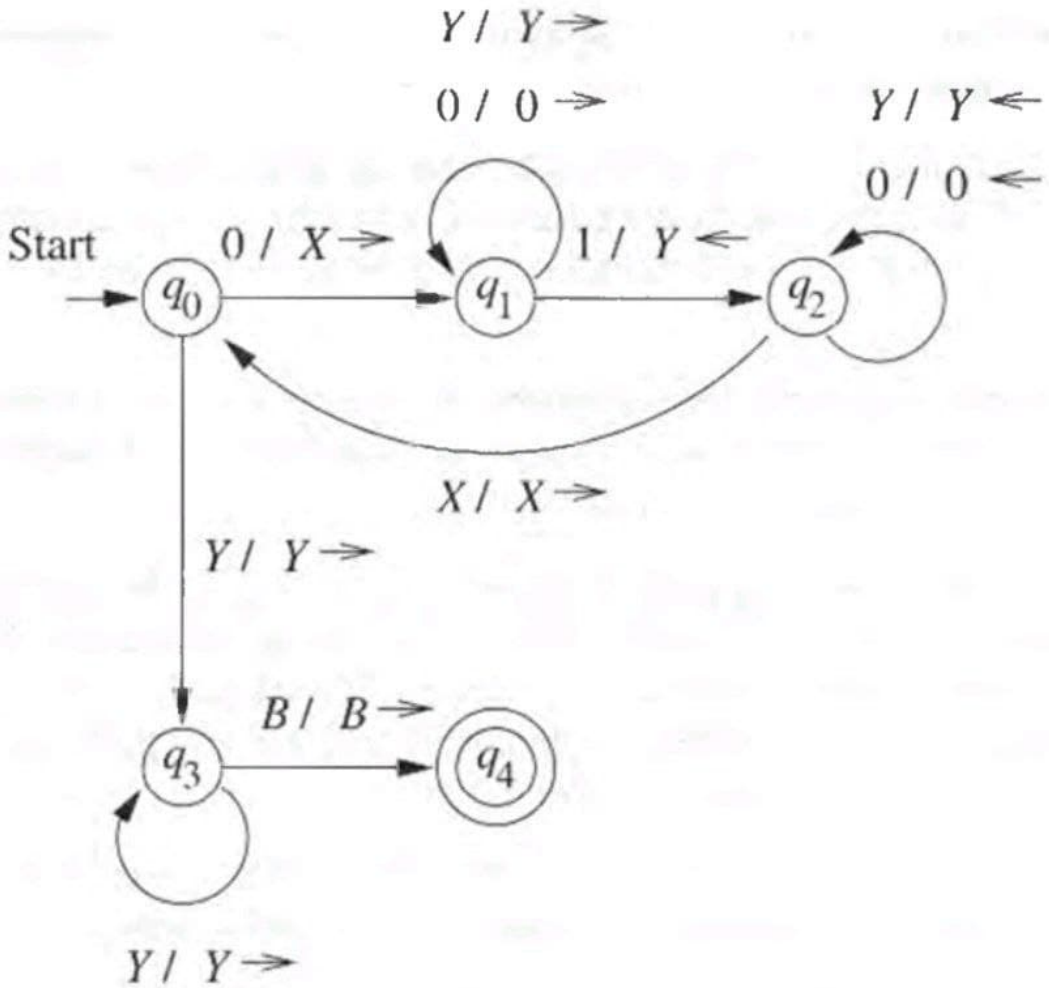
Le prime 8 configurazioni degli stati sono identiche a meno della cifra finale, la differenza è che q_1 scorre a destra finché non vede un 1, ma nel secondo caso non lo vede e finisce sul primo B, $\delta(q_1, B)$ è indefinita e quindi si arresta.

Diagrammi di transizione

Un diagramma di transizione si costruisce analogamente a quelli per gli automi a stati finiti:

- gli stati sono realizzati nello stesso modo, compresi quelli iniziali e finali
- al posto dei simboli dell'alfabeto ad un arco da q a p sono assegnate tutte le formule del tipo $X/Y\ D$ relative a $\delta(q, X) = (p, Y, D)$, perché oltre al simbolo letto vogliamo sapere anche cosa accade al nastro e alla testina.

Diagramma per M_1



La funzione monus $\dot{-}$

Consideriamo la funzione $m \dot{-} n = \max\{m - n, 0\}$ che associa alla coppia di interi positivi (m, n) la loro differenza se $m \geq n$ e 0 altrimenti.
Assumiamo di indicare (m, n) con la stringa $0^m 1 0^n$, allora $M_2 = (\{q_0, \ldots, q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B)$ con δ

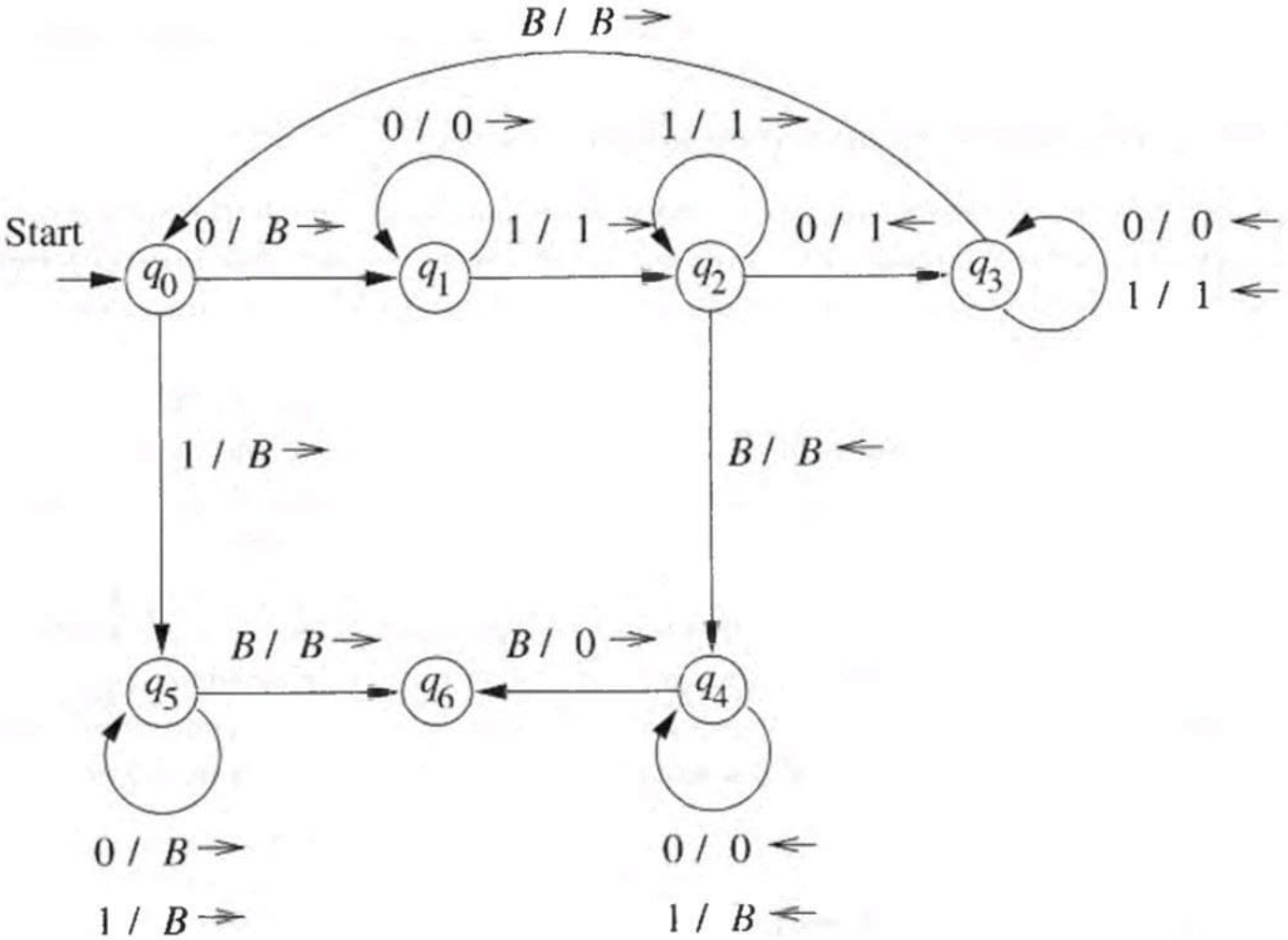
	Simbolo		
Stato	0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	–
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	–
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
q_6	–	–	–

Definitivamente si ferma con $m - n$ simboli 0 consecutivi se $m \geq n$ e con il nastro vuoto se $m > n$.

Come funziona M_2?

- q_0: È lo stato iniziale, se vede 0 lo sostituisce con B e scorre a destra andando in q_1, se vede 1 allora $m < n$ e va in q_5.
- q_1: Scorre verso destra e quando vede un 1 passa in q_2.
- q_2: Scorre verso destra cercando uno 0, se lo trova lo cambia in 1 e torna indietro passando a q_3, altrimenti trova un B, allora $m \geq n$ e passa in q_4.
- q_3: Scorre verso sinistra finché non trova il primo B, in tal caso va a destra tornando in q_0.
- q_4: Uno 0 di 0^m resta spaiato, così scorrendo verso sinistra vengono rimossi gli 1 e al primo B si mette uno 0, finendo poi in q_6.
- q_5: Siccome $m < n$ scorre verso destra cancellando tutto.
- q_6: Serve solo ad arrestare il processo.

Diagramma per M_2



Convenzioni notazionali

Specificare in ogni momento a cosa si riferisce una determinata variabile può prendere molto tempo, quindi in assenza di ambiguità è possibile adottare le seguenti convenzioni.

- Si usano le prime lettere minuscole (dell'alfabeto) per i simboli di input.
- Si usano le lettere maiuscole vicine alla fine per generici simboli di nastro.
- Si usano ultime lettere minuscole per le stringhe di simboli di input.
- Si usano lettere greche per le stringhe di simboli di nastro.
- Si usano le lettere vicine a p e q per gli stati.

Il linguaggio di una TM

Il linguaggio $L(M)$ accettato da M è l'insieme delle $w \in \Sigma^*$ che definitivamente entrano in uno stato accettante, formalmente

$$L(M) = \{w \in \Sigma^* | \exists \alpha, \beta, p \in F : q_0 w \vdash^* \alpha p \beta\}$$

$L(M)$ al variare di M identifica i cosiddetti linguaggi ricorsivamente enumerabili (RE); questi sono ancora più generali dei CFL, ma esistono anche linguaggi non RE.

Il ruolo dell'arresto

Una TM si arresta se nello stato q vede il simbolo X e $\delta(q, X)$ è indefinito. In generale è sempre possibile modificare una TM affinché si arresti quando accetta, basta infatti rendere indefinita $\delta(f, -)$ per ogni $f \in F$. Non è invece sempre possibile cambiare una TM in modo che definitivamente termini in ogni caso, ciò porta alle seguenti considerazioni:

- Se L è accettato da un M che definitivamente si arresta per ogni stringa si dice che è ricorsivo (e M rispecchia la nostra definizione intuitiva di "algoritmo").
- Questa distinzione è particolarmente rilevante poiché se interpretiamo L come un problema, è decidibile se e solo se è ricorsivo.

Indice

Tecniche di programmazione per Macchina di Turing

Per mostrare che la Macchina di Turing può essere usata in modo simile ad un calcolatore tradizionale, presentiamo ora degli espedienti che rendono più intuitive le operazioni della macchina. Questi espedienti non estendono il modello fondamentale della macchina, poiché i linguaggi riconoscibili sono gli stessi, ma ne rendono più chiara la notazione. Vedremo le seguenti tre tecniche:

1. Memoria nello stato
2. Tracce multiple
3. Subroutine

Memoria nello stato

- Ad ogni passo della macchina, il controllo può essere utilizzato non solo per controllare in che stato ci si trovi, ma anche a rappresentare e controllare un numero finito di variabili.

Memoria nello stato

- Ad ogni passo della macchina, il controllo può essere utilizzato non solo per controllare in che stato ci si trovi, ma anche a rappresentare e controllare un numero finito di variabili.
- A tal scopo, per esempio, si può decidere di controllare lo stato q e tre variabili A, B, C. Per farlo basterà considerare lo stato come una quadrupla (q, A, B, C). In tal modo la strategia del programma è più chiara.

TM per $01^* + 10^*$

- Definiamo una TM per il linguaggio linguaggio $01^* + 10^*$:

$$M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, [q_0, B], \{[q_1, B]\})$$

TM per $01^* + 10^*$

- Definiamo una TM per il linguaggio linguaggio $01^* + 10^*$:

$$M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0, B], \{[q_1, B]\})$$

- L'insieme degli stati Q è $\{q_0, q_1\}x\{0,1,B\}$, ogni stato è composto da una parte di controllo e un dato che ricorda il primo simbolo letto

TM per $01^* + 10^*$

- Definiamo una TM per il linguaggio linguaggio $01^* + 10^*$:

$$M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0, B], \{[q_1, B]\})$$

- L'insieme degli stati Q è $\{q_0, q_1\}x\{0,1,B\}$, ogni stato è composto da una parte di controllo e un dato che ricorda il primo simbolo letto
- La funzione di transizione δ è:
  1. $\delta([q_0, B], a) = ([q_1, a], a, R)$ per $a = 0$ o $a = 1$
  2. $\delta([q_1, a], \bar{a}) = ([q_1, a], \bar{a}, R)$
  3. $\delta([q_1, a], B) = ([q_1, B], B, R)$

Tracce multiple

- A volte è utile considerare il nastro della macchina come se avesse più tracce, cioè più di un simbolo scritto sopra.

Tracce multiple

- A volte è utile considerare il nastro della macchina come se avesse più tracce, cioè più di un simbolo scritto sopra.
- A tal scopo, come nell'esempio della memoria nello stato, è sufficiente esprimere i simboli di nastro come delle ennuple, ognuna contenente un certo tipo di informazione.

Tracce multiple

- A volte è utile considerare il nastro della macchina come se avesse più tracce, cioè più di un simbolo scritto sopra.
- A tal scopo, come nell'esempio della memoria nello stato, è sufficiente esprimere i simboli di nastro come delle ennuple, ognuna contenente un certo tipo di informazione.
- Anche l'impiego di tracce multiple non estende le capacità delle macchine, ma rende soltanto più comoda la notazione.

TM per linguaggio non libero

- Consideriamo il linguaggio non libero

$$L = \{wcw | w \text{ è in } (0+1)^+\}$$

TM per linguaggio non libero

- Consideriamo il linguaggio non libero

$$L = \{wcw | w \text{ è in } (0+1)^+\}$$

- La TM corrispondente è

$$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_9, B]\})$$

TM per linguaggio non libero

- Consideriamo il linguaggio non libero

$$L = \{wcw | w \text{ è in } (0+1)^+\}$$

- La TM corrispondente è

$$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_9, B]\})$$

- $Q = \{q_1, q_2, \ldots, q_9\} x \{0, 1, B\}$

TM per linguaggio non libero

- Consideriamo il linguaggio non libero

$$L = \{wcw | w \text{ è in } (0+1)^+\}$$

- La TM corrispondente è

$$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_9, B]\})$$

- $Q = \{q_1, q_2, \ldots, q_9\} x \{0, 1, B\}$
- $\Gamma = \{B, *\} x \{0, 1, c, B\}$

TM per linguaggio non libero

- Consideriamo il linguaggio non libero

$$L = \{wcw | w \text{ è in } (0+1)^+\}$$

- La TM corrispondente è

$$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_9, B]\})$$

- $Q = \{q_1, q_2, \ldots, q_9\} x \{0, 1, B\}$
- $\Gamma = \{B, *\} x \{0, 1, c, B\}$
- I simboli di Σ sono $[B, 0]$, $[B, 1]$ e $[B, c]$, che identifichiamo con 0, 1 e c.

TM per linguaggio non libero

La funzione δ è definita con queste regole, dove a e b possono valere 0 o 1.

- $\delta([q_1, B], [B, a]) = ([q_2, a], [*, a], R)$
- $\delta([q_2, a], [B, b]) = ([q_2, a], [B, b], R)$
- $\delta([q_2, a], [B, c]) = ([q_3, a], [B, c], R)$
- $\delta([q_3, a], [*, b]) = ([q_3, a], [*, b], R)$
- $\delta([q_3, a], [B, a]) = ([q_4, B], [*, a], L)$
- $\delta([q_4, B], [*, a]) = ([q_4, B], [*, a], L)$
- $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$
- $\delta([q_5, B], [B, a]) = ([q_6, B], [B, a], L)$
- $\delta([q_6, B], [B, a]) = ([q_6, B], [B, a], L)$
- $\delta([q_6, B], [*, a]) = ([q_1, B], [*, a], R)$

TM per linguaggio non libero

- $\delta([q_5, B], [*, a]) = ([q_7, B], [*, a], R)$
- $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$
- $\delta([q_8, B], [*, a]) = ([q_8, B], [*, a], R)$
- $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], R)$

Subroutine

- Come i programmi in C possono essere suddivisi in funzioni e procedure, possiamo suddividere anche una macchina di Turing in diverse parti, o *subroutine*, che interagiscono
- Ognuna di queste parti avrà uno stato iniziale e uno stato senza mosse, che serve per passsare il controllo all'insieme che ha chiamato la subroutine, ovvero l'insieme in cui c'era una transizione allo stato iniziale della subroutine
- Se vogliamo chiamare una subroutine da stati diversi dobbiamo crearne diverse copie, per dare ad ognuna delle istruzioni di ritorno diverse

Moltiplicazione

Possiamo in questo modo suddividere una TM che realizzi la moltiplicazione fra due interi. Essa comincerà con 0^m10^n1 sul nastro e terminerà con 0^{mn}. La macchina funziona in questo modo:

Moltiplicazione

Possiamo in questo modo suddividere una TM che realizzi la moltiplicazione fra due interi. Essa comincerà con 0^m10^n1 sul nastro e terminerà con 0^{mn}. La macchina funziona in questo modo:

1. Ad ogni passaggio il nastro ospiterà una stringa non vuota della forma $0^i10^n10^{kn}$

Moltiplicazione

Possiamo in questo modo suddividere una TM che realizzi la moltiplicazione fra due interi. Essa comincerà con 0^m10^n1 sul nastro e terminerà con 0^{mn}. La macchina funziona in questo modo:

1. Ad ogni passaggio il nastro ospiterà una stringa non vuota della forma $0^i10^n10^{kn}$
2. Ad ogni passo si cambia in B uno 0 del primo gruppo e si aggiunge n volte 0 all'ultimo gruppo, ottenendo una stringa della forma $0^{i-1}10^n10^{(k+1)n}$

Moltiplicazione

Possiamo in questo modo suddividere una TM che realizzi la moltiplicazione fra due interi. Essa comincerà con 0^m10^n1 sul nastro e terminerà con 0^{mn}. La macchina funziona in questo modo:

1. Ad ogni passaggio il nastro ospiterà una stringa non vuota della forma $0^i10^n10^{kn}$
2. Ad ogni passo si cambia in B uno 0 del primo gruppo e si aggiunge n volte 0 all'ultimo gruppo, ottenendo una stringa della forma $0^{i-1}10^n10^{(k+1)n}$
3. Quando tutti gli 0 del primo gruppo saranno stati cancellati, basterà cancellare la sequenza iniziale 10^n1

Moltiplicazione

Possiamo in questo modo suddividere una TM che realizzi la moltiplicazione fra due interi. Essa comincerà con 0^m10^n1 sul nastro e terminerà con 0^{mn}. La macchina funziona in questo modo:

1. Ad ogni passaggio il nastro ospiterà una stringa non vuota della forma $0^i10^n10^{kn}$
2. Ad ogni passo si cambia in B uno 0 del primo gruppo e si aggiunge n volte 0 all'ultimo gruppo, ottenendo una stringa della forma $0^{i-1}10^n10^{(k+1)n}$
3. Quando tutti gli 0 del primo gruppo saranno stati cancellati, basterà cancellare la sequenza iniziale 10^n1

La macchina, per eseguire il passo (2) e copiare in coda il blocco di n simboli 0, si avvarrà di una subroutine che trasformerà una ID di forma $0^{m-k}1q_10^n10^{(k-1)n}$ in $0^{m-k}1q_50^n10^{kn}$

Estensioni alla macchina di Turing

Vediamo ora alcuni modelli analoghi alle macchine di Turing, e che riconoscono ancora gli stessi linguaggi della macchina base. Questi sono:

1. Macchine di Turing multinastro
2. Macchine di Turing non deterministiche

Macchine di Turing multinastro

Una TM multinastro ha un numero finito di nastri e di controlli. Come nella macchina normale ogni nastro è diviso in celle, che possono contenere un simbolo di nastro di cui i simboli di input sono un sottoinsieme. L'insieme degli stati contiene sempre uno stato iniziale e un insieme di stati accettanti. Le differenze sono:

- L'input, con le stesse caratteristiche della TM base, si trova solo sul primo nastro
- Le celle del nastro diverso dal primo contengono inizialmente un blank
- Le testine dei nastri diversi dal primo si trovano su celle arbitrarie, che sono equivalenti
- Ad ogni mossa ogni testina può spostarsi a destra, a sinistra o stare ferma indipendentemente dalle altre. Useremo S come simbolo per una testina che sta ferma

Equivalenza con Macchina base

Dimostriamo che un linguaggio accettato da una Macchina multinastro è ricorsivamente numerabile. Per farlo, costruiamo una macchina mononastro N che simuli il comportamento di una macchina multinastro M.

Equivalenza con Macchina base

Dimostriamo che un linguaggio accettato da una Macchina multinastro è ricorsivamente numerabile. Per farlo, costruiamo una macchina mononastro N che simuli il comportamento di una macchina multinastro M.

- Dividiamo il nastro di N in $2k$ tracce, dove metà delle tracce replicano i nastri di M e le rimanenti ospitano ognuna un marcatore, per indicare la posizione corrente della testina del corrispondente nastro di M

Equivalenza con Macchina base

Dimostriamo che un linguaggio accettato da una Macchina multinastro è ricorsivamente numerabile. Per farlo, costruiamo una macchina mononastro N che simuli il comportamento di una macchina multinastro M.

- Dividiamo il nastro di N in $2k$ tracce, dove metà delle tracce replicano i nastri di M e le rimanenti ospitano ognuna un marcatore, per indicare la posizione corrente della testina del corrispondente nastro di M
- Ad ogni mossa, la testina di N deve visitare tutti i K marcatori, per memorizzare in una componente del controllo i simboli nelle celle corrispondenti

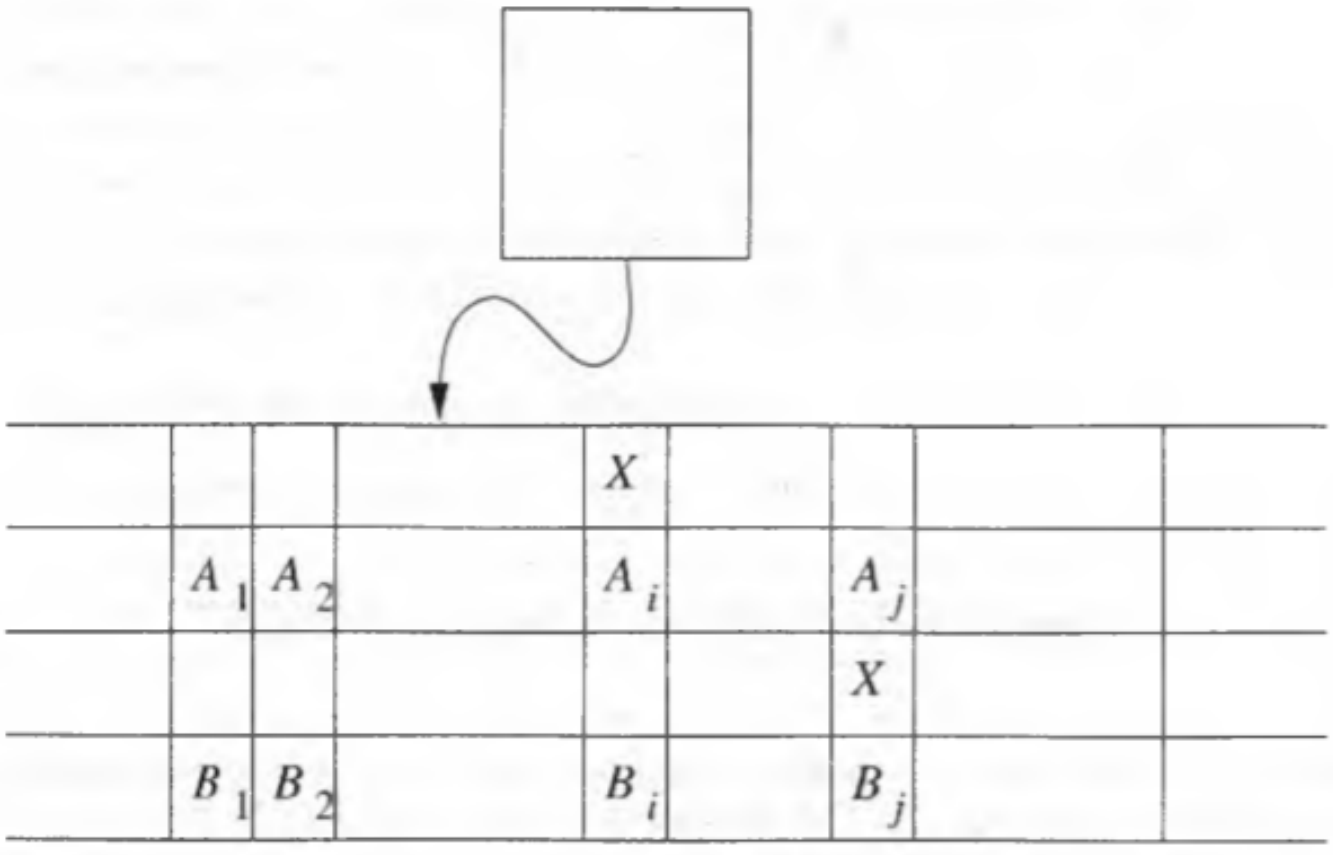
Equivalenza con Macchina base

- Dopo aver visitato ogni marcatore N conosce tutti i simboli guardati da M e il suo stato, e può determinare la prossima mossa. Per eseguirla deve modificare il proprio stato e la posizione di tutti i marcatori, rivisitandoli e spostandoli a destra o a sinistra se necessario

Equivalenza con Macchina base

- Dopo aver visitato ogni marcatore N conosce tutti i simboli guardati da M e il suo stato, e può determinare la prossima mossa. Per eseguirla deve modificare il proprio stato e la posizione di tutti i marcatori, rivisitandoli e spostandoli a destra o a sinistra se necessario
- Gli stati accettanti di N saranno tutti e soli gli stati accettanti di M. In tal modo, poiché N simula le mosse di M, riconoscerà gli stessi linguaggi

Equivalenza con Macchina base



Macchine di Turing non deterministiche

- Una macchina di Turing è una versione modificata di quella base, in cui però ad ogni stato q e simbolo di nastro X viene associato un insieme finito di triple:

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$

Macchine di Turing non deterministiche

- Una macchina di Turing è una versione modificata di quella base, in cui però ad ogni stato q e simbolo di nastro X viene associato un insieme finito di triple:

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$

- Ad ogni passo la NTM sceglie come mossa una delle triple, dovendo però scegliere lo stato, il simbolo di nastro e la direzione dalla stessa tripla

Macchine di Turing non deterministiche

- Una macchina di Turing è una versione modificata di quella base, in cui però ad ogni stato q e simbolo di nastro X viene associato un insieme finito di triple:

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$

- Ad ogni passo la NTM sceglie come mossa una delle triple, dovendo però scegliere lo stato, il simbolo di nastro e la direzione dalla stessa tripla
- Analogamente ad altri modelli non deterministici, un input è accettato se c'è una sequenza di scelte che conduce dall'ID iniziale con quell'input ad una ID con stato accettante

Equivalenza NTM e Macchina base

Dimostriamo che anche ogni linguaggio accettato da una Macchina non deterministica è ricorsivamente numerabile. Per farlo, data una M_N non deterministica, costruiamo una macchina deterministica M_D con due nastri.

Equivalenza NTM e Macchina base

Dimostriamo che anche ogni linguaggio accettato da una Macchina non deterministica è ricorsivamente numerabile. Per farlo, data una M_N non deterministica, costruiamo una macchina deterministica M_D con due nastri.

- Il primo nastro di M_D contiene la sequenza di ID, separate dal simbolo *, che rappresentano tutte le ID possibili di M_N. Una ID è contrassegnata come corrente

Equivalenza NTM e Macchina base

Dimostriamo che anche ogni linguaggio accettato da una Macchina non deterministica è ricorsivamente numerabile. Per farlo, data una M_N non deterministica, costruiamo una macchina deterministica M_D con due nastri.

- Il primo nastro di M_D contiene la sequenza di ID, separate dal simbolo *, che rappresentano tutte le ID possibili di M_N. Una ID è contrassegnata come corrente
- Per elaborare l'ID corrente, M_D compie quattro operazioni:

Equivalenza NTM e Macchina base

Dimostriamo che anche ogni linguaggio accettato da una Macchina non deterministica è ricorsivamente numerabile. Per farlo, data una M_N non deterministica, costruiamo una macchina deterministica M_D con due nastri.

- Il primo nastro di M_D contiene la sequenza di ID, separate dal simbolo *, che rappresentano tutte le ID possibili di M_N. Una ID è contrassegnata come corrente
- Per elaborare l'ID corrente, M_D compie quattro operazioni:
  - Esamina lo stato e il simbolo della ID corrente, avendo nel controllo tutte le scelte di M_N. Se lo stato è accettante, M_D termina e accetta

Equivalenza NTM e Macchina base

Dimostriamo che anche ogni linguaggio accettato da una Macchina non deterministica è ricorsivamente numerabile. Per farlo, data una M_N non deterministica, costruiamo una macchina deterministica M_D con due nastri.

- Il primo nastro di M_D contiene la sequenza di ID, separate dal simbolo *, che rappresentano tutte le ID possibili di M_N. Una ID è contrassegnata come corrente
- Per elaborare l'ID corrente, M_D compie quattro operazioni:
  - Esamina lo stato e il simbolo della ID corrente, avendo nel controllo tutte le scelte di M_N. Se lo stato è accettante, M_D termina e accetta
  - Se non è accettante e porta a k mosse, M_D copia la ID sul secondo nastro e ne copia k copie in fondo al primo nastro

Equivalenza NTM e Macchina base

- Modifica ognuna delle k ID secondo le mosse possibili

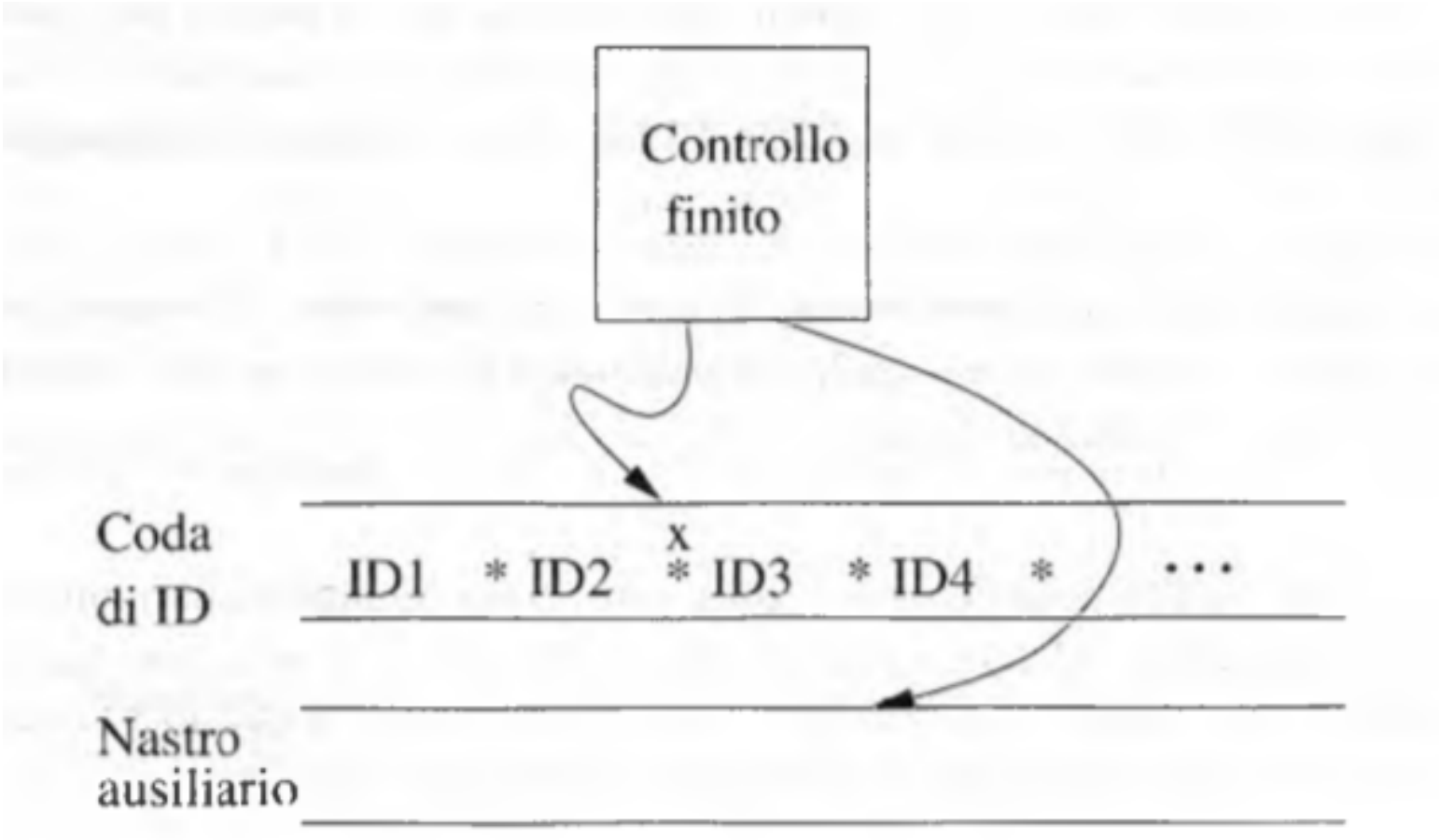
Equivalenza NTM e Macchina base

-
  - Modifica ognuna delle k ID secondo le mosse possibili
  - Torna alla ID corrente, cancella il contrassegno e lo sposta sulla ID successiva

Equivalenza NTM e Macchina base

- - Modifica ognuna delle k ID secondo le mosse possibili
  - Torna alla ID corrente, cancella il contrassegno e lo sposta sulla ID successiva
- Se M_N accettava in n mosse, allora M_D accetterà in al massimo nm^n mosse

Equivalenza NTM e Macchina base



Indice

Macchine di Turing ridotte

Può essere utile, specialmente per alcuni tipi di costruzioni, imporre delle limitazioni ad una generica macchina di Turing. In questa serie di slide verificheremo l'equivalenza tra una generica TM ed una con le due seguenti limitazioni:

1. Limitiamo il nastro in una delle due direzioni: creiamo così un nastro semi-infinito;
2. Vietiamo alla TM di stampare un blank.

Macchine di Turing ridotte

Può essere utile, specialmente per alcuni tipi di costruzioni, imporre delle limitazioni ad una generica macchina di Turing. In questa serie di slide verificheremo l'equivalenza tra una generica TM ed una con le due seguenti limitazioni:

1. Limitiamo il nastro in una delle due direzioni: creiamo così un nastro semi-infinito;
2. Vietiamo alla TM di stampare un blank.

Vediamo la dimostrazione.

Euristica

X_0	X_1	X_2	$\cdots$
$*$	X_{-1}	X_{-2}	$\cdots$

Dimostrazione

- Sostituiamo il carattere blank con un generico B' non vuoto: in tal modo la "nuova" TM è identica alla precedente eccetto che per la δ che modifichiamo nel seguente modo:

$$\delta(q, X) = (p, blank, D) \Rightarrow \delta(q, X) = (p, B')$$

$$\delta(q, B') = \delta(q, blank).$$

Dimostrazione

- Sostituiamo il carattere blank con un generico B' non vuoto: in tal modo la "nuova" TM è identica alla precedente eccetto che per la δ che modifichiamo nel seguente modo:

$$\delta(q, X) = (p, blank, D) \Rightarrow \delta(q, X) = (p, B')$$

$$\delta(q, B') = \delta(q, blank).$$

-

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

$$\Downarrow M =$$

$$(\{q_0, q_1\} \cup (Q \times \{U, L\}), \Sigma \times B, \Gamma \times \Gamma \cup \Gamma \times \{*\}, \delta_N, q_0, B_{new}, F \times \{U, L\})$$

Perché funziona?

Una facile dimostrazione per induzione sui singoli passi della TM mostra che la costruzione precedente risolve gli stessi problemi della TM originaria (per semplicità in quella di partenza avevamo già implementato l'esistenza di un carattere non blank che sostituiva il blank). Motiviamo la costruzione.

Perché funziona?

Una facile dimostrazione per induzione sui singoli passi della TM mostra che la costruzione precedente risolve gli stessi problemi della TM originaria (per semplicità in quella di partenza avevamo già implementato l'esistenza di un carattere non blank che sostituiva il blank). Motiviamo la costruzione.

- Gli stati sono esattamente quelli della TM iniziale, ma con la distinzione che ogni stato si sdoppia (in base a dove si trova la testina: nella parte superiore o nella parte inferiore del nastro), in maniera tale da distinguere la posizione della testina (differenziando dunque ogni "celletta" della semiretta del nastro semi-infinito, mettendola in relazione biunivoca con la retta del nastro infinito). q_0 e q_1 sono due stati: q_0 l'iniziale e q_1 ausiliare, ne capiremo l'utilità quando osserveremo la costruzione della δ caratteristica.

Perché funziona?

- L'alfabeto è formato da coppie di simboli dell'alfabeto iniziale: in particolare ogni simbolo di input è della forma $[a, B]$ con $a \in \Sigma$ e B è il sostituto a blank. Il blank in questa nuova TM è $[B, B]$. Per ogni $a \in \Gamma$ troviamo $[a, *]$ che corrisponde al carattere a della TM originaria, ma specificando che ci troviamo all'estremo sinistro del nastro.
- $\delta_N(q_0, [a, B]) = (q_1, [a, *], R) \forall a \in \Sigma$: la prima mossa di questa TM colloca il segno * sulla traccia inferiore della cella più a sinistra. Lo stato diventa q_1 e la testina si sposta a destra perché non può andare a sinistra né restare ferma.

Perché funziona?

- L'alfabeto è formato da coppie di simboli dell'alfabeto iniziale: in particolare ogni simbolo di input è della forma $[a, B]$ con $a \in \Sigma$ e B è il sostituto a blank. Il blank in questa nuova TM è $[B, B]$. Per ogni $a \in \Gamma$ troviamo $[a, *]$ che corrisponde al carattere a della TM originaria, ma specificando che ci troviamo all'estremo sinistro del nastro.
- $\delta_N(q_0, [a, B]) = (q_1, [a, *], R) \forall a \in \Sigma$: la prima mossa di questa TM colloca il segno * sulla traccia inferiore della cella più a sinistra. Lo stato diventa q_1 e la testina si sposta a destra perché non può andare a sinistra né restare ferma. $\delta_N(q_1, [a, B]) = ([q_{old}, U], [a, B], L) \forall a \in \Sigma$: sullo stato q_1 questa nuova *TM* imita la vecchia, torna allo stato q_{old} impostando la testina sulla "traccia superiore".

Perché funziona?

- Se $\delta_{old}(q, X) = (p, Y, D) \rightarrow \delta_N([q, U], [X, Z]) = ([p, U], [Y, Z], D)$ e simmetrico. Sostanzialmente la nuova TM simula esattamente la vecchia TM se non si trova vicino al bordo. Se supera la "fine sinistra" del nastro, la testina segue la traccia inferiore ed esegue lo spostamento contrario che avrebbe eseguito la vecchia TM.

Perché funziona?

- Se $\delta_{old}(q, X) = (p, Y, D) \rightarrow \delta_N([q, U], [X, Z]) = ([p, U], [Y, Z], D)$ e simmetrico. Sostanzialmente la nuova TM simula esattamente la vecchia TM se non si trova vicino al bordo. Se supera la "fine sinistra" del nastro, la testina segue la traccia inferiore ed esegue lo spostamento contrario che avrebbe eseguito la vecchia TM.
- Se la vecchia TM punta a destra della posizione iniziale, indipendentemente dalla direzione, anche la nuova si muoverà a destra (sulla traccia U, ossia quella in alto con riferimento alla figura della slide 'Euristica'). Simile per la sinistra.

Perché funziona?

- Gli stati finali sono esattamente quelli della TM iniziale, però con la giustapposizione di una U (parte superiore) o di una L (parte inferiore), necessarie, per quanto detto prima, in quanto è importante capire da dove si proviene con una determinata mossa (infatti se la vecchia TM agiva muovendo la testina verso sinistra in uno stato, nella nuova se lo stato vien raggiunto operando sulla stringa inferiore, la testina dovrà muoversi verso destra).

PDPA a due pile e Macchina di Turing

Teorema

Un generico linguaggio $L(M)$ accettato da una TM, è riconosciuto anche da un automa bi-pila.

Bozza di dimostrazione:
A livello intuitivo, uno stack conserva le informazioni che stanno a sinistra della testina (in termini di simboli di nastro), mentre l'altro stack ciò che sta a destra.
Creiamo un simbolo speciale ç che poniamo alla fine di ogni stack: quando il PDPA lo incontra simula la lettura di un blank.
Altrimenti per ogni stringa ordinata w di simboli sul nastro di una TM, poniamo quella stringa interna allo stack di sinistra (W.L.O.G.) con l'estremità verso il simbolo ç (possiamo farlo bene dato che supponiamo il nastro della TM semi-infinito per quanto dimostrato prima). Ogni volta che una certo simbolo viene letto dalla testina, lo spostamento della testina è simulato dal *pop* di un carattere da uno stack all'altro.
Ogni stato terminale di M è riconosciuto terminale dal controllo del PDPA.

Turing Machine e Computer

Un computer può simulare una TM.
Prendiamo il controllo finito di una determinata TM: esso è formato da un numero finito di stati, pertanto un ipotetico programma di un computer potrebbe implementare lo stesso controllo creando una associazione tra un numero finito di caratteri e le rispettive regole di transizione. I simboli di nastro sono sequenze di caratteri (in numero finito).
Se supponiamo che la memoria del computer sia sufficientemente grande da poter contenere tutti i dati che ci sono utili durante un ciclo della TM che vogliamo emulare, ci basta disporre le informazioni all'interno della memoria dividendole in due dischi: una disco sinistro e un disco destro.

Abbiamo in effetti creato un PDPA con due stack e senza problemi di memoria (se finiamo i dischi del computer, basta aggiungerne altri), che per quanto detto precedentemente è in grado di riconoscere i linguaggi che riconosce una TM. Vale notare che, senza la possibilità di aggiungere dischi di memoria, il computer sarebbe in grado di riconoscere solo linguaggi regolari.

Turing e computer, verso opposto

Presentiamo l'idea della dimostrazione.

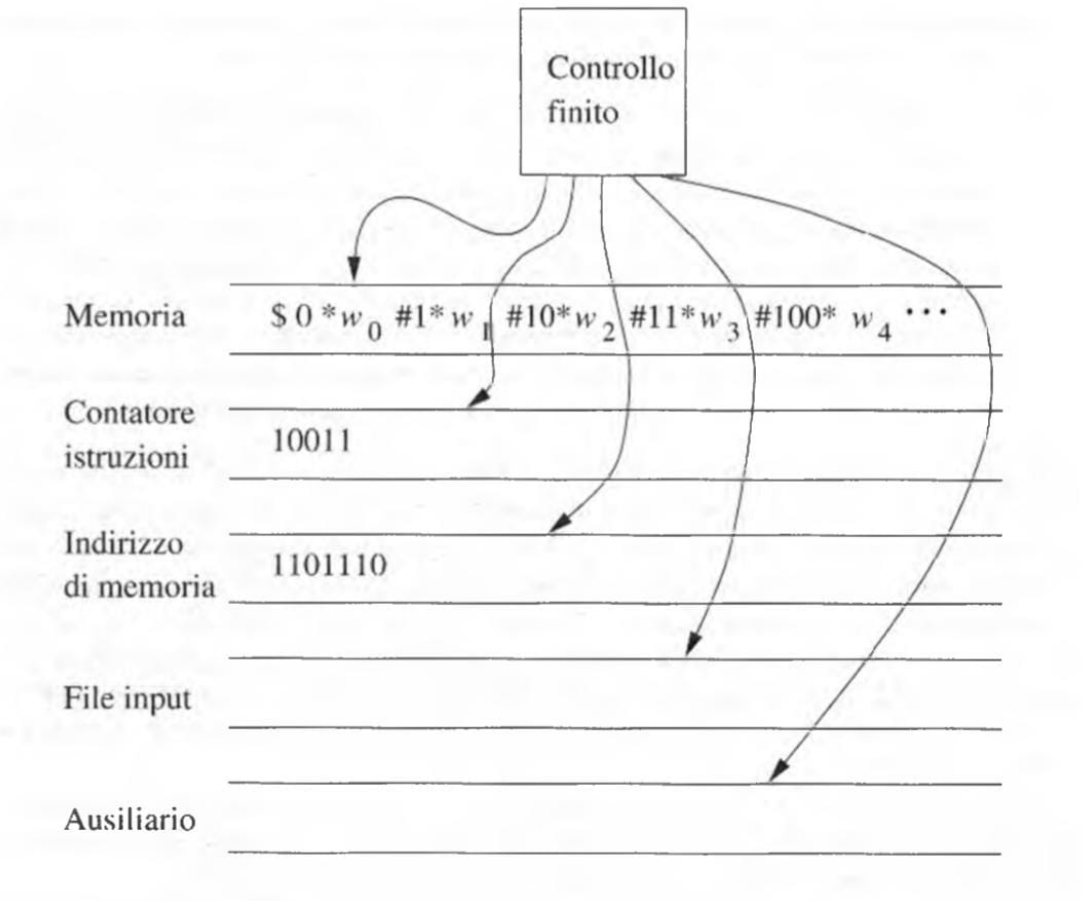


Figura: Esempio di TM Multinastro

Esempio di costruzioni

Conway's Game of Life

Regole: data una qualsiasi cella, si contino il numero di celle vive tra le 8 che la circondano. Se questo numero è inferiore a due, la cella precedentemente viva non sarà viva alla generazione successiva. Se ci sono 2 o 3 celle vive, la cella precedentemente viva sopravvive. Se ce ne sono di più muore. Se una cella vuota è circondata da esattamente 3 celle vive, alla generazione successiva vivrà.

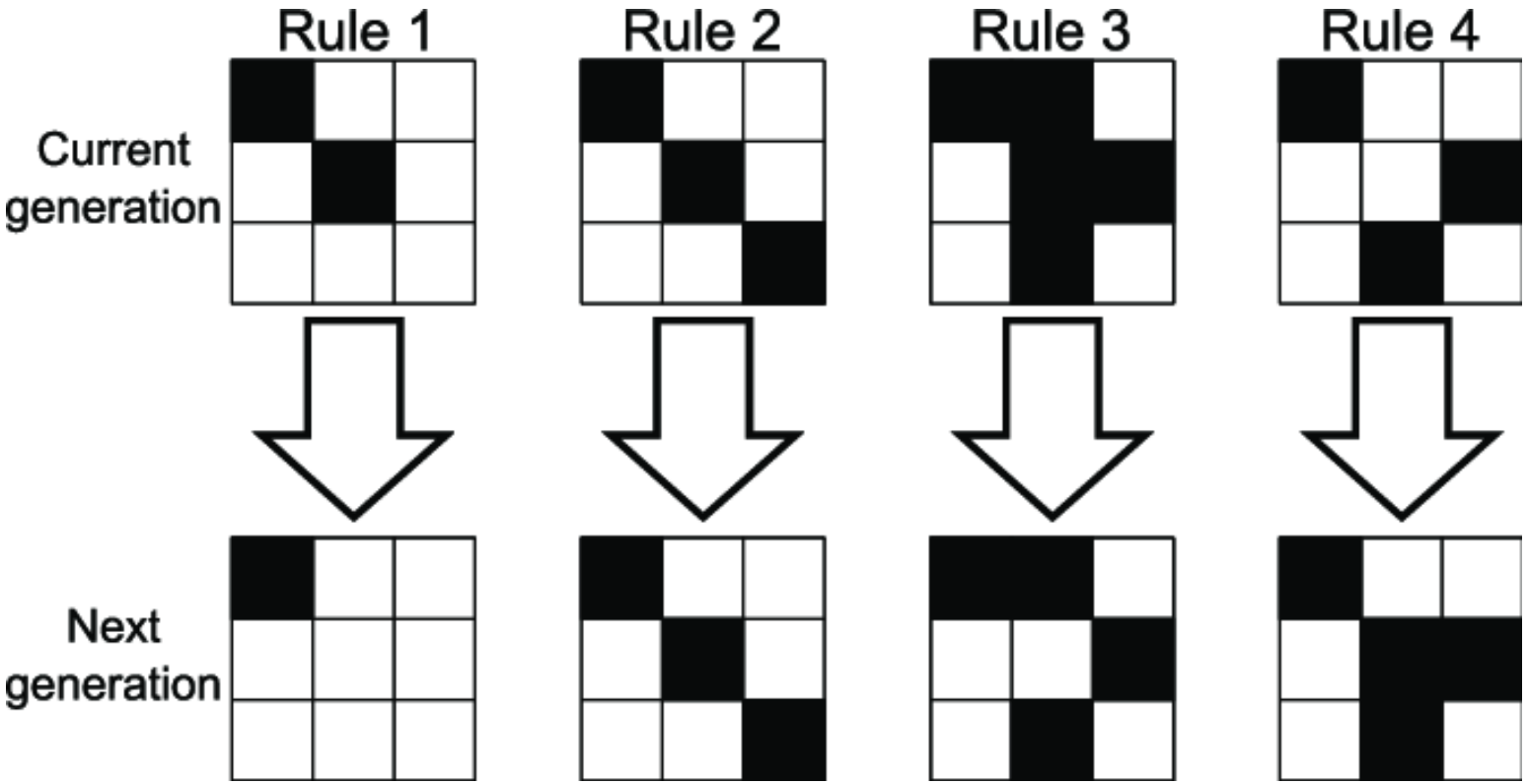


Figura: Esempio evolutivo

Conway's GOL

Teorema

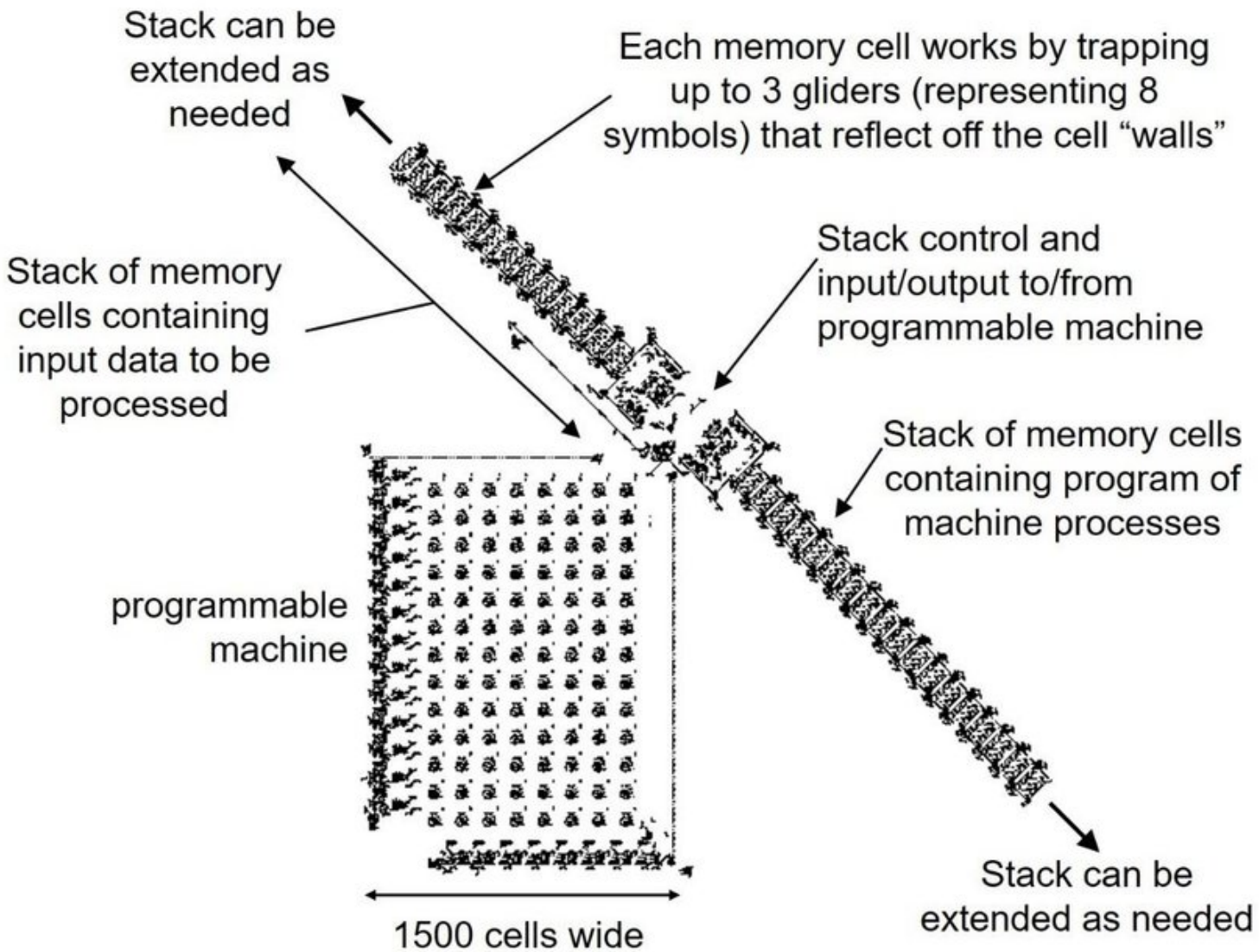
Il problema che consiste nello stabilire se una configurazione iniziale di celle in Game of Life sia destinata ad estinguersi o stabilizzarsi, è indecidibile

Conway's GOL

Teorema

Il problema che consiste nello stabilire se una configurazione iniziale di celle in Game of Life sia destinata ad estinguersi o stabilizzarsi, è indecidibile

Dimostrazione:
Tramite le costruzioni di Gosper (come il cannone ad aliante, insieme di celle che dopo un certo numero di generazioni "creano" proiettili o alianti) è possibile costruire una TM universale. Per quanto dimostrato precedente, qualsiasi entità Turing-completa ammette un insieme di problemi indecidibili (abbiamo visto come mandare in Halt la macchina): sia quella TM con i rispettivi simboli di nastro esattamente quella emulata in una partita di Game of Life $\Rightarrow$ l'esito è a priori indecidibile.


Stack can be extended as needed
Each memory cell works by trapping up to 3 gliders (representing 8 symbols) that reflect off the cell "walls"
Stack control and input/output to/from programmable machine
Stack of memory cells containing input data to be processed
Stack of memory cells containing program of machine processes
programmable machine
1500 cells wide
Stack can be extended as needed

Bibliografia e sitografia

 John E. Hopcroft, R. Motwani, Jeffrey D. Ullman, *Automi, linguaggi e calcolabilità*

 *Wikipedia*, http://it.wikipedia.org

 *Youtube*, https://www.youtube.com/