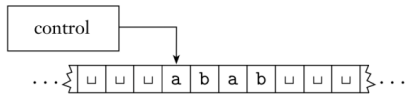
Una macchina di Turing è un modello matematico di calcolo che descrive una macchina astratta che manipola i simboli su una striscia di nastro secondo una tabella di regole. Nonostante la semplicità del modello, è in grado di implementare qualsiasi algoritmo informatico.

La macchina funziona su un nastro di memoria infinito diviso in celle discrete, ciascuna delle quali può contenere un singolo simbolo tratto da un insieme finito di simboli chiamato alfabeto della macchina.



- un nastro infinito come memoria illimitata
- una testina che legge e scrive simboli sul nastro
- all'inizio il nastro contiene l'input
- per memorizzare informazione si scrive sul nastro
- la testina si può muovere ovunque sul nastro
- stati speciali per accetta e rifiuta

Una Macchina di Turing (o Turing Machine, TM) è una tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$:

- Q è l'insieme finito di stati
- Σ è l'alfabeto di input che non contiene il simbolo blank $\sqcup$
- Γ è l'alfabeto del nastro che contiene $\sqcup$ e Σ
- $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$ è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale
- $q_{accept} \in Q$ è lo stato di accettazione
- $q_{reject} \in Q$ è lo stato di rifiuto (diverso da q_{accept})

1. Una TM può sia scrivere che leggere sul nastro
2. Una TM può muoversi sia a destra che a sinistra
3. Il nastro è infinito
4. Gli stati di rifiuto e accettazione hanno effetto immediato

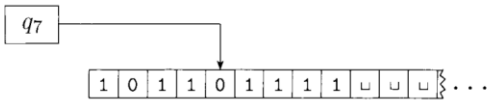


FIGURE 3.4
A Turing machine with configuration $1011q_701111$

1.1 Interpretazione della funzione di transizione

La funzione di transizione δ determina l'evoluzione della macchina in base allo stato interno attuale $q \in Q$ e al simbolo $X \in \Gamma$ letto dalla testina (quello presente nella cella del nastro attualmente visitata dalla testina):

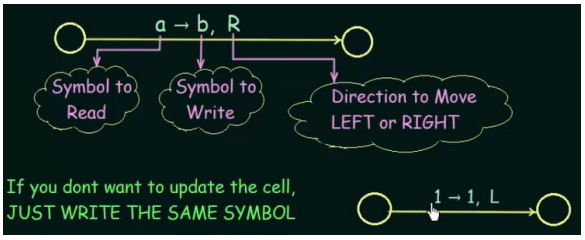
- Se $\delta(q, X)$ è indefinita, allora la macchina è **bloccata**, cioè la computazione della macchina si arresta.
- Se invece $\delta(q, X)$ è definita, e in particolare $\delta(q, X) = (p, Y, D)$, allora la macchina esegue la seguente mossa:
  - passa dallo stato interno q a p (si osservi che può essere $p = q$, dunque lo stato può non cambiare);
  - scrive il simbolo Y nella cella attualmente visitata, al posto di X (ma può essere $Y = X$);
  - sposta la testina a sinistra se $D = \texttt{L}$, o a destra se $D = \texttt{R}$ (in questa formalizzazione, la testina non può restare ferma).

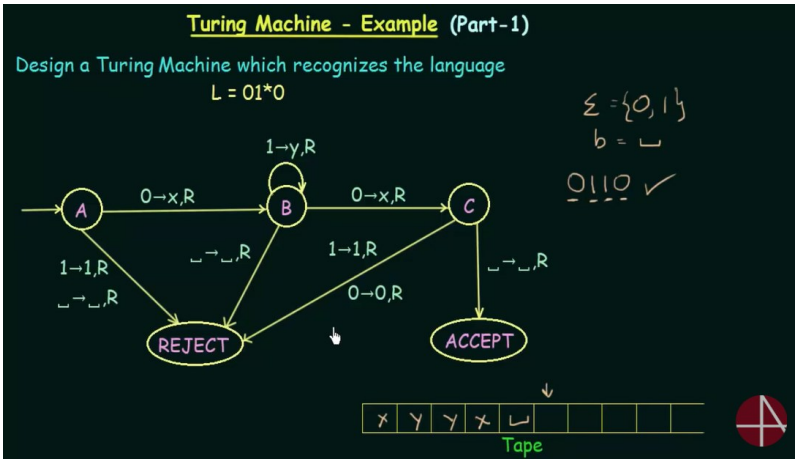
La configurazione C_1 produce C_2 se la TM può passare da C_1 a C_2 in un passo

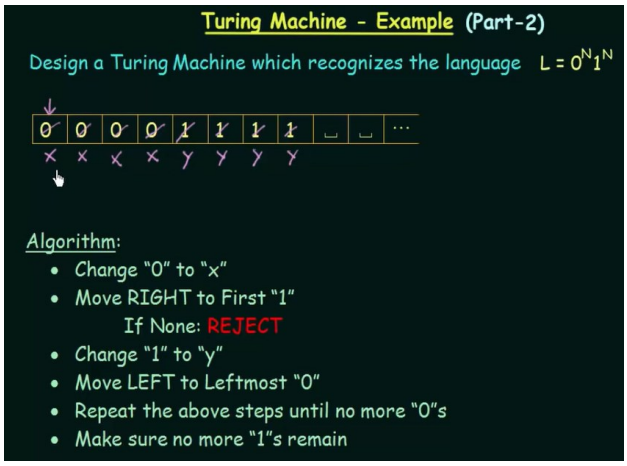
Se $a, b, c \in \Gamma$, $u, v \in \Gamma^*$ e q_i, q_j sono stati, allora:

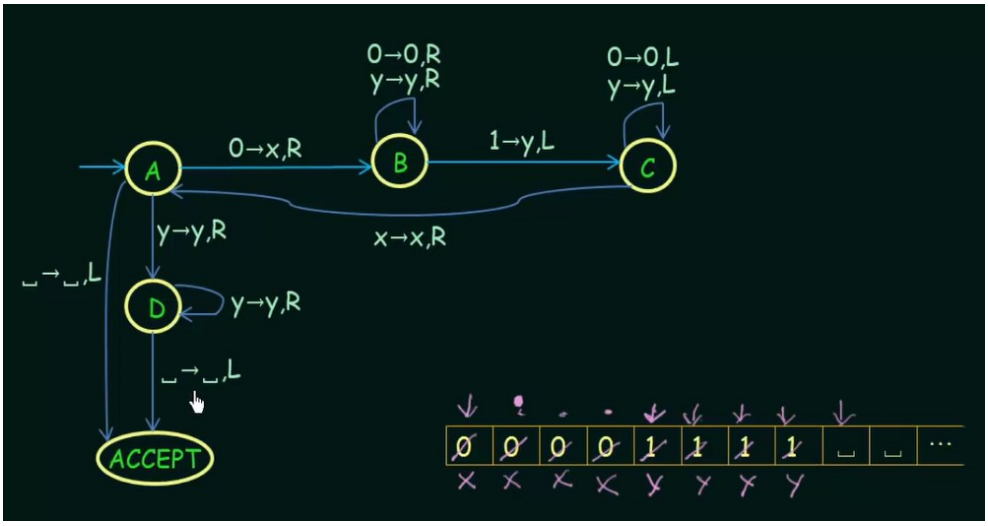
uaq_ibv produce uq_jacv se $\delta(q_i, b) = (q_j, c, L)$

uaq_ibv produce $uacq_jv$ se $\delta(q_i, b) = (q_j, c, R)$









- Un linguaggio è Turing-riconoscibile (o anche ricorsivamente enumerabile) se esiste una macchina di Turing che lo riconosce
- Se forniamo un input ad una TM, ci sono tre risultati possibili:
  - la macchina accetta
  - la macchina rifiuta
  - la macchina va in loop e non si ferma mai
  - una TM che termina sempre la computazione è un decisore
- Un linguaggio è Turing-decidibile (o anche ricorsivo) se esiste una macchina di Turing che lo decide

- Descrizione formale
  - Dichiara esplicitamente tutto quanto
  - Estremamente dettagliata
  - Da evitare a tutti i costi !!!
- Descrizione implementativa
  - Descrive a parole il movimento della testina e la scrittura sul nastro
  - Nessun dettaglio sugli stati
- Descrizione di alto livello
  - Descrizione a parole dell'algoritmo
  - Nessun dettaglio implementativo
  - Da utilizzare sempre, se non indicato altrimenti

Commento di: https://turingmachine.io/?import-gist=56dcb0347f17a48392f670c5c5009cd7

Esempi

1. Give an implementation-level description of a Turing machine that decides the language $B = \{ 0^n1^n2^n \mid n \geq 0 \}$.

M = "On input string w:

1. Scan the input from left to right to make sure that it is a member of $0^*1^*2^*$, and *reject* if it isn't.
2. Return tape head to left-hand end of tape.
3. Repeat the following until no more 0s left on tape.
4. Replace the leftmost 0 with x.
5. Scan right until a 1 occurs. If there are no 1s, *reject*.
6. Replace the leftmost 1 with x.
7. Scan right until a 2 occurs. If there are no 2s, *reject*.
8. Replace the leftmost 2 with x.
9. Return tape head to left-hand end of tape, and go to stage 3.
10. If the tape contains any 1s or 2s, *reject*. Otherwise, *accept*."

1. $\{w \in \{a, b\}^* \mid w$ contains as many a as $b\}$;

1.M = On input string w:

1. *Scan the tape and mark the first a that has not been marked. If no unmarked a has been found, go to step 3. Otherwise, bring the head back to the leftmost cell of the tape.*
2. *Scan the tape and mark the first b that has not been marked. If no unmarked b has been found,* reject. *Otherwise, bring the head back to the leftmost cell of the tape, and go back to step 1.*
3. *Bring the head back to the leftmost cell of the tape. Scan the tape: if there is an unmarked b,* reject*; otherwise,* accept.

3. Exercise: Design a Turing machine that recognizes the language `L = {w#w^R | w is a string over {a, b}}`, where `w^R` is the reversal of the string `w`.

Sull'input w#w^R (dove w^R è l'inversione di w):

1. Eseguire la scansione dell'input da sinistra a destra finché non viene incontrato il simbolo '#', che contrassegna la posizione corrente.

2. Continua la scansione verso destra, ricordando i simboli incontrati fino alla fine del nastro.

3. Torna alla posizione contrassegnata (subito dopo "#").

4. Confronta i simboli a destra di "#" con i simboli ricordati da destra a sinistra.

5. Se viene rilevata una mancata corrispondenza, rifiutare l'input.

6. Se la fine del nastro viene raggiunta senza discrepanze, accettare l'input come una stringa valida nel formato w#w^R

3.7 Explain why the following is not a description of a legitimate Turing machine.

M_{bad} = "On input $\langle p \rangle$, a polynomial over variables $x_1, \ldots, x_k$:

1. Try all possible settings of $x_1, \ldots, x_k$ to integer values.
2. Evaluate p on all of these settings.
3. If any of these settings evaluates to 0, *accept*; otherwise, *reject*."

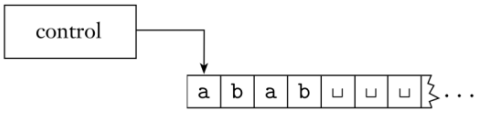
Solution: All possible settings of x_i are infinite, as there are infinite integers. Therefore, step 2 will never yield to step 3, so if a reject state *should* occur (i.e. there are no settings of x_i that make p evaluate to 0), we would never be able to reach that reject state, and if an accept state *should* be reached (i.e. we have found a suitable setting), we still have to iterate through infinite settings before being allowed to reach that state. Looping on its own is obviously not a problem, (you do not need to halt on all inputs to be a Turing-Recognizable language), however this machine puts its accept and reject states after a never ending search through infinite combinations of integers, so it won't halt on any input at all. So it's not a legitimate Turing Machine because it does not have a defined reject state

or accept state – they literally *never* can be reached and the machine will always loop forever.

Varianti di TM

Esistono definizioni alternative delle macchine di Turing. Chiamiamo varianti queste alternative e diciamo che tutte le varianti "ragionevoli" riconoscono la stessa classe di linguaggi, essendo robusti.

Macchine a nastro semi-infinito



- è una TM con un nastro infinito solo verso destra
- l'input si trova all'inizio del nastro
- la testina parte dalla posizione più a sinistra del nastro
- se M tenta di spostare la testina a sinistra quando si trova nella prima cella del nastro, allora la testina rimane ferma

- funzione di transizione:

$$\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R\}^k$$

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L) :$$

- se lo stato è q_i e le testine leggono $a_1, \ldots, a_k$
- allora scrivi $b_1, \ldots, b_k$ sui k nastri
- muovi ogni testina a sinistra o a destra come specificato

Esiste un “se e solo se” in questo tipo di dimostrazione.

Theorem

1. *Per ogni TM a nastro semi-infinito esiste una TM a nastro infinito equivalente.*
2. *Per ogni TM a nastro infinito esiste una TM a nastro semi-infinito equivalente.*

- S simula M
- i nastri sono separati da $\#$
- un punto sopra un simbolo indica la posizione della testina

$S =$ “Su input $w = w_1 \dots w_n$:

1. Inizializza il nastro per rappresentare i k nastri:

$$\#\dot{w}_1 w_2 \dots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \dots \#$$

2. Per simulare una mossa di M, scorri il nastro per determinare i simboli puntati dalle testine virtuali
3. Fai un secondo passaggio del nastro per aggiornare i nastri virtuali secondo la funzione di transizione di M.
4. Se S sposta una testina virtuale a destra su un $\#$, allora M ha spostato la testina sulla parte vuota del nastro. Scrivi un $\sqcup$ e sposta il contenuto del nastro di una cella a destra
5. Se si raggiunge una configurazione di accettazione, accetta, se si raggiunge una configurazione di rifiuto, rifiuta, altrimenti ripeti da 2.

$\Rightarrow$ Un linguaggio è Turing-riconoscibile se è riconosciuto da una TM con un solo nastro, che è un caso particolare di TM multinastro

$\Leftarrow$ Costruzione precedente

Supponiamo di avere una macchina di Turing M con un solo nastro che riconosce un linguaggio L. Possiamo costruire una macchina di Turing multinastro M' con k nastri che simula M e riconosce lo stesso linguaggio L nel seguente modo:

1. M' utilizza il primo dei suoi k nastri per simulare il nastro di M. Gli altri k-1 nastri vengono utilizzati come "nastri di lavoro" aggiuntivi.
2. L'input w viene inizialmente scritto sul primo nastro di M', proprio come sul nastro di M.
3. M' mantiene una codifica della configurazione corrente di M sui suoi nastri di lavoro utilizzando una rappresentazione ragionevole (ad esempio, codificando lo stato corrente, la posizione della testina e il contenuto del nastro).
4. Per simulare ogni transizione di M, M' decodifica la configurazione corrente dai nastri di lavoro, applica la funzione di transizione di M per determinare la nuova configurazione, quindi codifica la nuova configurazione sui nastri di lavoro.
5. Se durante la simulazione M' raggiunge una configurazione di accettazione di M, allora M' accetta. Analogamente, se M' raggiunge una configurazione di rifiuto di M, allora M' rifiuta.

Poiché M' è in grado di simulare fedelmente ogni passo di M, incluse le mosse della testina e gli aggiornamenti del nastro, M' riconoscerà esattamente lo stesso linguaggio L di M.

Quindi, dato che M è una macchina di Turing arbitraria che riconosce un linguaggio Turing-riconoscibile L, abbiamo costruito una macchina di Turing multinastro M' che riconosce lo stesso linguaggio L.

Macchine non deterministiche

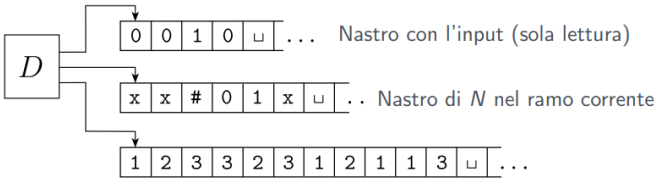
- Una TM non deterministica ha più strade possibili durante la computazione
- Consideriamo macchine con un solo nastro semi-infinito
- La funzione di transizione è:

$$\delta : Q \times \Gamma \mapsto 2^{(Q \times \Gamma \times \{L,R\})}$$

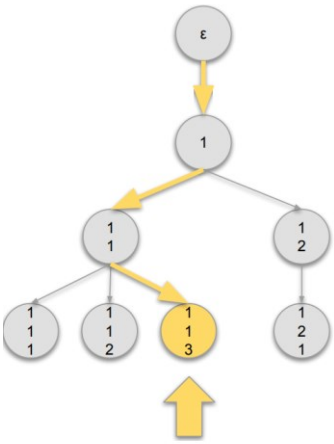
Theorem

Per ogni TM non deterministica N esiste una TM deterministica D equivalente.

Idea:



Il terzo nastro tiene traccia delle scelte non deterministiche



- Ad ogni nodo viene assegnato un indirizzo: una stringa sull'alfabeto $\Gamma_b = \{1, 2, \ldots, b\}$, dove b è il massimo numero di figli dei nodi dell'albero
- Il nodo 113 si raggiunge prendendo il primo figlio della radice, seguito dal primo figlio di quel nodo ed infine dal terzo figlio.
- Questo ordinamento può essere utilizzato per attraversare in modo efficiente l'albero in ampiezza.

1. Inizialmente il nastro 1 contiene l'input w e i nastri 2 e 3 sono vuoti
2. Copia il nastro 1 sul nastro 2 e inizializza la stringa sul nastro 3 a ε
3. Usa il nastro 2 per simulare N con input w su un ramo di computazione.
   Prima di ogni passo di N, consulta il simbolo successivo sul nastro 3 per determinare quale scelta fare (tra quelle consentite).

3. (cont.) Se non rimangono più simboli sul nastro 3, o se questa scelta non è valida, interrompi questo ramo e vai alla fase 4.
   Vai alla fase 4 anche se si incontra una configurazione di rifiuto.
   Se viene trovata una configurazione di accettazione, accetta.
4. Sostituire la stringa sul nastro 3 con la stringa successiva nell'ordine delle stringhe. Simula il ramo successivo di N andando alla fase 2.

Un linguaggio è Turing-riconoscibile se e solo se esiste una macchina di Turing non deterministica che lo riconosce.

(Come sopra: costruzione precedente)

3.12 A ***Turing machine with left reset*** is similar to an ordinary Turing machine, but the transition function has the form

$$\delta\colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{R}, \mathrm{RESET}\}.$$

If $\delta(q, a) = (r, b, \mathrm{RESET})$, when the machine is in state q reading an a, the machine's head jumps to the left-hand end of the tape after it writes b on the tape and enters state r. Note that these machines do not have the usual ability to move the head one symbol left. Show that Turing machines with left reset recognize the class of Turing-recognizable languages.

Solution: We can simulate a regular Turing Machine using the TM with left reset in the following way: When the regular TM would move right, it moves right; and when the regular TM would move left, it marks its current space, resets to the left, copies every space one space to the right (maintaining the position of the mark), then resets and moves to the position of the mark. The copying can be done in a single sweep, by simultaneously remembering the symbol that is being copied and the symbol that is being overwritten.

Bonus: we can also simulate the left-reset TM in the regular TM. When the left-reset TM would move right, regular TM moves right; and when the left-reset TM would reset to the left, the regular TM moves left repeatedly not changing any letters it sees until it gets to the left end of the tape.

1. *Una* macchina di Turing bidimensionale *utilizza una griglia bidimensionale infinita di celle come nastro. Ad ogni transizione, la testina può spostarsi dalla cella corrente ad una qualsiasi delle quattro celle adiacenti. La funzione di transizione di tale macchina ha la forma*

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\uparrow, \downarrow, \rightarrow, \leftarrow\},$$

dove le frecce indicano in quale direzione si muove la testina dopo aver scritto il simbolo sulla cella corrente.

Dimostra che ogni macchina di Turing bidimensionale può essere simulata da una macchina di Turing deterministica a nastro singolo.

Mostriamo come simulare una TM bidimensionale B con una TM deterministica a nastro singolo S. S memorizza il contenuto della griglia bidimensionale sul nastro come una sequenza di stringhe separate da #, ognuna delle quali rappresenta una riga della griglia. Due cancelletti consecutivi ## segnano l'inizio e la fine della rappresentazione della griglia. La posizione della testina di B viene indicata marcando la cella con $\hat{}$. Nelle altre righe, un pallino $\bullet$ indica che la testina si trova su quella colonna della griglia, ma in una riga diversa. La TM a nastro singolo S funziona come segue:

S = "su input w:

1. Sostituisce w con la configurazione iniziale $\#\#w\#\#$ e marca con $\hat{}$ il primo simbolo di w.
2. Scorre il nastro finché non trova la cella marcata con $\hat{}$.
3. Aggiorna il nastro in accordo con la funzione di transizione di B:
   - Se $\delta(r, a) = (s, b, \rightarrow)$, scrive b non marcato sulla cella corrente, sposta $\hat{}$ sulla cella immediatamente a destra. Poi sposta di una cella a destra tutte le marcature con un pallino. Se in qualsiasi momento S sposta una marcatura sopra un #, S scrive un blank marcato al posto del # e sposta il contenuto del nastro da questa cella fino al ## finale, di una cella più a destra.
   - Se $\delta(r, a) = (s, b, \leftarrow)$, scrive b non marcato sulla cella corrente, sposta $\hat{}$ sulla cella immediatamente a sinistra. Poi sposta di una cella a sinistra tutte le marcature con un pallino. Se in qualsiasi momento S sposta una marcatura sopra un #, S scrive un blank marcato al posto del # e sposta il contenuto del nastro da questa cella fino al ## iniziale, di una cella più a sinistra.
   - Se $\delta(r, a) = (s, b, \uparrow)$, scrive b marcato con un pallino nella cella corrente, e sposta $\hat{}$ sulla prima cella marcata con un pallino posta a sinistra della cella corrente. Se questa cella marcata non esiste, aggiunge una nuova riga composta solo da blank all'inizio della configurazione.
   - Se $\delta(r, a) = (s, b, \downarrow)$, scrive b marcato con un pallino nella cella corrente, e sposta $\hat{}$ sulla prima cella marcata con un pallino posta a destra della cella corrente. Se questa cella non esiste, aggiunge una nuova riga composta solo da blank alla fine della configurazione.
4. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di B, allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di B allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."

3.9 Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognize a larger class of languages) than 0-PDAs.

a. Show that 2-PDAs are more powerful than 1-PDAs.

b. Show that 3-PDAs are not more powerful than 2-PDAs.
(Hint: Simulate a Turing machine tape with two stacks.)

Un modo per farlo è simulare un nastro della macchina di Turing utilizzando due stack in un 2-PDA. Consideriamo un linguaggio L costituito da tutte le stringhe della forma ww^R, dove w è una stringa qualsiasi e w^R denota il contrario di w.

Possiamo costruire un 2-PDA che riconosca questo linguaggio come segue:

1. Inizialmente, il 2-PDA spinge tutti i simboli dell'input sul primo stack.

2. Quindi, estrae i simboli dalla prima pila e li inserisce nella seconda pila, invertendo di fatto l'input.

3. Successivamente, inizia a confrontare i simboli in cima a entrambi gli stack. Se sono uguali, si continua con i simboli successivi.

4. Se in qualsiasi momento i simboli in cima alle pile sono diversi, o se una pila si svuota prima dell'altra, il 2-PDA rifiuta l'input.

5. Infine, se entrambi gli stack si svuotano contemporaneamente, il 2-PDA accetta l'input.

Questo 2-PDA riconosce la lingua L, poiché può verificare se l'input è nella forma ww^R confrontando i simboli sui due stack.

Tuttavia, un 1-PDA non può riconoscere questa lingua. Questo perché un 1-PDA ha una memoria limitata e non può confrontare in modo efficiente i simboli in posizioni arbitrarie nell'input. Pertanto, un 2-PDA è più potente di un 1-PDA.

3.9 b Clearly a 3-PDA is at least as powerful as a 2-PDA (for the same reason that a 2-PDA is at least as powerful as a 1-PDA). Also, a 3-PDA is no more powerful than a 3-tape TM, since any 3-PDA could be simulated by a 3-tape TM. Therefore, if we can show that a 2-PDA is as powerful as a 3-tape TM, then it must be the case that 3-PDA's and 2-PDA's have the same power. Since 3-tape TM's have the same power as 1-tape TM's, it suffices to show that 2-PDA's have the same power as 1-tape TM's.

The configuration uqv can be simulated on a 2-PDA by having u on stack 1 with the beginning of u on the bottom of the stack and the end of u at the top of the stack, and v on stack 2 with the beginning of v on the top of the stack and the end of v on the bottom of the stack. In addition, below the last character of v on stack 2 is a ~ to mark the end of the word.

The 2-PDA will have the same states as the TM. The Turing transitions can be simulated by carefully moving symbols from one stack to the other. If the TM has a transition where uaq_ibv yields uq_jacv, and the 2-PDA is in state q_i with b on the top of stack 2, the 2-PDA should change the b to a c, pop stack 1, push the popped symbol on to stack 2, and go to state q_j. Similarly, if uaq_ibv yields $uacq_jv$, and the 2-PDA is in state q_i with b on the top of stack 2, the 2-PDA should pop the b off stack 2, push a c on to stack 1, and go to state q_j.

Problemi trattabili e non trattabili → Hilbert

Let's phrase Hilbert's tenth problem in our terminology. Doing so helps to introduce some themes that we explore in Chapters 4 and 5. Let

$$D = \{p \mid p \text{ is a polynomial with an integral root}\}.$$

Hilbert's tenth problem asks in essence whether the set D is decidable. The answer is negative. In contrast, we can show that D is Turing-recognizable. Before doing so, let's consider a simpler problem. It is an analog of Hilbert's tenth problem for polynomials that have only a single variable, such as $4x^3 - 2x^2 + x - 7$. Let

$$D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with an integral root}\}.$$

Here is a TM M_1 that recognizes D_1:

M_1 = "On input $\langle p \rangle$: where p is a polynomial over the variable x.

1. Evaluate p with x set successively to the values $0, 1, -1, 2, -2, 3, -3, \ldots$. If at any point the polynomial evaluates to 0, *accept*."

Le TM sono un mezzo potente e servono ad esprimere qualsiasi tipo di linguaggio o algoritmo:

2. (12 punti) Dati due DFA, considera il problema di determinare se esiste una stringa accettata da entrambi.

(a) Formula questo problema come un linguaggio $AGREE_{\text{DFA}}$.

(b) Dimostra che $AGREE_{\text{DFA}}$ è decidibile.

Soluzione.

(a) $AGREE_{\text{DFA}} = \{\langle A, B\rangle \mid A, B \text{ sono DFA, ed esiste una parola } w \text{ tale che } w \in L(A) \text{ e } w \in L(B)\}$

(b) La seguente macchina N usa la Turing machine M che decide E_{DFA} per decidere $AGREE_{\text{DFA}}$:

N = "su input $\langle A, B\rangle$, dove A, B sono DFA:

1. Costruisci il DFA C che accetta l'intersezione dei linguaggi di A e B
2. Esegui M su input $\langle C\rangle$. Se M accetta, rifiuta, se M rifiuta, accetta."