

Parti da ,, CFG del linguaggio L di partenza.

Creiamo una nuova grammatica in forma normale di Chomsky per il nostro linguaggio

Struttura (più pratica):

* Regola iniziale
* Produzione
  + Abbiamo una nuova regola che per ogni A e B produzione ci permette di aggiungere un cancelletto
* Una volta messo il cancelletto, otteniamo solo simboli terminali
  + (regola terminale = finisce producendo un carattere senza mettere di nuovo il cancelletto)

Struttura più formale:

* Per ogni variabile di , conteniamo sia che che vengono prodotte da
* Tutte le stesse regole di in
* , abbiamo delle regole intermedie ,
* Per ogni regola di , la regola
* Immagine che contiene testo, schermata, Carattere, linea

  Descrizione generata automaticamenteSe incontriamo un cancelletto nella produzione intermedia, la grammatica produrrà oppure

per variabile iniziale e rispetta .

1. è una CFG, , è persistente
2. Descrizione di una TM che fa quello che deve fare il problema (1)

Adesso facciamo “una descrizione a livello implementativo” / “ad alto livello”

Es. “per visualizzare il concetto logico di Macchina di Turing”

TM ##->[][][][][][][][]## (dove [] è una cella di memoria, -> è la testina, # è il confine (bound))

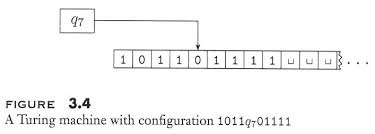
[a][w]awawawawawawaww (qua stiamo descrivendo a parole quello che vorremmo, in questo caso quando rifiuta; ad ogni regola, non ho sempre il fatto di “quando metti “a” allora metti “w”)

Sul punto in rosso rifiuta.

Se avessimo invece un input:

[a][w]awawawawawawawa

Allora accetterebbe! (Se fosse un DFA, si direbbe “termina uno stato di accettazione/stato finale)



Descrizione regolare:

è una TM, prende in input , CFG e

* Verifica che ogni regola di dia in output una produzione che contiene
* Se non vede in una regola la stringa rifiuta
* Continua finché non esaurisce tutte le regole
* Se accetta allora accetta, altrimenti rifiuta

Dimostriamo che è un decisore 🡪 si comporta sempre in modo deciso

* Se input , allora è persistente, in quanto la TM M verifica che ogni regola produca correttamente la variabile richiesta per ogni singolo passaggio. Se così non fosse, allora rifiuta altrimenti accetta.
* Se è persistente, dato l’input , la TM rifiuta dato che ogni regola non produce la stringa richiesta, quindi non può essere definita persistente.

Volendo per riduzione 🡪 vedi gli indecidibili.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamenteSe è decidibile, allora .

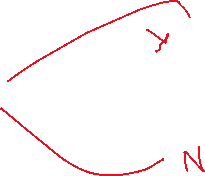


Se è indecidibile, allora è indecidibile.

* è una funzione di riduzione con , M è una TM (attento: che fa seguendo la descrizione di ) e una stringa)
  + M copia l’input sul nastro
  + Se , va in loop o rifiuta
  + Se , esegui
    - Se accetta, allora accetta
    - Altrimenti rifiuta
* Restituisci

Mostriamo che calcola una funzione di riduzione da :

* Se , la macchina trova sul nastro e si ferma accettando
* Se , la macchina rifiuta o va in loop in quanto sul nastro non era presente 42 come input. Quindi .



Complemento di A\_TM ≤m F se e solo se M' ∉ F.

In altre parole, il complemento di ATM, indicato come ATM, è l'insieme delle Turing machine M tali che (M, w) appartiene ad ATM se e solo se M' non appartiene a F.

Se sei nel caso negativo (rifiuta/va in loop) parte il complemento di A\_TM e se realizzi le condizioni del problema (esempio – Forty-Two) allora rifiuti. In questo caso, vuol dire che per chiusura

Esempio completo:

Sia (M, w) un'istanza di . Definiamo la TM M' che, su input x:

1. Esegue M su input w.
2. Se M accetta w, accetta.
3. Altrimenti (se M rifiuta o va in loop su w), calcola per 42 passi sul nastro (per vedere se appare 42) e poi si ferma rifiutando.

Mostriamo che (M, w) ∈ se e solo se (M', w) ∈ FORTY-TWO:

⇒) Se (M, w) ∈ , allora M termina la computazione su w. Quindi per costruzione M' simulerà M su w e poi accetterà. Inoltre terminerà avendo usato solo 42 celle del nastro (o meno). Quindi (M', w) ∈ FORTY-TWO.

⇐) Se (M', w) ∈ FORTY-TWO, allora M' termina la computazione su w usando solo 42 celle del nastro. Per come abbiamo definito M', ciò può accadere solo se M termina su w, quindi (M, w) ∈ .

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

che simula

= TM a nastro singolo che su input :

* Ricopio l’input sul nastro
* Se , controlliamo dove esiste la prima cella con valore P. Se non esiste nessuna cella con P, la macchina rifiuta (nel caso in cui abbiamo appena iniziato l’input).
  + Se la cella P non è stata marcata, rappresenta un nuovo padre
  + In questo caso, passo al primo puntatore L e scorro tutti i figli sinistri
  + Marco dove ho trovato l’ultimo L
  + Parto a scorrere tutti i figli destri dal primo R
  + Tutti questi saranno scorsi fino al prossimo P, che viene poi marcato
* Se , controlliamo di essere nella prima cella dopo un padre P marcato.
  + Se ciò avviene, comincio a scorrere tutti i simboli a sinistra dell’albero finché non ho finito i simboli che sono a sinistra prima di un prossimo P
  + Una volta finiti questi simboli, uso un marcatore (°) per indicare che ho finito i simboli di sinistra
  + Proseguo con i simboli di destra R
* Se , controlliamo di essere nella prima cella dopo un padre P marcato
  + Se ciò avviene, comincio a scorrere tutti i simboli a destra dell’albero finché non ho finito i simboli che sono a destra prima di un prossimo P
  + Una volta finiti questi simboli, uso un marcatore (\*) per indicare che ho finito i simboli di destra
  + Avrò un puntatore ad un padre P (^) che mi dice “torna al nodo padre precedente”
* Trovato un padre precedente, vado al padre successivo e continuo così scorrendo tutti gli input finché esistono dei padri P
* Se M arriva in uno stato di accettazione, allora accetta altrimenti rifiuta
* Altrimenti, riprendi la simulazione di cui sopra (Padre/Sinistro/Destro)