CLF

Cap 2 - Linguaggi Context-Free

Cap 2.1 Grammatiche Context-Free

Una quadrupla (V, $\sum$, R, S) dove:

- V è insieme delle variabili
- $\sum$ è insieme terminali
- R è insieme delle regole
- S è stato iniziale

Progettare grammatiche context-free Prima si costruisce il DFA per il linguaggio regolare, poi lo si trasforma in CFG.

- Una variabile R per ogni stato
- Regola Ri -> aRq se c'è transazione da stato i a stato q tramite a nel DFA
- Regola Ri -> ϵ se stato i è accettante

Grammatiche possono generare stessa stringa in modi diversi, sono dette ambigue. Se linguaggi possono essere generati solo così, sono detti inerentemente ambigui. Es: aibjck | i = j oppure j = k

Forma normale di Chomsky Se ogni regola della grammatica è della forma:

- A -> BC
- A -> a

Per ottenerla:

- Aggiungiamo nuova variabile iniziale
- Eliminiamo le regola A -> ϵ, poi in ogni regola dove appare A nel lato destro, scriviamo tutte le occorenze senza la A.
  - A -> ϵ fa diventare S -> ASA | aB in S -> ASA | aB | SA | AS | S
- Eliminiamo le regole unitarie A -> B, poi aggiungiamo A -> u per la regola B -> u, dove u è una stringa di variabili e terminali
- Rimpiazziamo le regole A->u1u2u3 in poi con A->u1A1 e A1->u2A2 e A2 -> u3.
- Infine convertimo le ui che sono terminali con una nuova regola Ui -> ui e sostituiamo i terminali.

Cap 2.2 Automi a Pila

Sono potenti automi che hanno a disposizione una pilla pushdown per contenere quantità non limitata di dati. Tra deterministici e non deterministici c'è differenza computazionale.

Automi a pila non deterministici sono computazionalmente equivalenti alle grammatiche context-free.

Una sestupla (Q, $\sum$, Γ, δ, q0, F) dove:

- Q è insieme degli stati
- $\sum$ è alfabeto dell'input
- δ è funzione transizione Qx$\sum\epsilon$ x$\Gamma\epsilon$ -> P(Qx$\Gamma\epsilon$)
- Γ è alfabeto della pila
- q0 è stato iniziale
- F è insieme degli stati accettanti

Nel diagramma di stato si scrive a,b -> c che vuol dire

- legge input a
- toglie b dalla cima della pila
- scrive c sulla pila

Se a è vuoto, può leggere e scrivere sulla pila senza input Se b è vuoto, può scrivere senza togliere dalla pila Se c è vuoto, può togliere dalla pila senza scrivere

Equivalenza con le grammatiche context-free Un linguaggio è context-free se e solo se esiste un automa a pila che lo riconosce.

Se un linguaggio è context-free, allora esiste un automa a pila che lo riconosce.

`Convertiamo da una CFG G a un PDA P`

Dobbiamo realizzare una pila che dato un input, riuscirà a generarlo. Alla fine li confronteremo e accetteremo se uguali. Per farlo lavoreremo sulle stringhe intermedie generate ad ogni derivazione. Quindi distinguiamo il comportamento in base a se troviamo una variabile o un terminale sulla cima della pila.

1) Inserisce simbolo marcato $ e la prima variabile nella pila
2) Se sulla cima della pila c'è A, scegli non deterministicamente una regola per A e sostituisce con la parte destra di A
3) Se sulla cima della pila c'è terminale a, legge l'input e lo confronta. Se sono uguali, ripete. Se sono diversi, rifiuta su questo ramo
4) Se sulla cima c'è il simbolo $, entra nello stato accettante. L'input è stato completamente letto
5) Ripete dal punto 2

Nuova notazione (r, u) $\in \delta$(q, a, s) Siamo nello stato q, leggiamo a dal input, s è nella cima della pila. Andremo allo stato r e nella pila sostituiremo s con u.

Nel diagramma del PDA avremo stato qstart con inserimento di $. Lettura variabile iniziale. Stato qloop con le regole per i terminali e sostituzione parte destra per le variabili. Stato qaccept quando leggo il $ dalla pila e sono quindi vuoto. ____

Se un linguaggio è riconosciuto da un automa a pila, allora è context-free.

`Convertiamo da PDA a CFG`

G dovrebbe generare una stringa, se quella stringa fa andare P da uno stato iniziale ad accettante.

Ideare una variabile Apq che porta la pila da p a q, nello stesso stato.

Il PDA sarà

- unico stato accettante
- svuota la pila prima di accettare
- una transizione o push o pop

Ogni linguaggio regolare è context-free. Questo perché un linguaggio regolare è riconosciuto da un automa finito. Quest'ultimo è un automa a pila che ignora la sua pila.

Cap 2.3 Linguaggi non Context-Free

Pumping Lemma Se A è linguaggio context-free, allora esiste un numero p tale che, se s è una stringa in A di lunghezza almeno p, può essere divisa in cinque parti s = uvxyz:

1) per ogni i >= 0, uvixyiz ∈ A
2) |vy| > 0
3) |vxy| <= p

Alcuni linguaggi non context-free:

- anbncn | n >= 0 -> pumping up
- aibjck | 0 <= i <= j <= k -> pumping up e pumping down
- ww | w ∈ {0, 1}* -> usando 0p1p0p1p

Cap 3 - La tesi di Church-Turing

Cap 3.1 Macchine di Turing

Simile ad un automa, memoria illimitata.

- Può sia scrivere sia leggere sul nastro
- La testina si muove a destra e sinistra
- Nastro è infinito
- Gli stati di accettazione e rifiuto sono immediati

Definizione formale di macchina di Turing Una setupla (Q, $\sum$, Γ, δ, q0, qaccept, qreject) dove:

- Q è insieme degli stati
- $\sum$ è alfabeto dell'input senza simbolo blank
- δ è funzione transizione QxΓ -> QxΓx{L, R}
- Γ è alfabeto del nastro con blank

- q0 è stato iniziale
- qaccept è stato di accettazione
- qreject è stato di rifiuto

La configurazione è lo stato della macchina durante la computazione, composto dalla posizione della testina e della situazione del nastro.

La configurazione C1 produce la configurazione C2 se la MdT passa tra le due in un unico passo.

Un linguaggio è Turing-riconoscibile se esiste una MdT che lo riconosce.

L'insieme delle stringhe che M accetta rappresenta il linguaggio riconosciuto da M.

Una stringa è accettata se produce una serie di configurazioni che terminano in uno stato di accettazione.

Le MdT che si fermano su ogni input sono dette decisori, in quanto decidono se accettare o rifiutare, non ciclano mai. Un decisore decide un linguaggio se riconosce tale linguaggio.

Un linguaggio è Turing-decidibile se esiste una MdT che lo decide.

Ogni linguaggio decidibile è Turing-riconoscibile.

Esempi di macchine di Turing Descrizione di come si muove la testina, dei controlli che fa sullo stato che è ed eventuali sostituzioni.

Cap 3.2 Varianti di macchine di Turing

Sono delle varianti che non vanno però a modificare la classe di linguaggi riconosciuti o aumentare il potere computazionale.

Macchina di Turing multinastro Inizialmente solo nastro 1 ha input, gli altri vuoti. Ogni testina è su un nastro.

Ogni macchina multinastro ha una MdT a nastro singolo equivalente.

Lo si fa usando il simbolo # per delimitare l'inizio e la fine dei nastri, inoltre si aggiunge il simbolo puntato per identificare la testina virtuale dei vari nastri.

Macchina di Turing non deterministica Avrà una funzione di transizione

$Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$

Ogni macchina di Turing non deterministica ha una MdT deterministica equivalente.

Si esegue un approccio di ricerca ad albero in ampiezza. La MdT D avrà 3 nastri con input, copia di N, stato di D rispetto all'esplorazione su N.

Quindi il nastro 3 avrà la scelta successiva del passo. Se vuota o non valida, resetta con la stringa successiva del cammino.

Questo per evitare loop su un cammino infinito.

N può diventare un decisore se si fermasse su ogni input di ogni ramificazione.

Enumeratori Ha una stampante collegata, che genera un output delle stringhe che compongono il linguaggio.

La TM esegue l'input e lo confronta con E. Se appare, accetta.

E invece ignora l'input ed esegue M su ogni stringa. Se M accetta, E stampa la stringa.

Cap 3.3 Definizione di algoritmo

I problemi di Hilbert Il decimo è verificare con un algoritmo se un polinomio ha una radice intera.

Riuscire a dimostrare l'esistenza o meno di un algoritmo, avendo però definito formalmente cos'è un algoritmo.

La tesi di Church-Turing dimostra il collegamento tra una nozione intuitiva e una definizione formale, di algoritmo.

Noi proviamo a verificare se è Turing-riconoscibile cercando una radice su un polinomio a singola variabile. Esso può diventare decidibile definendo poi i limiti entro il quale deve terminare. Questo modo però per trovare i limiti non è applicabile nel caso di più variabili. Ecco perché l'algoritmo del problema non esiste.

Terminologia per la descrizione di MdT Descriviamo l'input della macchina, che è spesso una stringa o trasformato in tale. Verifichiamo in caso che sia nella forma corretta.

Cap 4 - Decidibilità

Cap 4.1 Linguaggi decidibili

Problemi decidibili relativi a linguaggi regolari Problema dell'accettazione per DFA.

ADFA è il linguaggio per indicare se un automa finito deterministico accetta una data stringa.

Esso è decidibile. Quindi verificare il linguaggio è equivalente a verificare la decibilità del problema computazionale.

M decide ADFA:

1) Su input <B, w> dove B è un DFA e w è una stringa

2) Simula B su w
3) Se la simulazione termina in uno stato di accettazione, accetta. Se non termina in uno stato di accettazione, rifiuta.

ANFA è decidibile Trasformiamo l'NFA in un DFA equivalente C. Eseguiamo C su M. Accetta, se accetta. Altrimenti rifiuta.

AREX è decidibile Converte in un NFA equivalente. Esegue su N. Accetta, se accetta. Altrimenti rifiuta.

EDFA è decidibile Test del vuoto. {<A> | A è un DFA e L(A) = $\emptyset$}

1) Marca stato iniziale di A
2) Ripete finché non sono marcati nuovi stati:
3) Marca uno stato che ha una transizione proveniente da uno stato già marcato
4) Se nessuno stato di accettazione è marcato, accetta. Altrimenti rifiuta.

EQDFA è decidibile Costruiamo linguaggio L(C) = (L(A) interseca $\overline{L(B)}$) U ($\overline{L(A)}$ interseca L(B))

Solo stringhe riconosciute da A o da B, differenza simmetrica.

Se L(C) è vuoto allora L(A) e L(B) sono uguali.

1) Costruiamo DFA C
2) Eseguiamo TM di E su input C
3) Accetta, se accetta. Altrimenti rifiuta.

Problemi decidibili relativi a linguaggi context-free

ACFG è decidibile Per essere sicuri sia un decisore trasformiamo nella forma di Chomsky, così avremo al massimo 2n-1 passi dove n è la lunghezza di w. TM S

1) Convertiamo G in una grammatica equivalente in forma normale di Chomsky
2) Lista tutte le derivazioni di 2n-1 passi o un passo se n=0
3) Se una di queste genera w, accetta. Altrimenti rifiuta.

ECFG è decidibile Non possiamo usare teorema precedente dato che ci possono essere infinite w. Partiamo quindi al contrario cercando tutte le variabili che generano stringhe di terminali.

1) Marca tutti i terminali di G
2) Ripete finché nessuna nuova variabile viene marcata:
3) Marca una variabile A che ha la regola in G, A -> U1U2Un dove ogni simbolo è già stato marcato.

4) Se la variabile iniziale non è segnata, accetta. Altrimenti rifiuta.

EQCFG non è decidibile Questo perché la classe dei linguaggi context-free non è chiusa rispetto a complemento o intersezione.

Ogni linguaggio context-free è decidibile A è un CFL.

G una CFG per A, progettiamo una TM M che decide A.

Creiamo una copia di G in M. Su input w

1) Esegue TM S su G, w
2) Se S accetta, accetta. Altrimenti rifiuta.

Turing-riconoscibili Decidibili Context-free Regolari

Cap 4.2 Indecidibilità

ATM non è decidibile Essendo però riconoscibile, dimostra che i riconoscitori sono più potenti dei decisori.

Macchina universale U che cicla su M se M cicla su w.

Il metodo della diagonalizzazione Serve a dimostrare l'indecidibilità di ATM.

Funzione iniettiva e suriettiva è detta biettiva: per ogni elemento di B esiste un solo elemento di A. Ogni elemento di A è mappato in unico elemento in B. Se tale funzione esiste, A e B hanno la stessa cardinalità.

Insieme R dei numeri reali non è numerabile.

L'insieme dei linguaggi è non numerabile, mentre l'insieme delle MdT si. Quindi alcuni linguaggi non sono ne decidibili ne turing-riconoscibili.

Un linguaggio indecidibile Dimostriamo per assurdo, usando un decisore H che si ferma su M.

- Accetta se M accetta w
- Rifiuta se M non accetta w

Costruiamo una MT D che chiama H per determinare M.

- Esegue H su input <M, <M> >
- Accetta se H rifiuta, rifiuta se H accetta

Se provassimo D(<D>) avremmo

- Accetta se D non accetta
- Rfiuta se D non rifiuta

Ovviamente una contraddizione. Quindi ne D ne H possono esistere.

Un linguaggio non Turing-riconoscibile Se un linguaggio ed il suo complemento sono turing-riconoscibili, allora è decidibile.

Se un linguaggio è decidibile allora è riconoscibile. Il complemento di un linguaggio decidibile è anch'esso decidibile.

M1 riconosce A e M2 riconosce $\overline{A}$, M è decisore per A.

1) Esegue sia M1 sia M2 su w in parallelo
2) Se M1 accetta, accetta; se M2 accetta, rifiuta.

Dato che M si ferma se una delle due macchine accetta, è un decisore. Inoltre accetta solo le stringhe di A, altrimenti respinge, quindi è un decisore per A, ed A è decidibile.

$\overline{A\ TM}$ **non è turing-riconoscibile** Questo perché ATM è riconoscibile e abbiamo visto essere non decidibile, quindi $\overline{A\ TM}$ non può essere coTuring riconoscibile.

Cap 5 - Riducibilità

Ridurre un problema in modo che risolvendo il secondo, si abbia anche una soluzione al primo.

Se A è riducibile a B, e A è indecidibile allora anche B lo è.

Cap 5.1 Problemi indecidibili dalla teoria dei linguaggi

Problema della fermata, decidere se una MdT si ferma dato un input.

HALTTM è indecidibile Per assurdo supponiamo che è decidibile e dimostriamo che ATM è riducibile a HALTTM.

Abbiamo R che decide HALTTM. Costruiamo S per decidere ATM.

- Esegue R su (M, 2)
- Se R rifiuta, rifiuta
- Se R accetta, simula M su w finchè non si ferma
- Se M accetta, accetta. Se rifiuta, rifiuta

Se R decide HALTTM allora S dedice ATM ma è indecidibile, quindi anche HALTTM lo è.

ETM è indecidibile Qui avremo S che esegue una modifica di M1, dove controllo che l'input sia = a w.

- Costruiamo M1
- Esegue R su M1
- Se R accetta, rifiuta. Se R rifiuta, accetta

Se R decide ETM allora S dedice ATM ma è indecidibile, quindi anche ETM lo è.

REGULARTM è indecidibile

- Costruiamo M2
- Se x ha forma 0n1n, accetta
- Se non ha forma, esegue M e accetta, se accetta
- Esegue R su M2
- Se R accetta, accetta. se R rifiuta, rifiuta

EQTM è indecidibile Riduciamo da ETM, quindi fissiamo che una delle due TM abbia il linguaggio vuoto.

- Eseguiamo R su (M, M1) dove M1 rifiuta ogni input
- Se R accetta, allora accetta. Se R rifiuta, rifiuta.

Se R dedice EQTM, S dedice ETM. Questo non è possibile.

Riduzioni mediante storie di computazione Una storia di computazione è l'insieme di configurazioni da C1 a Cl dove Cl accetta per M.

Automa linearmente limitato: la testina non si sposta fuori dal input.

ALBA è decidibile Il numero di configurazioni è = stati * lunghezza nastro * simbolin

- Simula M per qngn passi o finchè non si ferma
- Se M è ferma, accetta se ha accettato. Rifiuta, se ha rifiutato. Se non si è fermata, rifiuta.

Questo perché altrimenti tornerebbe su una stessa configurazione, andando quindi in ciclo.

ELBA è indecidibile Tramite un LBA. Dove accetta se esiste una configurazione accettante per M, altrimenti rifuta. E ritorna l'opposto.

Cioè se R accetta, vuol dire che insieme vuoto.

Il decisore quindi per ATM farebbe l'opposto.

L'unica stringa accettante sarebbe w, quindi M accetta w. Di conseguenza S accetta (M, w) ma non è possibile.

ALLCFG è indecidibile Una CFG genera tutte le possibili stringhe.

Cap 5.3 Riducibilità mediante funzione

Funzioni calcolabili Se esiste una MdT che si ferma avendo solo f(w) sul nastro su input w.

Definizione formale di riducibilità mediante funzione Un linguaggio si dice riducibile mediante funzione, indicato con A <=m B, se esiste una funzione calcolabile da $\sum$* -> $\sum$*, dove per ogni w, w $\in$ A sse f(w) $\in$ B. f è chiamata riduzione da A a B.

Cap 6.3 Turing riducibilità

Un oracolo è un dispositivo che riferisce se w appartiene al linguaggio. Una MdT con oracolo, può interrogare un oracolo. MB oracolo per linguaggio B.

Tramite oracolo, ETM è decidibile rispetto a ATM.

Turing riducibilità: A <=T B se A è decidibile rispettto a B.