LDP-Flashcards

1. Definizioni

Grammatica:
Una grammatica generativa è una quadrupla

$$G = (X, V, S, P)$$

dove:

- X è l'alfabeto terminale per la grammatica
- V è l'alfabeto non terminale per la grammatica
- S è il simbolo di partenza per la grammatica
- P è l'insieme di produzioni della grammatica con le seguenti condizioni:$X \cap V = \oslash$ (non hanno elementi comuni tra loro) e $S \in V$ (esiste un simbolo di partenza nell'alfabeto non terminale)

Produzione:
Una produzione è una coppia di parole (v, w),dove $v \in (X \cup V)^+$ e dove $w \in (X \cup V)^*$
Un elemento (v, w) di P viene comunemente scritto nella forma

$$v \to w$$

Derivazione diretta:
Una produzione diretta avviene quando, dove data una grammatica $G = (X, V, S, P)$, abbiamo due stringhe y e z (composte da simboli terminali e non terminali con pezzi in comune) tali che:

$$y \Rightarrow z$$

Linguaggio generato da una grammatica:
Sia $G = (X, V, S, P)$ una grammatica, il **linguaggio generato da G**, denotato con $L(G)$, è l'insieme delle stringhe di terminali derivabili dal simbolo di partenza S

$$L(G) = (w \in X^* | S \Rightarrow w)$$

Albero di derivazione:
Data una grammatica G, che sia C.F. e una parola w derivabile da G con $w \in X^*$, un albero di derivazione T rappresenta graficamente le produzioni della grammatica e rispetta le seguenti proprietà:

- S è la radice
- I nodi interni sono rappresentati dai simboli NT V
- Le foglie sono rappresentati dai simboli terminali X o λ
- La stringa w è rappresentata dalla frontiera dell'albero

Grammatica Context Free:
Una grammatica $G = (X, V, S, P)$ è **libera da contesto** (o **context-free - C.F.**) se, per ogni produzione, $v \to w$, v è un non terminale.

Grammatica Context-Sensitive:
Una grammatica $G = (X, V, S, P)$ è **dipendente da contesto** se ogni produzione in P è in una delle seguenti forme:

- **Produzione contestuale**:

  $$yAz \to ywz$$

  con $A \in V$,$y, z \in (X \cup V)^*$ e $w \in (X \cup V)^+$.
- **Produzione speciale per la stringa vuota:**

  $$S \to \lambda$$

  Grammatica Lineare Destra:
  Una grammatica G viene definita lineare destra quando le produzioni sono limitate alla forma
  1. $A \to bC$ con $A, C \in V$ e $b \in X$
  2. $A \to b$ con $A \in V$ e $b \in X \cup \{\lambda\}$

Grammatica monotona:
Una grammatica $G = (X, V, S, P)$ si dice **monotona** se tutte le sue produzioni $v \to w$ soddisfano la condizione:

$$|v| \leq |w|$$

Grammatica ambigua:
Una grammatica G libera da contesto è ambigua se esiste almeno una stringa x in $L(G)$ che ha due alberi di derivazione differenti

Teorema della Gerarchia di Chomsky:
Il **Teorema della Gerarchia di Chomsky** dimostra che le quattro classi di linguaggi formali formano una gerarchia strettamente inclusiva, dove ogni classe è un sottoinsieme proprio della precedente.
Denotiamo con $\mathcal{L}_i$ (insieme dei linguaggi di tipo i) il seguente insieme:

$$\mathcal{L}_i = \{L \subset X^* | L = L(G), G \text{ di tipo i}\}$$

La gerarchia di Chomsky è una gerarchia in senso stretto di classi di linguaggi:

$$\mathcal{L}_3 \underset{\neq}{\subset} \mathcal{L}_2 \underset{\neq}{\subset} \mathcal{L}_1 \underset{\neq}{\subset} \mathcal{L}_0$$

2. Algoritmi e Procedure

Da automa a grammatica

Algoritmo: Costruzione di una grammatica lineare destra equivalente ad un automa accettore a stati finiti

- Sia dato un automa accettore a stati finiti:

$$M = (Q, \delta, q_0, F)$$

con alfabeto di ingresso X.
La grammatica lineare destra G equivalente a M, ossia tale che $L(G) = T(M)$, si costruisce come segue:

- (I) X = alfabeto di ingresso di M
- (II) $V = Q$;
- (III) $S = q_0$;
- (IV) $P = \{q \to xq' \mid q' \in \delta(q,x)\} \cup \{q \to x \mid \delta(q,x) \in F\} \cup \{q_0 \to \lambda \mid q_0 \in F\}$

Da grammatica ad automa

Algoritmo: Costruzione di un automa a stati finiti non deterministico equivalente ad una grammatica lineare destra

- Data una grammatica lineare destra:

$$G = (X, V, S, P)$$

l'automa accettore a stati finiti equivalente ($T(M) = L(G)$) viene costruito come segue:

$$M = (Q, \delta, q_0, F)$$

- (I) X come alfabeto di ingresso;
- (II) $Q = V \cup \{q\},\ q \notin V$
- (III) $q_0 = S$
- (IV) $F = \{q\} \cup \{B \mid B \to \lambda \in P\}$
- (V) $\delta : Q \times X \to 2^Q$ э′ $V.a \quad \forall B \to aC \in P,\ C \in \delta(B,a)$
  $V.b \quad \forall B \to a \in P,\ q \in \delta(B,a)$

Da Automa Non Deterministico a Deterministico:

Sia $M = (Q, \delta, q_0, F)$ un automa accettore a stati finiti non deterministico di alfabeto di ingresso, l'automa M può essere trasformato in un automa deterministico di alfabeto di ingresso X $M' = (Q', \delta', q_0', F')$ come segue:

- $Q' = 2^Q$ (tutti i sottoinsiemi di Q)
- $q_0' = \{q_0\}$

- $F' = \{p \subseteq Q \mid p \cap F \neq \emptyset\}$
- $\delta'(q, x) = \bigcup_{q \in p} \delta(q, x), \quad \forall p \in Q', x \in X$

3. Teoremi e Dimostrazioni

Proprietà degli Alberi di Derivazione:
Sia G una grammatica libera da contesto (CFG) e sia T un albero di derivazione generato da G, allora esiste una costante $k > 0$, dipendente da G, tale che per ogni albero di derivazione T di altezza h la lunghezza $|w|$ della stringa derivata (frontiera) soddisfa:

$$|w| \leq k^h$$

4. Teoria del Compilatore

Funzioni Generali:
Ogni riga della TS contiene **attributi** legati a una variabile. Gli attributi possono variare in base al linguaggio, ma generalmente includono:

1. **Nome della variabile** – può essere di lunghezza variabile, spesso gestita dallo scanner.
2. **Indirizzo** – la posizione della variabile nella memoria a run-time. Nei linguaggi senza allocazione dinamica (es. FORTRAN), questo è sequenziale; nei linguaggi a blocchi può essere rappresentato come coppia `<livello di blocco, offset>`.
3. **Tipo** – può essere implicito (FORTRAN), esplicito (PASCAL), o assente (LISP). Determina il controllo semantico e la quantità di memoria necessaria.
4. **Dimensione** – serve per array, matrici, o numero di parametri di una procedura. Ad esempio, un array avrà dimensione 1, una matrice 2.
5. **Linea di dichiarazione**.
6. **Linee di riferimento** – dove la variabile viene utilizzata nel codice.
7. **Puntatore** – usato per ordinamenti (es. ordine alfabetico) o per generare cross-reference.

Le funzioni principali sono **inserimento** e **ricerca**. Se il linguaggio richiede dichiarazioni esplicite, l'inserimento avviene durante l'elaborazione delle dichiarazioni. Se la tabella è ordinata (per esempio per nome), ogni inserimento implica una ricerca e possibile spostamento degli elementi per mantenere l'ordine. Se disordinata, l'inserimento è rapido ma la ricerca diventa costosa.

Gestione nei Linguaggi a Blocchi:
Nei linguaggi a blocchi (come Pascal o C), variabili con lo stesso nome possono esistere in blocchi annidati. Servono quindi due operazioni:

- **Set**: entra in un nuovo blocco, inizializza una nuova sotto-tabella.
- **Reset**: esce da un blocco, rimuove la relativa sotto-tabella.
  La **ricerca** inizia dalla sotto-tabella più interna, risolvendo correttamente l'ambiguità con

le regole di scope. Alla fine del blocco, le variabili locali non sono più visibili e vengono eliminate.

Modello del compilatore

Il compilatore traduce un **programma sorgente** in un **programma oggetto**.
Si articola in due fasi principali:

1. **Analisi**: trasforma il sorgente in una rappresentazione intermedia e comprende:
   - **Analisi lessicale (scanner)**: riconosce token (identificatori, parole chiave, operatori, costanti) e costruisce la tabella dei simboli.
   - **Analisi sintattica (parser):** verifica le regole grammaticali e costruisce l'albero sintattico.
   - **Analisi semantica**: controlla vincoli di contesto (tipi, dichiarazioni, compatibilità) e produce rappresentazione intermedia (IR).
2. **Sintesi**: genera e ottimizza il codice oggetto e comprende:
   - **Ottimizzazione intermedia**: riduce ridondanze (sotto-espressioni comuni o propagazioni di costanti) migliorando l'efficienza senza modificare la semantica
   - **Generazione del codice oggetto**: traduce la rappresentazione intermedia in linguaggio assembler o macchina, allocando registri e memoria.
   - **Ottimizzazione finale (opzionale)**: ottimizzazioni dipendenti/indipendenti dalla macchina.
3. **Programma oggeto**: Infine si ha il programma finale, che esegue le operazioni di:
   - **Linking:** unisce il codice oggetto con librerie e moduli esterni, risolvendo i riferimenti.
   - **Loading:** carica il programma eseguibile in memoria, trasformando gli indirizzi relativi in assoluti.