Definizioni LFT

Aggiornato al 4 settembre 2017

Indice

1 Linguaggi e Stringhe **3**
- 1.1 Linguaggio Formale . . . 3
- 1.2 Stringhe . . . 3
  - 1.2.1 Sottostringhe . . . 3
  - 1.2.2 Prefisso . . . 3
  - 1.2.3 Suffisso . . . 3
  - 1.2.4 Sottostringa Propria . . . 3
- 1.3 Operazioni su Stringhe . . . 3
  - 1.3.1 Riflessione . . . 3
  - 1.3.2 Potenza m-esima . . . 3
- 1.4 Linguaggi . . . 4
  - 1.4.1 Definizione . . . 4
  - 1.4.2 Proprietà . . . 4
- 1.5 Operazioni sui Linguaggi . . . 4
  - 1.5.1 Cardinalità . . . 4
  - 1.5.2 Unione . . . 4
  - 1.5.3 Concatenazione . . . 4
  - 1.5.4 Chiusura di Kleene: . . . 4
  - 1.5.5 Complemento . . . 4
  - 1.5.6 Differenza . . . 4
  - 1.5.7 Inversione di due linguaggi . . . 5
  - 1.5.8 Intersezione . . . 5
  - 1.5.9 Potenza n-esima di due linguaggi . . . 5
  - 1.5.10 Inclusione . . . 5
  - 1.5.11 Inclusione Propria . . . 5
- 1.6 Note . . . 5
  - 1.6.1 Ambiguità inerente . . . 5

2 Automi, Grammatiche e Parsing **6**
- 2.1 Minimizzazione di automi . . . 6
  - 2.1.1 K+1 Equivalenza di stati: . . . 6
- 2.2 Distinguibilità di due stati . . . 6
- 2.3 Espressioni e Linguaggi regolari . . . 6
  - 2.3.1 Pumping Lemma . . . 6
  - 2.3.2 Proprietà di chiusura . . . 6
  - 2.3.3 Eliminazione Ricorsione Sinistra . . . 7
  - 2.3.4 Fattorizzazione sinistra . . . 7
- 2.4 DFA . . . 7
- 2.5 NFA . . . 8
- 2.6 ϵ-NFA . . . 8
- 2.7 PDA . . . 8
  - 2.7.1 Descrizioni Istantanee . . . 9

2.7.2 Accettazione per stato finale . . . 9
2.7.3 Accettazione per pila vuota . . . 9
2.7.4 Linguaggi dei PDA . . . 9
2.8 Parsificazione LL(1) - Top-Down . . . 9
2.8.1 LL(1) . . . 9
2.8.2 First . . . 9
2.8.3 Follow . . . 10
2.8.4 Insieme Giuda . . . 10
2.9 Parsificazione LR(1) - Bottom-Up . . . 11
2.9.1 Handle . . . 11
2.9.2 Prefisso Ammissibile Valido . . . 11
2.9.3 Item ("cursore") . . . 11
2.9.4 Chiusura di I . . . 11
2.9.5 Stati del parser SLR(1) . . . 11
2.9.6 LR(0) . . . 11
2.9.7 LR(1) . . . 12
2.9.8 SLR(1) . . . 12

3 Pseudocodice dei Parser **13**
3.1 Parser LL(1) . . . 13
3.2 Parser LR(1) . . . 14

4 Sintax Directed Definitions **15**
4.1 Definizione . . . 15
4.1.1 Definizione breve . . . 15
4.2 Tipi di Attributi . . . 15
4.2.1 Simboli Terminali . . . 15
4.2.2 Simboli Non-Terminali . . . 15
4.3 Tipi (o Classi) di SDD . . . 16
4.3.1 SDD S-Attribuite . . . 16
4.3.2 SDD L-Attribuite . . . 16
4.4 Tabella riassuntiva SDD . . . 18
4.5 Codice On-The-Fly . . . 18

5 Sintax Directed Translation scheme **19**
5.1 Definizione . . . 19
5.1.1 Definizione breve . . . 19
5.2 Da SDD S-Attribuita a SDT . . . 19
5.3 Da SDD L-Attribuita a SDT . . . 19
5.4 Valutazione Top-Down di grammatiche L-Attribuite . . . 19
5.5 Schemi di traduzione . . . 20
5.5.1 Schema di traduzione codice On-The-Fly . . . 20
5.5.2 Schemi di Traduzione di espressioni booleane . . . 20
5.6 Produzioni in Bytecode . . . 21

Capitolo 1

Linguaggi e Stringhe

1.1 Linguaggio Formale

Per linguaggio formale, si intende un insieme di stringhe su un alfabeto di riferimento

1.2 Stringhe

1.2.1 Sottostringhe

la stringa y e' una sottostringa della stringa x se esistono delle stringhe u e v tali che x $=$ uyv

1.2.2 Prefisso

la stringa y e' un prefisso della stringa x se esiste una stringa v tale che x $=$ yv
N.B. un prefisso e' una sottostringa in cui u $= \epsilon$

1.2.3 Suffisso

la stringa y e' un suffisso della stringa x se esiste una stringa u tale che x $=$ uy
N.B. un suffisso e' una sottostringa in cui v $= \epsilon$

1.2.4 Sottostringa Propria

Una sottostringa (prefisso, suffisso) di una stringa e' **propria** se non coincide con ϵ o con la stringa stessa.

1.3 Operazioni su Stringhe

1.3.1 Riflessione

la riflessione di una stringa e' la stringa ottenuta scrivendo i caratteri in ordine inverso. x^R denota la riflessione della stringa x.

1.3.2 Potenza m-esima

della stringa x e' il concatenamento di x con se stessa m volte. x^m denota potenza m-esima di x.

1.4 Linguaggi

1.4.1 Definizione

Un linguaggio su un alfabeto e' un insieme di stringhe su quell'alfabeto.

1.4.2 Proprietà

Chiusura dei linguaggi liberi

La famiglia dei **linguaggi liberi dal contesto (CFL)** è **chiusa** per la **concatenazione e l'unione** ma **non per l'intersezione o la differenza**. **Però**, in ogni caso è **chiusa per l'intersezione e la differenza con linguaggi lineari**.

1.5 Operazioni sui Linguaggi

1.5.1 Cardinalità

La cardinalità' di un linguaggio e' il numero delle sue stringhe. Se L denota un linguaggio, |L| denota la sua cardinalità.

Un linguaggio e' **finito** se la sua cardinalità' e' finita: un linguaggio finito e' anche detto vocabolario.
Un linguaggio e' **infinito** se la sua cardinalità' e' infinita.

1.5.2 Unione

insieme delle stringhe che appartengono a L1 oppure a L2

1.5.3 Concatenazione

Insieme ottenuto concatenando in tutti i modi possibili le stringhe di L1 con le stringhe di L2

1.5.4 Chiusura di Kleene:

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

è l'unione di tutte le potenze di L.

1.5.5 Complemento

Il complemento di un linguaggio L su un alfabeto Σ rispetto a un alfabeto Δ (notazione $(\neg L)_\Delta$) e' la differenza fra Δ^* ed L

$$\neg L = \Delta^* - L$$

1.5.6 Differenza

Insieme delle stringhe di L1 che non appartengono a L2

1.5.7 Inversione di due linguaggi

Insieme delle stringhe riflesse (o "inverse") di L

1.5.8 Intersezione

Insieme delle stringhe che appartengono sia a L1 che a L2

1.5.9 Potenza n-esima di due linguaggi

Concatenamento di L con se stesso n volte

1.5.10 Inclusione

Il linguaggio L1 e' incluso nel linguaggio L2 (notazione L1 $\subseteq$ L2) se tutte le stringhe appartenenti a L1 appartengono anche a L2

1.5.11 Inclusione Propria

Il linguaggio L1 e' propriamente incluso nel linguaggio L2 (notazione L1 $\subset$ L2) se tutte le stringhe di L1 appartengono ad L2 ed almeno una stringa di L2 non appartiene ad L1

1.6 Note

1.6.1 Ambiguità inerente

Un linguaggio è inerentemente ambiguo se tutte le grammatiche che lo generano sono ambigue.

Capitolo 2

Automi, Grammatiche e Parsing

2.1 Minimizzazione di automi

2.1.1 K+1 Equivalenza di stati:

Due stati p,q sono k+1 equivalenti se

$$\forall a \in \Sigma$$
$$\delta(p,a) \equiv^k \delta(q,a)$$

dove $\equiv^k$ è una relazione di equivalenza.

2.2 Distinguibilità di due stati

Due stati di un DFA sono distinguibili se esiste una stringa di input che porta solo uno dei due in uno stato di accettazione.
Partendo soltanto dal fatto che le coppie formate da uno stato accettante e uno non accettante sono distinguibili, e aggiungendo come nuove coppie quelle i cui successori su un certo simbolo.

2.3 Espressioni e Linguaggi regolari

2.3.1 Pumping Lemma

Sia L un linguaggio regolare, allora $\exists n$ che dipende solo dal linguaggio, tale che

$$\forall w \in L \ , \ |w| \geq n$$

si può scrivere w come la concatenazione di 3 sottostringhe xyz tali che

$$y \neq \epsilon$$
$$|xy| \leq n$$
$$\forall k \geq 0 \ , \ xy^k z \in L$$

2.3.2 Proprietà di chiusura

Siano L e M due linguaggi regolari. Allora i seguenti linguaggi sono regolari:

1. Unione: $L \cup M$

2. Concatenazione: $L.M$
3. Chiusura: L^*
4. Complemento: $\overline{L}$
5. Differenza: $L - M$
6. Inversione: $L^R = \{w^R | w \in L\}$
7. Intersezione: $L \cap M$

2.3.3 Eliminazione Ricorsione Sinistra

$$\begin{aligned} A \to A\alpha_1|A\alpha_2|...|A\alpha_n \qquad & n \geq 1 \\ A \to \beta_1|\beta_2|...|\beta_k \qquad & k \geq 1 \\ & \alpha_i \neq \epsilon \end{aligned}$$

$$\begin{aligned} A &\to \beta_1 A'|\beta_2 A'|...|\beta_k A' \\ A' &\to \alpha_1 A'|\alpha_2 A'|...|\alpha_n A'|\epsilon \end{aligned}$$

2.3.4 Fattorizzazione sinistra

La fattorizzazione sinistra serve ad eliminare un eventuale prefisso comune a due parti destre di regole associate allo stesso simbolo non terminale.
Per ogni non terminale A si trova il massimo prefisso α comune a due o più alternative.
Si sostituiscono tutte le produzioni

$$A \to \alpha\beta_1|\alpha\beta_2|...|\alpha\beta_m$$

con

$$\begin{aligned} A &\to \alpha A' \\ A' &\to \beta_1|\beta_2|...|\beta_m \end{aligned}$$

lasciando le altre produzioni da A inalterate.

2.4 DFA

Un DFA M e' una quintupla di 5 elementi:

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

1. Q = è un insieme finito di stati,
2. $\Sigma =$ è l'alfabeto finito di input,
3. $\delta = Q \times \Sigma \to Q$
4. q_0 è lo stato iniziale ($q_0 \in Q$),
5. $F \subseteq Q$ è l'insieme di stati di accettazione.($F \subseteq Q$).

2.5 NFA

Un NFA M e' una quintupla di 5 elementi:

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

1. Q = è un insieme finito di stati,
2. $\Sigma =$ è l'alfabeto finito di input,
3. $\delta = Q \times \Sigma \to P(Q)$ (dove P è l'insieme delle parti),
4. q_0 è lo stato iniziale ($q_0 \in Q$),
5. $F \subseteq Q$ è l'insieme di stati di accettazione.($F \subseteq Q$).

2.6 ϵ-NFA

Un ϵ-NFA M e' una quintupla di 5 elementi:

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

1. Q = è un insieme finito di stati,
2. $\Sigma =$ è l'alfabeto finito di input,
3. $\delta = Q \times (\Sigma \cup \{\epsilon\}) \to P(Q)$ (dove P è l'insieme delle parti),
4. q_0 è lo stato iniziale ($q_0 \in Q$),
5. $F \subseteq Q$ è l'insieme di stati di accettazione.($F \subseteq Q$).

2.7 PDA

Un PDA e' una tupla di 7 elementi:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

1. Q = è un insieme finito di stati,
2. $\Sigma =$ è l'alfabeto finito di input,
3. $\Gamma =$ è l'alfabeto finito di pila,
4. $\delta = Q \times \Sigma^* \cup \{\epsilon\} \times \Gamma^* \to 2^{Q \times \Gamma^*}$
5. q_0 è lo stato iniziale,
6. $Z_0 \in \Gamma$ è il simbolo iniziale per la pila,
7. $F \subseteq Q$ è l'insieme di stati di accettazione.

2.7.1 Descrizioni Istantanee

Descrizione Istantanea (q, w, γ)
Sia P un PDA, allora:

$$\forall w \in \Sigma^*, \beta \in \Gamma^*$$
$$(p, \alpha) \in \delta(q, a, X) \rightarrow (q, aw, X\beta) \vdash (p, w, \alpha\beta)$$

2.7.2 Accettazione per stato finale

Sia $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA. Il linguaggio accettato da P per stato finale è

$$L(P) = \{w : (q_0, w, Z_0) \vdash^* (q, \epsilon, \alpha), q \in F\}$$

2.7.3 Accettazione per pila vuota

Sia $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA. Il linguaggio accettato da P per pila vuota è

$$L(P) = \{w : (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)\}$$

2.7.4 Linguaggi dei PDA

Gli automi pushdown sono equivalenti per le seguenti classi di linguaggi:

1. la classe dei linguaggi denotati dalle espressioni regolari;
2. la classe dei linguaggi generati dalle grammatiche lineari;

2.8 Parsificazione LL(1) - Top-Down

2.8.1 LL(1)

una grammatica è LL(1) quando:

1. Non ha ricorsioni sinistre
2. Se per ogni non terminale A e per ogni coppia di produzioni $A \rightarrow \alpha$ e $A \rightarrow \beta$ (Ovvero dallo stesso non terminale), gli insiemi guida sono disgiunti:

   $$Gui(A \rightarrow \alpha) \cap Gui(A \rightarrow \beta) = \Phi$$

   per ulteriore conferma di questa definizione, andare alla slide numero 14 "Traduzione top-down di schemi L-Attribuiti", a pagina numero 10 in basso.

2.8.2 First

Data una grammatica $G =< V, \Sigma, P, S >$, l'insieme FIRST di una stringa a di variabili e terminali, è definito formalmente come:

$$F(\alpha) = \{a | \alpha \rightarrow^* a\beta\} \cup \{\epsilon |\ Se\ \alpha \rightarrow^* \epsilon\}$$

definizione ricorsiva:

1. $F(\epsilon) = \{\epsilon\}$ $\qquad F(A) = \bigcup_{A \to \gamma_i \in P} F(\gamma_i)$
2. $F(a\beta) = \{a\}$
3. $F(A\beta) = \begin{cases} F(A) & \text{Se A } \textbf{non è annullabile} \\ (F(A) - \{\epsilon\}) \cup F(\beta) & \text{Se A } \textbf{è annullabile} \end{cases}$

2.8.3 Follow

Data una grammatica $G =< V, \Sigma, P, S >$, l'insieme FOLLOW (insieme dei seguiti) di una variabile A è l'insieme dei terminali con cui iniziano le stringhe che seguono A nelle derivazioni della grammatica G (assumendo $ in fine stringa). Formalmente:

$$F(\alpha) = \{a | S \to^* \alpha A a \beta\} \cup \{\$ |\ Se\ S \to^* aA\}$$

$$Fw(A) = \left(\bigcup_{B \to \alpha A \beta \in P} (F(\beta) - \{\epsilon\}) \right) \bigcup \left(\bigcup_{B \to \alpha A \beta \in P \text{ Tali che } \beta \text{ annullabile e } B \neq A} Fw(B) \right) \bigcup \{\$\} \text{ Se A è lo Start Simbol (o Assioma) di G}$$

2.8.4 Insieme Giuda

Data una grammatica G, l'insieme guida di una produzione della grammatica $A \to a$, $Gui(A \to a)$ è l'insieme dei terminali con cui iniziano le stringhe generate a partire dalla produzione stessa. $Gui(A \to a)$ è così definito:

$$Gui(A \to \alpha) = \begin{cases} F(\alpha) & \text{Se A } \textbf{non è annullabile} \\ (F(\alpha) - \{\epsilon\}) \cup Fw(A) & \text{Se A } \textbf{è annullabile} \end{cases}$$

2.9 Parsificazione LR(1) - Bottom-Up

2.9.1 Handle

Se $S \rightarrow_R^* \alpha A w \rightarrow_R \alpha\beta w$ allora $A \rightarrow \beta$ è un handle (dove $\rightarrow_R$ è una derivazione rightmost)

2.9.2 Prefisso Ammissibile Valido

Se $S \rightarrow_{Rm}^* \alpha A w \rightarrow_{Rm}^* \alpha\beta w$ è una derivazione rightmost, e $\beta = \beta_1\beta_2$

1. $\alpha\beta_1$è un **prefisso ammissibile**, e se
2. $\beta_2 = \epsilon$ allora esso è **completo**, e si è trovato un handle.

2.9.3 Item ("cursore")

Un item per una grammatica G è una produzione di G con un punto in qualche posizione del suo membro destro. Esso indica quanta parte di una produzione è stata presa in esame in un certo momento del processo di parsificazione.

2.9.4 Chiusura di I

Dato un insieme di item I, definiamo "chiusura di I " l'insieme costruito a partire da I aggiungendo per ogni item della forma

$$A \rightarrow \alpha \cdot B\beta$$

gli item $B \rightarrow \cdot\gamma_1$ $B \rightarrow \cdot\gamma_2$ ... $B \rightarrow \cdot\gamma_k$, se le produzioni sono
$B \rightarrow \cdot\gamma_1$ $B \rightarrow \cdot\gamma_2$ ... $B \rightarrow \cdot\gamma_k$ fino a quando l'insieme non resta inalterato

2.9.5 Stati del parser SLR(1)

1. **Shift (stato di spostamento)** che contiene solo item della forma $B \rightarrow x \cdot \gamma$
2. **Reduce (stato di riduzione)** che contiene un **unico** item della forma $A \rightarrow B$

2.9.6 LR(0)

L'informazione negli stati suggerisce le possibili mosse del parser.
In uno stato, un item del tipo:

$$A \rightarrow \beta \cdot a\gamma$$

suggerisce di fare una mossa di spostamento per portare 'a' nella pila.
mentre in un item del tipo:

$$B \rightarrow \delta\cdot$$

suggerisce di fare una riduzione di δ in B

Può accadere che per una grammatica gli item del tipo $B \rightarrow \delta\cdot$ stiano sempre da soli in uno stato (è il caso della gramm. $S \rightarrow 0S1|01$).
Le grammatiche che hanno questa proprietà sono dette LR(0).

2.9.7 LR(1)

Nota bene: il Parser effettivo degli Item, degli Handle, dei prefissi ammissibili è un **Parser LR(1)**, mentre il **metodo di parsificazione che usiamo è SRL(1)**!

2.9.8 SLR(1)

In molti casi e' possibile capire la mossa giusta da fare in

un **conflitto shift-reduce**:

$$A \to \beta \cdot a\gamma \text{ e } B \to \delta\cdot$$

è sensato ridurre solo se il simbolo di lookahead ('a') appartiene al FOLLOW(B). Se $a \notin FOLLOW(B)$ il conflitto si risolve.

un **conflitto reduce-reduce**:

$$A \to \alpha \cdot \text{ e } B \to \delta\cdot$$

è sensato ridurre la produzione $A \to \alpha\cdot$ solo se il simbolo di lookahead appartiene al FOLLOW(A). Analogamente, è sensato ridurre la produzione $B \to \delta\cdot$ solo se il simbolo di lookahead appartiene al FOLLOW(B). Se i due insiemi sono disgiunti, in conflitto si risolve.

Nota

All'esame: "Quando un linguaggio **non è SLR(1)**?" Quando ha conflitti e stati inadeguati.

Capitolo 3

Pseudocodice dei Parser

3.1 Parser LL(1)

cc :: contiene il primo simbolo dell'input, quello su cui verranno prese le decisioni;
PROSS :: è una funzione che restituisce simbolo sotto la testina di lettura e fa avanzare la testina stessa;

```
1   public static void main(){

3     cc = PROSS;
      x = top(STACK);
5
      while (not empty (STACK)) {
7
        if (x appartiene all alfabeto (Sigma) && x = cc){
9         cc = PROSS;
          pop(STACK);
11      }
        else if (M[X, cc] == X --> aplha) {
13        pop(STACK);
          push(alpha, STACK);
15        System.out.println("X --> alpha");
        }
17      else {
          ERRORE();
19      }
        X = top(STACK);
21    }

23    if (cc = '$'){
        System.out.println("Stringa accettata");
25    }
      else {
27      ERRORE();
      }
29  }

31
    // MAIN OTTIMIZZATO ----------------------------------------------------------
33
    public static void main(){
35    cc = PROSS;
      x = top(STACK);
37    while (not empty (STACK)) {
        if (x appartiene all alfabeto (Sigma) && x = cc){
39        cc = PROSS;
          pop(STACK);
```

```
41            }
              else if (M[X, cc] == X --> aplha && alpha != cc.beta) {
43              pop(STACK);
                push(alpha, STACK);
45              System.out.println("X --> alpha");
              }
47            else if (M[X, cc] == X --> aplha && alpha == cc.beta) {
                cc = PROSS;
49              pop(STACK);
                push(beta, STACK);
51              System.out.println("X --> alpha");
              }
53            else {
                ERRORE();
55            }
              X = top(STACK);
57          } // end while
            if (cc = '$'){
59            System.out.println("Stringa accettata");
            }
61          else {
              ERRORE();
63          }
        } // end main
65
```

3.2 Parser LR(1)

```
        public static void main(){
2         cc = PROSS;
          while (true) {
4           k = top(STACK);
            if (azione(k, cc) == shift(i)){
6             push(I_i, stack);    // dove I_i == stato i di indice i

8           }
            else if (azione(k, cc) = reduce(A --> beta)) {
10            pop(beta);
              k = top(STACK);
12            push(goto(k, A))
              System.out.println("A --> beta");
14          }
            else if (azione(k, cc) == "accetta") {
16            break;
            }
18          else {
              ERRORE();
20          }
          } // end while
22      } // end main
```

Capitolo 4

Sintax Directed Definitions

4.1 Definizione

Si associa l'informazione a un simbolo di una grammatica associando **"attributi"** al simbolo della grammatica che rappresenta il costrutto. In una definizione guidata dalla sintassi (Syntax-Directed Definition) i valori degli attributi sono **calcolati da "regole di valutazione"** associate alle produzioni della grammatica.

La traduzione specificata da un SDD per una certa stringa è calcolata, in linea di principio, partendo dall'albero di parsificazione della stringa, usando le regole per valutare gli attributi in ogni nodo dell'albero.

4.1.1 Definizione breve

Sono generalizzazioni delle grammatiche context-free in cui ad ogni simbolo della grammatica è associato un insieme di attributi.

4.2 Tipi di Attributi

4.2.1 Simboli Terminali

Gli attributi per i simboli terminali hanno i **valori forniti dall'analizzatore lessicale** (**.lexval**). Nelle SDD, non vi sono regole semantiche per calcolare i valori degli attributi per i terminali.

4.2.2 Simboli Non-Terminali

Per i simboli non terminali consideriamo due tipi di attributi:

1. **Sintetizzati**: un attributo sintetizzato per una variabile A in un nodo n dell'albero di parsificazione è definito da una regola semantica associata alla produzione in n e il suo valore è **calcolato** solo in termini dei valori degli **attributi nei nodi figli di n e in n stesso**. (A è il simbolo a sinistra nella produzione).

2. **Ereditati**: un attributo ereditato per una variabile A in un nodo n dell'albero di parsificazione è definito da una regola semantica associata alla produzione nel nodo padre di n e il suo valore è **calcolato** solo in termini dei valori degli **attributi del padre di n, di n stesso e dei suoi fratelli**. (A è un simbolo nel corpo della produzione, cioè al membro destro).

4.3 Tipi (o Classi) di SDD

4.3.1 SDD S-Attribuite

Una SDD è **S-Attribuita** quando tutti i suoi attributi sono **sintetizzati**.

4.3.2 SDD L-Attribuite

Una SDD è **L-Attribuita** se i suoi attributi sono

- **sintetizzati**, oppure,
- **ereditati**, e soddisfano i seguenti vincoli:
  Per ogni produzione $A \to X_1 X_2 ... X_n$ ogni attributo ereditato di X_j dipende solo
  1. dagli attributi dei **suoi fratelli a sinistra**, cioè dagli attributi ereditati o sintetizzati dei simboli $X_1 X_2 ... X_{j-1}$ a sinistra di X_j nella produzione,
  2. dagli attributi di **suo padre**, cioè dagli attributi ereditati di A
  3. dagli attributi di **sè stesso**, cioè, da attributi ereditati o sintetizzati di X_j purché non vi siano cicli nel grafo delle dipendenze formati dagli attributi di questa occorrenza di X_j.

Pseudocodice d'esempio di seguito

C → N#	{L.*elem* = N.*val*}
L	{C.*list* = L.*list*}
L → N;	{L_1.*elem* = L.*elem*}
L_1	{*L.list* = *cons* (N.*val* - L.*elem*, L_1.*list*)}
L → ε	{L.*list* = <>|}
N → **digit**	{N.*val* = **digit**.*val*}

La grammatica è LL(1):

	Insiemi guida
C → N#L	{**digit**}
L → N;L_1	{**digit**}
L → ε	{$}
N → **digit**	{**digit**}

```
// Valutatore Top-Down di SDD L-Attribuite

int main(){
  cc = PROSS;
  list = C();
  if (cc = '$') {
    printf("stringa accettata");
  }
  else { ERRORE(); }
}

var N(){
  var N_val
  if (cc = digit) {
    N_val = digit.val;
  }
  cc = PROSS;
  else { ERRORE(); }
  return N_val
}

var C(){
  var C_list, N_val, L_elem, L_list;
  if (cc = digit) {
    N_val = N();
    if (cc = '#') {
      cc = PROSS;
      L_elem = N_val;
      L_list = L (L_elem);
      C_list = L_list;
    }
    else { ERRORE(); }
  }
  else { ERRORE(); }
  return C_list
}

var L(L_elem){
  var L_list, N_val, L1_elem, L1_list;
  if (cc = digit){
    N_val = N()
    if (cc = ';') {
      cc = PROSS;
      L1_elem = L_elem;
      L1_list = L (L1_elem);
      L_list = cons(N_val - L_elem, L1_list);
    }
    else { ERRORE(); }
  }
  else if (cc = '$'){
    L_list = <>;
  }
  else { ERRORE(); }
  return L_list;
}

```

4.4 Tabella riassuntiva SDD

$P \rightarrow S$	$S.next = newlabel\ ()$ $P.code = S.code \parallel \text{label}(S.next) \parallel emit(\text{Stop})$
$S \rightarrow \textbf{id} = E;$	$S.code = E.code \parallel istore(\text{addr}(\textbf{id}.lex))$
$S \rightarrow \textbf{if}\ (B)\ S_1$	$B.true = \text{newlabel}\ ()\quad B.false = S_1.next = S.next$ $S.code = B.code \parallel \text{label}(B.true) \parallel S_1.code$
$S \rightarrow \textbf{if}\ (B)\ S_1\ \textbf{else}\ S_2$	$B.true = \text{newlabel}\ ()\quad B.false = \text{newlabel}\ ()$ $S_1.next = S_2.next = S.next$ $S.code = B.code \parallel \text{label}(B.true) \parallel S_1.code \parallel$ $\parallel$ 'goto' $S.next) \parallel \text{label}(B.false) \parallel S_2.code$
$S \rightarrow \textbf{while}\ (B)\ S_1$	$B.true = newlabel\ ()\quad B.false = S.next$ $S_1.next = newlabel()$ $S.code = label(S_1.next) \parallel B.code \parallel \text{label}(B.true) \parallel$ $\parallel S_1.code \parallel$ 'goto $S_1.next$'
$S \rightarrow \{SL\}$	$SL.next = S.next\quad S.code = SL.code$
$SL \rightarrow SL_1\ ;\ S$	$SL_1.next = newlabel()$ $S.next = SL.next$ $SL.code = SL_1.code \parallel label(SL_1.next) \parallel S.code$
$SL \rightarrow S$	$S.next = SL.next$ $SL.code = S.code$

4.5 Codice On-The-Fly

Un caso di particolare interesse si verifica quando in una traduzione un file di output (tipicamente il codice in un compilatore/traduttore) viene generato seguendo l'ordine di lettura in profondità, da sinistra a destra, dei nodi dell'albero sintattico.
In questo caso si puo' sostituire la valutazione di attributi tipo code inserendo negli schemi dei comandi per la **generazione diretta del codice nei punti opportuni**.

$E \rightarrow T\ E'$
$E' \rightarrow +\ T\ \{emit('+')\}\ E'$
$E' \rightarrow \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ \{emit('*')\}\ T'$
$T' \rightarrow \varepsilon$

$F \rightarrow \textbf{num}\ \{emit(\textbf{num}.\text{lexval})\}$
$F \rightarrow (E)$

(a) Grammatica

```
function E()
    begin
        if cc = '(' or cc = $ then T()
          E'()
          return
    end
function E'()
    begin  if (cc = '+') then
             cc ← PROSS
             T()
             emit('+')
             E'()
             return
           else if (cc = ')' or cc=$ then
             return
           else ERRORE (...)
    end
```

(b) Pseudocodice (**del traduttore**)

Figura 4.1: Esempio

Capitolo 5

Sintax Directed Translation scheme

5.1 Definizione

Gli schemi di traduzione (SDT) sono un'utile notazione per specificare la traduzione durante la parsificazione.
Uno schema di traduzione è una definizione guidata dalla sintassi in cui le azioni semantiche, racchiuse tra parentesi graffe, sono inserite nel corpo delle produzioni, in posizione tale che, durante il processo di parsificazione, il valore di un attributo sia disponibile quando un'azione fa ad esso riferimento.

5.1.1 Definizione breve

E' una definizione guidata dalla sintassi in cui le azioni semantiche, racchiuse tra graffe, sono inserite nei membri destri delle produzioni, in posizione tale che il valore di un attributo sia disponibile quando un'azione fa ad esso riferimento.

5.2 Da SDD S-Attribuita a SDT

Per trasformare una SDD S-Attribuita in un SDT bisogna

1. Inserire le azioni che calcolano gli attributi sintetizzati della variabile di testa alla fine delle produzioni.

5.3 Da SDD L-Attribuita a SDT

Per trasformare una SDD L-Attribuita in un SDT bisogna

1. Inserire le azioni che calcolano gli attributi ereditati per un non terminale A immediatamente prima dell'occorrenza di A nel corpo della produzione.

2. Se diversi attributi ereditati per A dipendono uno dall'altro, ordinare la valutazione degli attributi in modo che quelli necessari prima siano calcolati per primi.

3. Porre le azioni che calcolano un attributo sintetizzato per la variabile di testa di una produzione alla fine del corpo della produzione stessa.

5.4 Valutazione Top-Down di grammatiche L-Attribuite

Ad ogni non terminale si associa una funzione che ha come parametri in input i valori degli attributi ereditati dalla variabile e restituisce i valori dei suoi attributi sintetizzati.

La funzione per un non terminale ha una variabile locale per ogni attributo ereditato o sintetizzato per i simboli che compaiono nelle parti destre delle produzioni per quel non terminale.

5.5 Schemi di traduzione

5.5.1 Schema di traduzione codice On-The-Fly

$P \rightarrow \{SL.next = newlabel(\,)\}\ SL\ \{emitlabel(S.next), emit(\text{'stop'})\}$

$S \rightarrow \mathbf{id} = E\ \ \{emit(\text{'istore'}\ (\mathbf{id}.addr))\}$

$S \rightarrow \mathbf{if}\ \{B.true = newlabel(\,),\ B.false = S.next\}\ (B)$
$\{emitlabel(B.true),\ S_1.next = S.next\ \}\ S_1\ \{gen(\text{'goto'}\ B.false)\}$

$S \rightarrow \mathbf{if}\ \{B.true = newlabel(\,),\ B.false = \text{newlabel}(\,)\}\ (B)$
$\{emitlabel(B.true),\ S_1.next = S.next\ \}\ S_1\ \mathbf{else}$
$\{emitlabel(B.false)),\ S_2.next = S.next\ \}\ S_2$

$S \rightarrow \mathbf{while}\ \{B.true = newlabel(\,),\ B.false = S.next,\ S_1.next = newlabel(),$
$emitlabel(S_1.next\,)\}\ \ (B)\ \{emitlabel(B.true)\ \}$
$S_1\ \{emit(\text{'goto'}\ S_1.next\,)\}$

$B \rightarrow E_1 = E_2 \qquad \{emit(\text{'if\_cmpeq'}\ B.true);\ emit(\text{'goto'}\ B.false)\}$

$S \rightarrow \{\ \{SL.next = S.next\ \}\ SL\}$

$SL \rightarrow \{SL_1.next = newlabel(\,)\}\ SL_1\ ;$
$\{emitlabel(SL_1.next\,)\}\ \{S.next = SL.next\ \}\ S$

$SL \rightarrow\ \{S.next = SL.next\ \}\ \ S$

Nota: esempio tratto dalla classe **traduttore**.

5.5.2 Schemi di Traduzione di espressioni booleane

$B \rightarrow\ \ \{B_1.true = B.true\ ;\ B_1.false = newlabel\ ()\ \}\ B_1\ ||$
$\{B_2.true = B.true\ ;\ B_2.false = B.false\ \}\ \ B_2$
$\{\ B.code = B_1.code\ ||\ label(B_1.false)\ ||\ B_2.code\}$

$B \rightarrow\ \ \{B_1.true = newlabel\ ()\ ;\ B_1.false = B.false\ \}\ \ B_1\ \&\&$
$\{B_2.true = B.true\ ;\ B_2.false = B.false\ \}\ \ B_2$
$\{\ B.code = B_1.code\ ||\ label(B_1.true)\ ||\ B_2.code\ B_2\ \}$

$B \rightarrow\ \ !\ \{B_1.true = B.false;\ B_1.false = B.true;\ \}\ B_1\ \{B.code = B_1.code\}$

$B \rightarrow\ \ (\ \{B_1.true = B.true;\ B_1.false = B.false\ \}\ \ B_1\ )\ \{B.code = B_1.code\}$

$B \rightarrow E_1 == \ E_2\ \{\ B.code = E_1.code\ ||\ E_2.code\ ||$
$\text{'if\_cmpeq'}\ B.true\ ||\ \text{'goto'}\ B.false\ \}$

$B \rightarrow \mathbf{true}\ \ \{B.code = \text{'}goto\text{'}\ B.true\}$

$B \rightarrow \mathbf{false}\ \{B.code = \text{'goto'}\ B.false\}$

Nota: esempio tratto dalla classe **traduttore**.

5.6 Produzioni in Bytecode

```
1
        S --> if(B) S1
3       B.true = newlabel()
        B.false = S1.next = S.next
5       S.code = B.code || label(B.true) || S1.code

7       S --> if(B) S1 else S2
        B.true = newlabel()
9       B.false = newlabel()
        S1.next = S2.next = S.next
11      S.code = B.code || label(B.true) || S1.code || 'goto' S.next ||
        || label(B.false) || S2.code
13
        S --> while(B) S1
15      B.true = newlabel()
        B.false = S.next
17      S1.next = newlabel()
        S.code = label(S1.next) || B.code || label(B.true) || S1.code ||
19      || 'goto' S1.next
```