

# guaggi Formali e Traduttori

## 2.2 Schemi di traduzione

- Sommario
- Schemi di traduzione
- Differenze tra regole e azioni semantiche
- Conversione da SDD L-attribuite a SDT
- Esempio
- SDT e parsing ricorsivo discendente
- Esempio: stringhe della forma  $a^n b^n$
- Esempio: espressioni in forma infissa
- Esempio: da forma prefissa a forma infissa
- Traduzione “on-the-fly”
- Esempio: da forma infissa a forma postfissa
- Esercizi

È proibito condividere e divulgare in qualsiasi forma i materiali didattici caricati sulla piattaforma e le lezioni svolte in videoconferenza: ogni azione che viola questa norma sarà denunciata agli organi di Ateneo e perseguita a termini di legge.

# Sommario

## Problema

- Le SDD forniscono una specifica ad alto livello del processo di traduzione in cui l'ordine di valutazione degli attributi è implicito.
- Una SDD richiede l'albero sintattico per la memorizzazione degli attributi.

## In questa lezione


1. Introduciamo gli **schemi di traduzione** (SDT), una variante delle SDD in cui si rende esplicito l'ordine di valutazione degli attributi.
2. Vediamo come convertire una SDD L-attribuita in uno SDT.
3. Vediamo come integrare uno SDT in un parser ricorsivo discendente sfruttando la pila del linguaggio di programmazione per la memorizzazione degli attributi ed evitando la costruzione esplicita dell'albero sintattico.

# Schemi di traduzione

## Definizione

Uno **schema di traduzione** (o **SDT**, da **Syntax-Directed Translation scheme**) è una grammatica in cui le produzioni sono arricchite da frammenti di codice detti **azioni semantiche** che sono eseguite nel momento in cui tutti i simboli alla loro sinistra sono stati riconosciuti.

## Esempi



Produzione	Produzione + Azioni	Descrizione
$A \rightarrow BC$	$A \rightarrow BC\{code\}$	<i>code</i> eseguito dopo il riconoscimento di <b>B</b> e <b>C</b>
$A \rightarrow BC$	$A \rightarrow B\{code\}C$	<i>code</i> eseguito dopo il riconoscimento di <b>B</b> ma prima del riconoscimento di <b>C</b>
$A \rightarrow BC$	$A \rightarrow \{code_1\}BC\{code_2\}$	<i>code</i> <sub>1</sub> eseguito subito dopo la riscrittura di <b>A</b> e prima del riconoscimento di <b>B</b> , <i>code</i> <sub>2</sub> eseguito dopo il riconoscimento di <b>B</b> e <b>C</b>
$A \rightarrow \varepsilon$	$A \rightarrow \{code\}$	<i>code</i> eseguito subito dopo la riscrittura di <b>A</b>

# Differenze tra regole e azioni semantiche



## Regole semantiche (SDD)

- Specificano come determinare il valore degli attributi.
- Sono valutate in un ordine implicito determinato dal grafo delle dipendenze.
- Poiché valutate in un ordine arbitrario, in generale richiedono la costruzione dell'albero sintattico annotato.

## Azioni semantiche (SDT)

- Solitamente specificano come determinare il valore degli attributi, ma possono contenere codice arbitrario (es. stampe, invocazione di metodi, ecc.).
- Sono eseguite in un ordine esplicito determinato dalla loro posizione nel corpo delle produzioni.
- Poiché eseguite da sinistra verso destra, possono essere integrate al parsing ricorsivo discendente senza richiedere la costruzione dell'albero sintattico annotato.

# Conversione da SDD L-attribuite a SDT

## Algoritmo

Data una SDD L-attribuita, si può ottenere uno SDT corrispondente nel modo seguente. Per ogni produzione  $A \rightarrow X_1 X_2 \dots X_n$  della grammatica:

1. Subito prima di  $X_i$ , aggiungere un'azione semantica che calcola il valore degli attributi ereditati di  $X_i$ . **Nota:** in una SDD L-attribuita, questi attributi possono dipendere solo da attributi ereditati di  $A$  ed attributi di  $X_1, \dots, X_{i-1}$ .
2. In fondo alla produzione, aggiungere un'azione semantica che calcola il valore degli attributi sintetizzati di  $A$ .

All'interno di ogni azione semantica ordinare il codice rispettando le dipendenze tra diversi attributi. **Nota:** in una SDD L-attribuita, tali dipendenze sono necessariamente acicliche e dunque esiste un ordinamento che le rispetta.

# Esempio

## SDD

Produzioni	Regole semantiche
$T \rightarrow FT'$	$T'.e = F.s$ $T.s = T'.s$
$T' \rightarrow *FT'_1$	$T'_1.e = T'.e \times F.s$ $T'.s = T'_1.s$
$T' \rightarrow \epsilon$	$T'.s = T'.e$
$F \rightarrow n$	$F.s = n.v$

## SDT

Produzioni	Azioni semantiche
$T \rightarrow F$	$\{T'.e = F.s\}$ ered
$T'$	$\{T.s = T'.s\}$ n4tet/
$T' \rightarrow *F$	$\{T'_1.e = T'.e \times F.s\}$
$T'_1$	$\{T'.s = T'_1.s\}$
$T' \rightarrow \epsilon$	$\{T'.s = T'.e\}$
$F \rightarrow n$	$\{F.s = n.v\}$

# SDT e parsing ricorsivo discendente

- Il parser ha una procedura per ogni variabile della grammatica che riconosce le stringhe generate da **A** nella grammatica.
- La procedura **A** usa il simbolo corrente e gli insiemi guida, per scegliere la produzione  $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$  da usare per riscrivere **A**.
- Per ogni simbolo **X** nel corpo della produzione scelta:
  - Se **X** è un simbolo terminale, il metodo controlla che il simbolo corrente sia proprio **X**. In tal caso, fa avanzare il lexer al simbolo successivo. In caso contrario, il metodo segnala un errore di sintassi.
  - Se **X** è una variabile, il metodo invoca la procedura **X**

# SDT e parsing ricorsivo discendente

- Il parser ha una procedura per ogni variabile della grammatica che riconosce le stringhe generate da  $A$  nella grammatica.
- La procedura  $A$  ha tanti argomenti quanti sono gli attributi ereditati di  $A$  e restituisce tanti valori quanti sono gli attributi sintetizzati di  $A$ .
- La procedura  $A$  usa il simbolo corrente e gli insiemi guida, per scegliere la produzione  $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$  da usare per riscrivere  $A$ .
- Per ogni simbolo o azione semantica  $X$  nel corpo della produzione scelta:
  - Se  $X$  è un simbolo terminale, il metodo controlla che il simbolo corrente sia proprio  $X$ . In tal caso, fa avanzare il lexer al simbolo successivo. In caso contrario, il metodo segnala un errore di sintassi.
  - Se  $X$  è una variabile, il metodo invoca la procedura  $X$  passando a  $X$  come argomenti i suoi attributi ereditati e raccogliendo in variabili locali gli attributi sintetizzati restituiti da  $X$ .
  - Se  $X$  è una azione semantica, il metodo la esegue.



# Esempio: stringhe della forma $a^n b^n$

Schema di traduzione

Produzioni	Azioni semantiche
$S \rightarrow \varepsilon$	$\{S.n = 0\}$
$S \rightarrow aS_1b$	$\{S.n = S_1.n + 1\}$

Esempi

- $\varepsilon \implies 0$
- $ab \implies 1$
- $aabb \implies 2$
- $aaabbb \implies 3$
- ...
- $a^n b^n \implies n$

# Esempio: stringhe della forma $a^n b^n$

Schema di traduzione

Produzioni	Azioni semantiche
$S \rightarrow \varepsilon$	$\{S.n = 0\}$
$S \rightarrow aS_1b$	$\{S.n = S_1.n + 1\}$

Esempi

- $\varepsilon \Rightarrow 0$
- $ab \Rightarrow 1$
- $aabb \Rightarrow 2$
- $aaabbb \Rightarrow 3$
- ...
- $a^n b^n \Rightarrow n$

Codice Java

```
private int S() {  
    switch (peek()) {  
  
        case 'a': // S → aSb  
        { check('a');  
          int S_n = S();  
          check('b');  
          return S_n + 1; }  
  
        case 'b': // S → ε  
        case '$':  
          return 0;  
  
        default:  
          throw error("S");  
    }  
}
```

# Esempio: espressioni in forma infissa

Produzioni	Azioni semantiche
$T \rightarrow F$	$\{T'.e = F.s\}$
$T'$	$\{T.s = T'.s\}$
$T' \rightarrow *F$	$\{T'_1.e = T'.e \times F.s\}$
$T'_1$	$\{T'.s = T'_1.s\}$
$T' \rightarrow \varepsilon$	$\{T'.s = T'.e\}$
$F \rightarrow \mathbf{n}$	$\{F.s = \mathbf{n}.v\}$

# Esempio: espressioni in forma infissa

Produzioni	Azioni semantiche
$T \rightarrow F$	$\{T'.e = F.s\}$
$T'$	$\{T.s = T'.s\}$
$T' \rightarrow *F$	$\{T'_1.e = T'.e \times F.s\}$
$T'_1$	$\{T'.s = T'_1.s\}$
$T' \rightarrow \varepsilon$	$\{T'.s = T'.e\}$
$F \rightarrow n$	$\{F.s = n.v\}$

```
private int T() {
  switch (peek()) {
  case 'n': { // T → FT'
    int F_s = F();
    int TT_s = TT(F_s);
    return TT_s;
  }
  default:
    throw error("T");
  }
}
```

```
private int TT(int TT_e) {
  switch (peek()) {
  case '*': // T' → *FT'
    { check('*');
      int F_s = F();
      int TT_s = TT(TT_e * F_s);
      return TT_s; }
  case '+':
  case '$': // T' → ε
    return TT_e;
  default:
    throw error("T");
  } }
}
```

```
private int F() {
  switch (peek()) {
  case 'n': // F → n
    { int F_s = peek() - '0';
      check('n');
      return F_s; }
  default:
    throw error("F");
  } }
}
```

# Esempio: da forma prefissa a forma infissa

SDT

Produzioni	Azioni semantiche
$S \rightarrow$	$\{E.c = \emptyset\}$
$E$	$\{S.i = E.i\}$
$E \rightarrow +$	$\{E_1.c = \emptyset\}$
$E_1$	$\{E_2.c = \{+\}\}$
$E_2$	$\{E.i = \text{PAR}(+, E_1.i \parallel "+" \parallel E_2.i)\}$
$E \rightarrow *$	$\{E_1.c = \{+\}\}$
$E_1$	$\{E_2.c = \{+, *\}\}$
$E_2$	$\{E.i = \text{PAR}(*, E_1.i \parallel "*" \parallel E_2.i)\}$
$E \rightarrow \mathbf{n}$	$\{E.i = \mathbf{n}.v\}$

$$\text{PAR}(o, s) \stackrel{\text{def}}{=} \begin{cases} "(" \parallel s \parallel ")" & \text{se } o \in E.c \\ s & \text{altrimenti} \end{cases}$$

# Esempio: da forma prefissa a forma infissa

## SDT

Produzioni	Azioni semantiche
$S \rightarrow$	$\{E.c = \emptyset\}$
$E$	$\{S.i = E.i\}$
$E \rightarrow +$	$\{E_1.c = \emptyset\}$
$E_1$	$\{E_2.c = \{+\}\}$
$E_2$	$\{E.i = \text{PAR}(+, E_1.i \parallel "+" \parallel E_2.i)\}$
$E \rightarrow *$	$\{E_1.c = \{+\}\}$
$E_1$	$\{E_2.c = \{+, *\}\}$
$E_2$	$\{E.i = \text{PAR}(*, E_1.i \parallel "*" \parallel E_2.i)\}$
$E \rightarrow n$	$\{E.i = n.v\}$

$$\text{PAR}(o, s) \stackrel{\text{def}}{=} \begin{cases} "(" \parallel s \parallel ")" & \text{se } o \in E.c \\ s & \text{altrimenti} \end{cases}$$

```
String E(String E_c) {
    switch (peek()) {
        case '+': // E → +E1E2
        { check('+');
          String E1_i = E("");
          String E2_i = E("+");
          return par(E_c.indexOf('+'),
                    E1_i+" "+E2_i); }
        case '*': // E → *E1E2
        { check('*');
          String E1_i = E("+");
          String E2_i = E(" * ");
          return par(E_c.indexOf('*'),
                    E1_i+" * "+E2_i); }
        case 'n': // E → n
        { char d = peek();
          check(d);
          return String.valueOf(d); }
        default:
        { throw error("E"); }
    }
}
```

# Traduzione “on-the-fly”

## Definizione

Un attributo sintetizzato si dice **principale** se:

1. Il suo valore **include sempre** la **concatenazione** dei valori dello stesso attributo per **tutte** le variabili nel corpo di ogni produzione, **oltre ad eventuali elementi ausiliari**.
2. La concatenazione **rispetta sempre l'ordine delle variabili** nel corpo delle produzioni.

## Esempio di attributo principale

$$E \rightarrow E_1 + T \{ E.post = E_1.post \parallel T.post \parallel "+" \}$$

## Osservazione

Gli attributi principali possono essere eliminati “emettendo al volo” solo gli elementi ausiliari nel punto in cui devono comparire.

## Nota

Il tipo degli attributi principali deve supportare l'operazione di concatenazione. Esempi tipici sono stringhe, liste, sequenze di istruzioni, file, ecc.

# Esempio: da forma infissa a forma postfissa

SDT con accumulo della traduzione

Produzioni	Azioni semantiche
$S \rightarrow E$	$\{\text{print}(E.post)\}$
$E \rightarrow E_1 + T$	$\{E.post = E_1.post \parallel T.post \parallel "+"\}$
$E \rightarrow T$	$\{E.post = T.post\}$
$T \rightarrow T_1 * F$	$\{T.post = T_1.post \parallel F.post \parallel "*"\}$
$T \rightarrow F$	$\{T.post = F.post\}$
$F \rightarrow (E)$	$\{F.post = E.post\}$
$F \rightarrow n$	$\{F.post = n.v\}$

- $E.post = E$  in forma postfissa
- $E.post$  è un attributo principale



# Esempio: da forma infissa a forma postfissa

## SDT con accumulo della traduzione

Produzioni	Azioni semantiche
$S \rightarrow E$	$\{\text{print}(E.post)\}$
$E \rightarrow E_1 + T$	$\{E.post = E_1.post \parallel T.post \parallel "+"\}$
$E \rightarrow T$	$\{E.post = T.post\}$
$T \rightarrow T_1 * F$	$\{T.post = T_1.post \parallel F.post \parallel "*"\}$
$T \rightarrow F$	$\{T.post = F.post\}$
$F \rightarrow (E)$	$\{F.post = E.post\}$
$F \rightarrow n$	$\{F.post = n.v\}$

- $E.post = E$  in forma postfissa
- $E.post$  è un attributo principale

## Traduzione “on-the-fly”

Produzioni	Azioni semantiche
$S \rightarrow E$	
$E \rightarrow E_1 + T$	$\{\text{print}("+")\}$
$E \rightarrow T$	
$T \rightarrow T_1 * F$	$\{\text{print}("*")\}$
$T \rightarrow F$	
$F \rightarrow (E)$	
$F \rightarrow n$	$\{\text{print}(n.v)\}$

- nessun attributo!

# Esercizi

1. Definire uno SDT corrispondente alla **SDD delle parentesi quadre bilanciate** e implementarne il parser ricorsivo discendente.
2. Definire uno SDT corrispondente alla **SDD della lista delle differenze** e implementarne il parser ricorsivo discendente.
3. Definire SDT corrispondenti alle **SDD ottenute risolvendo gli esercizi 1 e 2 della sezione precedente** e implementarne il parser ricorsivo discendente.
4. Definire uno SDT per la traduzione “on-the-fly” di espressioni aritmetiche dalla **forma infissa a quella postfissa** per la grammatica LL(1) in cui è stata eliminata la ricorsione sinistra. Implementarne poi il parser ricorsivo discendente.