

Linguaggi Formali e Traduttori

5.1 Definizioni dirette dalla sintassi

- Sommario
- Alberi sintattici annotati
- Definizioni dirette dalla sintassi
- Esempio: espressioni in forma infissa
- Esempio: stringhe della forma $a^n b^n$
- Esempio: parentesi quadre bilanciate
- Esempio: da forma infissa a forma prefissa
- Attributi sintetizzati
- Attributi ereditati
- Esempio: espressioni senza ricorsione sinistra
- Esempio: lista delle differenze
- Esempio: da forma prefissa a forma infissa
- Ordine di valutazione degli attributi
- Definizioni S-attribuite ed L-attribuite
- 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

- Tradurre un programma da un linguaggio (sorgente) a un altro (oggetto).
- Il parser risponde solo **sì/no** alla domanda “il programma è sintatticamente corretto?”

In questa lezione

Introduciamo le **definizioni dirette dalla sintassi** (SDD), che consistono in:

- Una grammatica libera, che specifica la **sintassi** dei programmi da tradurre
- Un insieme di **attributi** associati alle variabili della grammatica e che contengono il risultato della traduzione (o comunque informazioni accessorie alla traduzione)
- Un insieme di **regole semantiche** che specificano come calcolare il valore degli attributi, e quindi **come** tradurre il programma

Alberi sintattici annotati

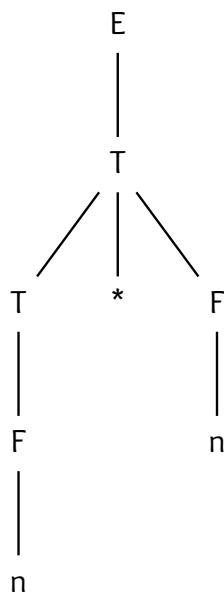
Grammatica

- $E \rightarrow E + T$
- $E \rightarrow T$
- $T \rightarrow T * F$
- $T \rightarrow F$
- $F \rightarrow (E)$
- $F \rightarrow n$

Stringa

3 * 5

Albero sintattico



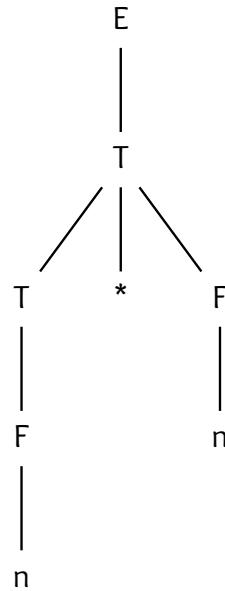
Alberi sintattici annotati

Grammatica

- $E \rightarrow E + T$
- $E \rightarrow T$
- $T \rightarrow T * F$
- $T \rightarrow F$
- $F \rightarrow (E)$
- $F \rightarrow n$

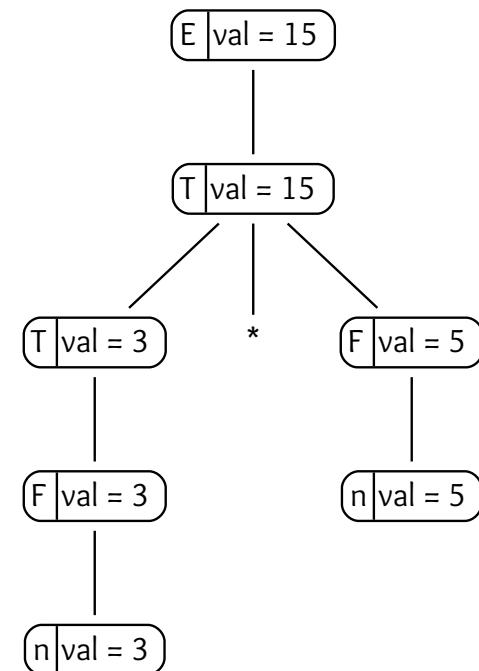
Stringa

3 * 5



Albero sintattico

Albero sintattico annotato



Definizioni dirette dalla sintassi

Definizione

Un **attributo** è una coppia (**nome, valore**) che rappresenta una qualunque informazione associata ad un nodo di un albero sintattico.

Definizione

Un **albero sintattico annotato** è un albero sintattico in cui ogni nodo può essere **annotato** con zero o più attributi.

Definizione

Una **definizione diretta dalla sintassi** (o SDD, da **Syntax-Directed Definition**) è una grammatica le cui produzioni sono associate a zero o più **regole semantiche** che specificano come calcolare il valore degli attributi associati ai nodi degli alberi sintattici della grammatica.

Il valore di eventuali attributi associati ai simboli terminali è fornito dal lexer.

Esempio

Produzioni	Regole semantiche
$E \rightarrow E_1 + T$	$E.v = E_1.v + T.v$
$E \rightarrow T$	$E.v = T.v$
$T \rightarrow T_1 * F$	$T.v = T_1.v \times F.v$
$T \rightarrow F$	$T.v = F.v$
$F \rightarrow (E)$	$F.v = E.v$
$F \rightarrow n$	$F.v = n.v$

Esempio: espressioni in forma infissa

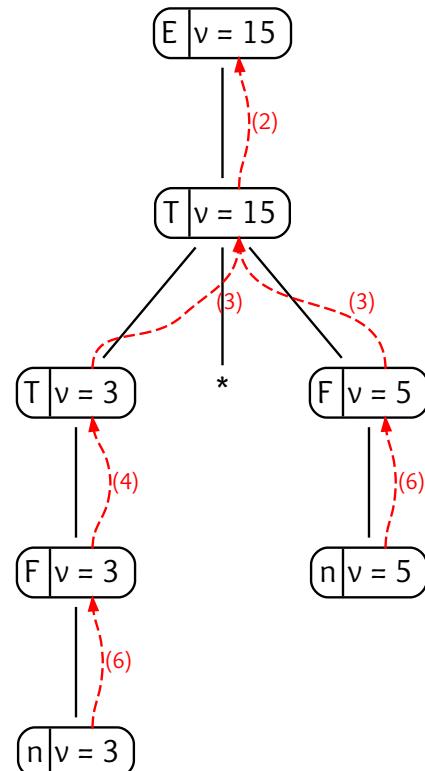
Esempio

- $3 * 5 \Rightarrow 15$

SDD

Produzioni	Regole semantiche
$E \rightarrow E_1 + T$	(1) $E.v = E_1.v + T.v$
$E \rightarrow T$	(2) $E.v = T.v$
$T \rightarrow T_1 * F$	(3) $T.v = T_1.v \times F.v$
$T \rightarrow F$	(4) $T.v = F.v$
$F \rightarrow (E)$	(5) $F.v = E.v$
$F \rightarrow n$	(6) $F.v = n.v$

- $n.v =$ valore del numero (dal lexer)
- $E.v / T.v / F.v =$ valore di $E / T / F$



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

Obiettivo

Albero annotato per aaabbb

$$S \rightarrow \epsilon \mid aSb$$

Tradurre una stringa della forma $a^n b^n$ nel numero n .

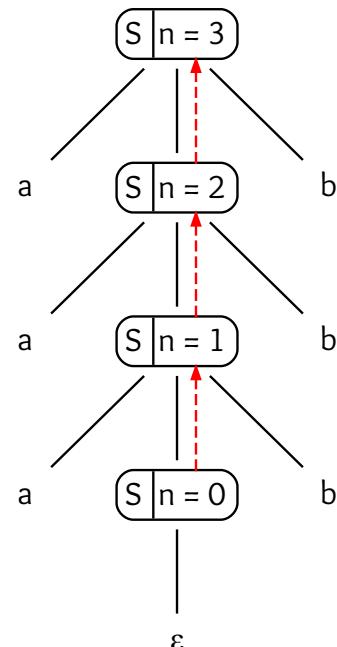
SDD

Produzioni	Regole semantiche
$S \rightarrow \epsilon$	$S.n = 0$
$S \rightarrow aS_1b$	$S.n = S_1.n + 1$

- $S.n$ = numero di a e b nella stringa generata da S

Esempi

- $\epsilon \Rightarrow 0$
- $ab \Rightarrow 1$
- $aabb \Rightarrow 2$
- $aaabbb \Rightarrow 3$



Esempio: parentesi quadre bilanciate

Obiettivo

Albero annotato per $[[[]]]$

$$S \rightarrow \epsilon \mid [S]S$$

Tradurre una stringa di parentesi quadre bilanciate nel massimo numero di parentesi annidate.

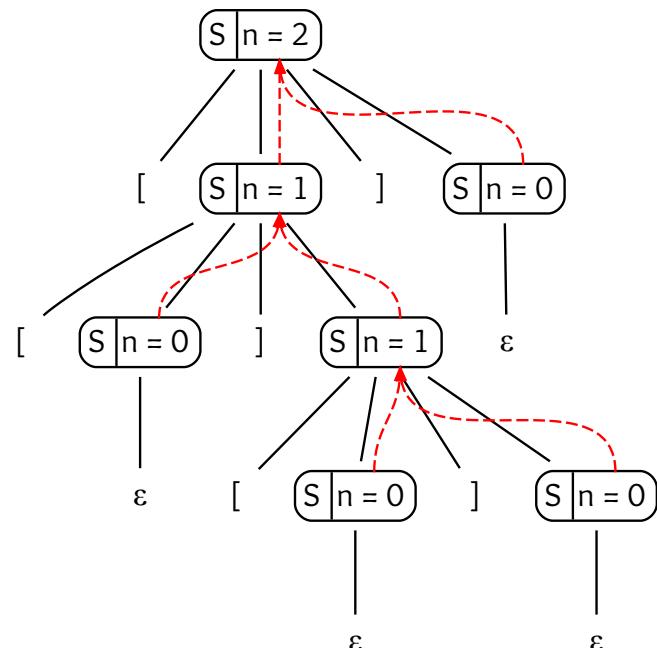
SDD

Produzioni	Regole semantiche
$S \rightarrow \epsilon$	$S.n = 0$
$S \rightarrow [S_1]S_2$	$S.n = \max\{S_1.n + 1, S_2.n\}$

- $S.n$ = massimo numero di parentesi annidate nella stringa generata da S

Esempio

- $[[[]]] \Rightarrow 2$



Esempio: da forma infissa a forma prefissa

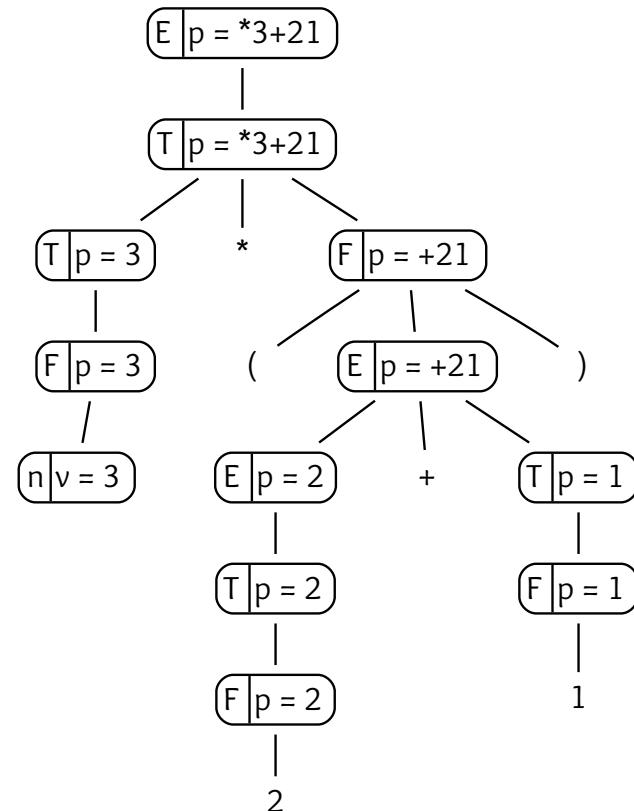
Esempio

$$3 * (2 + 1) \implies *3+21$$

SDD

Produzioni	Regole semantiche
$E \rightarrow E_1 + T$	$E.p = "+" \parallel E_1.p \parallel T.p$
$E \rightarrow T$	$E.p = T.p$
$T \rightarrow T_1 * F$	$T.p = "*" \parallel T_1.p \parallel F.p$
$T \rightarrow F$	$T.p = F.p$
$F \rightarrow (E)$	$F.p = E.p$
$F \rightarrow n$	$F.p = n.v$

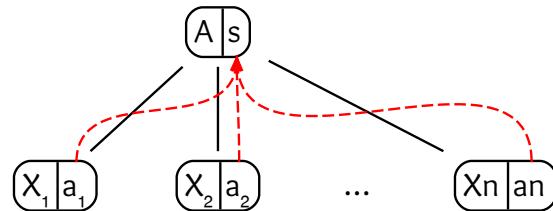
- $n.v$ = cifra (dal lexer)
- $E.p / T.p / F.p$ = forma prefissa di $E / T / F$
- \parallel = concatenazione tra stringhe



Attributi sintetizzati

Definizione

Un attributo di un nodo N in un albero annotato si dice **sintetizzato** se il suo valore dipende da quello di attributi dei figli di N ed eventualmente da altri attributi di N stesso.



Il valore di un attributo sintetizzato per un nodo etichettato con la variabile A è determinato da una regola semantica associata a una produzione per A :

$$A \rightarrow X_1 X_2 \cdots X_n \quad A.s = f(X_1.a_1, X_2.a_2, \dots, X_n.a_n)$$

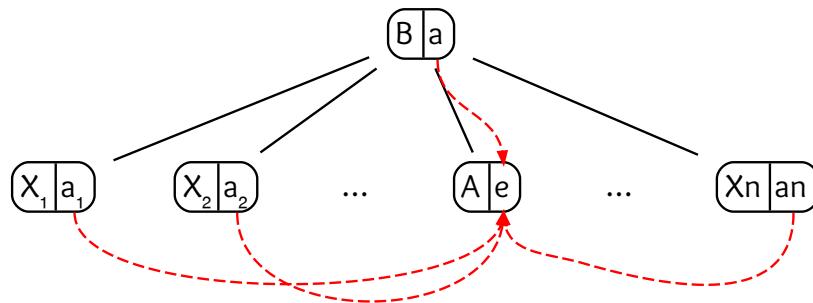
Osservazione

- L'attributo v usato per la valutazione delle espressioni è sintetizzato.

Attributi ereditati

Definizione

Un attributo di un nodo N in un albero annotato si dice **ereditato** se il suo valore dipende da quello di attributi del padre e dei fratelli di N .



Il valore di un attributo ereditato per un nodo etichettato con la variabile A è determinato da una regola semantica associata a una produzione per B (etichetta del nodo padre) nel cui corpo compare A :

$$B \rightarrow X_1 X_2 \cdots A \cdots X_n$$

$$A.e = f(B.a, X_1.a_1, X_2.a_2, \dots, X_n.a_n)$$

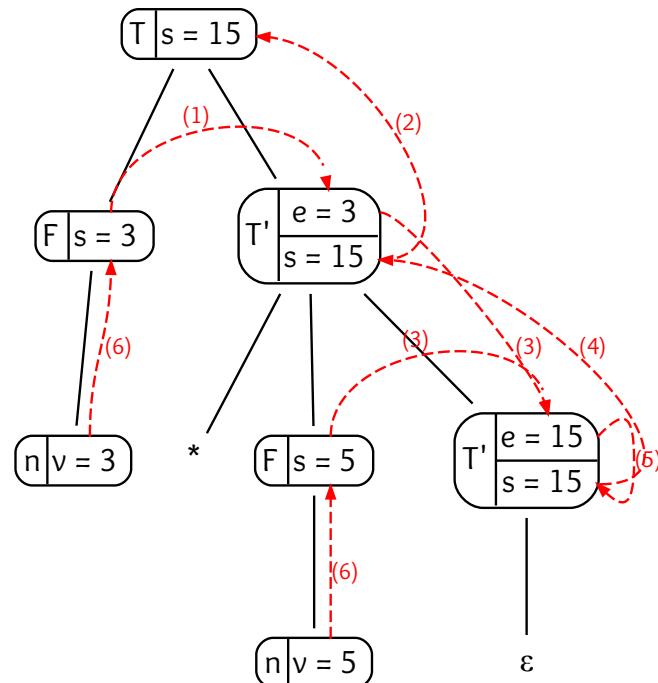
Esempio: espressioni senza ricorsione sinistra

SDD

Albero annotato per $3*5$

Produzioni	Regole semantiche
$T \rightarrow FT'$	(1) $T'.e = F.s$
	(2) $T.s = T'.s$
$T' \rightarrow *FT'_1$	(3) $T'_1.e = T'.e \times F.s$
	(4) $T'.s = T'_1.s$
$T' \rightarrow \epsilon$	(5) $T'.s = T'.e$
$F \rightarrow n$	(6) $F.s = n.v$

- $n.v$ = valore del numero (dal lexer)
- $T.s$ = valore del termine
- $F.s$ = valore del fattore
- $T'.e$ = prodotto dei fattori a sinistra di T'
- $T'.s$ = prodotto dei fattori a sinistra di e generati da T'



Esempio: lista delle differenze

SDD

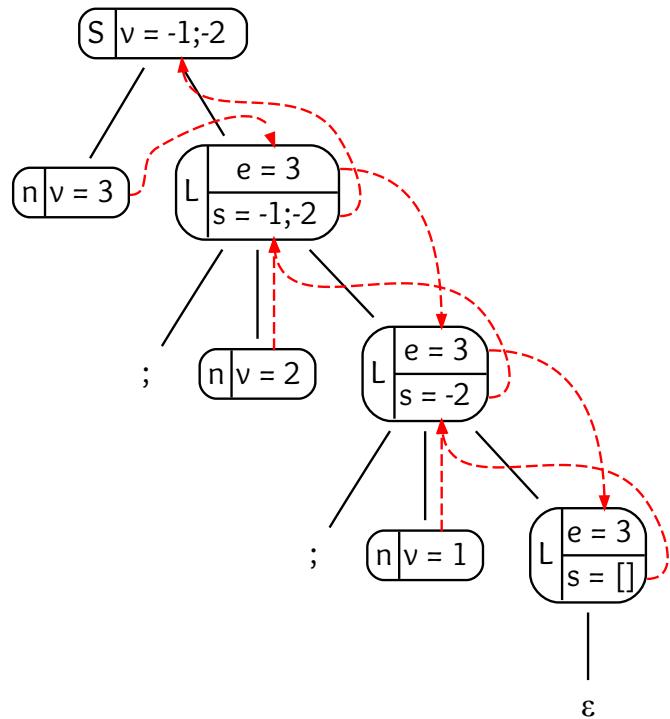
Albero annotato per 3;2;1

Produzioni	Regole semantiche
$S \rightarrow n \ L$	$L.e = n.v$
	$S.v = L.s$
$L \rightarrow \epsilon$	$L.s = []$ (lista vuota)
$L \rightarrow ; \ n \ L_1$	$L_1.e = L.e$ $L.s = n.v - L.e \parallel L_1.s$

- $n.v$ = valore del numero (dal lexer)
- $S.v$ = risultato
- $L.e$ = primo elemento della lista
- $L.s$ = risultato parziale

Esempio

$$3;2;1 \Rightarrow -1;-2$$



Esempio: da forma prefissa a forma infissa

Esempio

$$*3+21 \implies 3 * (2 + 1)$$

Intuizione

Per minimizzare il numero di parentesi usiamo un attributo ereditato $E.c \subseteq \{+, *\}$ che indica gli operatori che, se presenti in E , richiedono le parentesi.

La seguente funzione ausiliaria racchiude una stringa s tra parentesi se questa compare in un contesto in cui l'operatore o le richiede:

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

SDD

Produzioni	Regole semantiche
$S \rightarrow E$	$E.c = \emptyset$ $S.i = E.i$
$E \rightarrow +E_1E_2$	$E_1.c = \emptyset$ $E_2.c = \{+\}$ $E.i = \text{PAR}(+, E_1.i \parallel "+" \parallel E_2.i)$
$E \rightarrow *E_1E_2$	$E_1.c = \{+\}$ $E_2.c = \{+, *\}$ $E.i = \text{PAR}(*, E_1.i \parallel "*" \parallel E_2.i)$
$E \rightarrow \mathbf{n}$	$E.i = \mathbf{n}.v$

- $\mathbf{n}.v$ = cifra (dal lexer)
- $E.c$ = operatori da racchiudere in (...) se incontrati in E
- $E.i$ = forma infissa di E
- \parallel = concatenazione tra stringhe

Ordine di valutazione degli attributi

Grafo delle dipendenze

Le regole semantiche inducono un **grafo di dipendenze** tra attributi. Se l'attributo $A.a$ dipende dall'attributo $B.b$, è necessario conoscere il valore di $B.b$ prima di calcolare $A.a$:

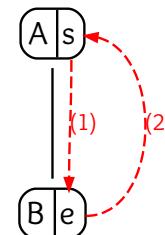
$$A.a = f(\dots, B.b, \dots)$$

Negli esempi precedenti, il grafo è definito dalle **frecce rosse tratteggiate**.

Dipendenze circolari

Se il grafo delle dipendenze contiene dei **cicli**, non è possibile trovare un ordine di valutazione degli attributi:

Produzione	Regole semantiche
$A \rightarrow B$	(1) $A.s = B.e$
	(2) $B.e = A.s + 1$



Definizioni S-attribuite ed L-attribuite

Definizione

Una definizione diretta dalla sintassi si dice **S-attribuita** se contiene solo attributi sintetizzati.

Definizione

Una definizione diretta dalla sintassi si dice **L-attribuita** se, per ogni produzione $A \rightarrow X_1X_2 \cdots X_n$ ed ogni attributo ereditato $X_i.e$, la regola semantica che definisce il valore di $X_i.e$ dipende solo da:

1. attributi ereditati di A ;
2. attributi sintetizzati ed ereditati dei simboli X_1, X_2, \dots, X_{i-1} alla sinistra di X_i .

Osservazioni

- Ogni SDD S-attribuita è anche L-attribuita.
- Ogni SDD L-attribuita ha un grafo delle dipendenze aciclico, in quanto gli attributi sintetizzati hanno solo dipendenze “dal basso verso l’alto” mentre quelli ereditati solo “dall’alto verso il basso” e/o “da sinistra verso destra”.
- Tutte le SDD viste fino ad ora sono S-attribuite o L-attribuite.

Esercizi

1. Definire una SSD per la grammatica seguente che traduca sequenze di bit con la cifra meno significativa più a sinistra nel numero naturale in base 10 corrispondente:

- $S \rightarrow BL$
- $L \rightarrow \epsilon \mid BL$
- $B \rightarrow 0 \mid 1$

Alcuni esempi di traduzione: $1010 \Rightarrow 5$ e $1011 \Rightarrow 13$.

2. Ripetere l'esercizio precedente assumendo che la cifra meno significativa sia quella più a destra.
Alcuni esempi di traduzione: $1010 \Rightarrow 10$ e $1011 \Rightarrow 11$.

3. In riferimento alla SSD della [slide 13](#), disegnare l'albero annotato per ***3+21**.

Linguaggi Formali e Traduttori

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

In questa lezione

1. Introduciamo gli **schemi di traduzione** (SDT), una variante delle SSD in cui si rende esplicito l'ordine di valutazione degli attributi.
2. Vediamo come convertire una SSD 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 **schemma 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 e C
$A \rightarrow BC$	$A \rightarrow B\{code\}C$	<i>code</i> eseguito dopo il riconoscimento di B ma prima del riconoscimento di C
$A \rightarrow BC$	$A \rightarrow \{code_1\}BC\{code_2\}$	$code_1$ eseguito subito dopo la riscrittura di A e prima del riconoscimento di B , $code_2$ eseguito dopo il riconoscimento di B e C
$A \rightarrow \epsilon$	$A \rightarrow \{code\}$	<i>code</i> eseguito subito dopo la riscrittura di A

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 SSD L-attribuite a SDT

Algoritmo

Data una SSD L-attribuita, si può ottenere uno SDT corrispondente nel modo seguente. Per ogni produzione $A \rightarrow X_1 X_2 \cdots X_n$ 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 SSD 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 SSD 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
$T \rightarrow F$	$\{T'.e = F.s\}$
T'	$\{T.s = T'.s\}$
$T' \rightarrow *F$	$\{T'_1.e = T'.e \times F.s\}$
T'_1	$\{T'.s = T'_1.s\}$
$T' \rightarrow \epsilon$	$\{T'.s = T'.e\}$
$F \rightarrow n$	$\{F.s = n.v\}$

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 $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ 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 $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ 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
$S \rightarrow \epsilon$	$\{S.n = 0\}$
$S \rightarrow aS_1b$	$\{S.n = S_1.n + 1\}$

Esempi

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

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

Schema di traduzione

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

Esempi

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

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	$\{T'.s = T'_1.s\}$
$T' \rightarrow \epsilon$	$\{T'.s = T'.e\}$
$F \rightarrow \mathbf{n}$	$\{F.s = \mathbf{n}.v\}$

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	$\{T'.s = T'_1.s\}$
$T' \rightarrow \epsilon$	$\{T'.s = T'.e\}$
$F \rightarrow n$	$\{F.s = n.v\}$

```
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	$\{E.i = \text{PAR}(*, E_1.i \parallel "*" \parallel E_2.i)\}$
$E \rightarrow n$	$\{E.i = n.v\}$

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

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

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

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

- $E.\text{post} = E$ in forma postfissa
- $E.\text{post}$ è un attributo principale

Esempio: da forma infissa a forma postfissa

SDT con accumulo della traduzione

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

- $E.\text{post} = E$ in forma postfissa
- $E.\text{post}$ è un attributo principale

Traduzione “on-the-fly”

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

- 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.

Linguaggi Formali e Traduttori

5.3 Codice intermedio

- Sommario
- Java Virtual Machine
- Componenti della JVM
- Struttura di un frame della JVM
- Gestione della pila degli operandi
- Operazioni aritmetiche e su bit
- Gestione degli array
- Controllo di flusso

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

- Stabilito che il programma da tradurre è sintatticamente corretto, il compilatore lo deve tradurre in un programma equivalente scritto in **codice intermedio**.

In questa lezione

- Adottiamo il bytecode di Java come codice intermedio per la traduzione.
- Riepiloghiamo la struttura e il significato delle istruzioni più importanti della Java Virtual Machine (JVM).

Riferimenti esterni

- [Java Language and Virtual Machine Specifications](#)
- [JVM Instruction set](#)

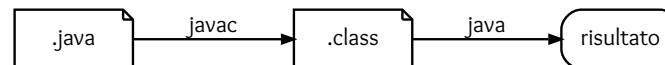
Java Virtual Machine

La **Java Virtual Machine** (JVM) è un **interprete** in grado di eseguire **bytecode**.

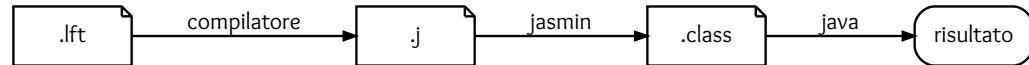
Caratteristiche principali della JVM

- Macchina virtuale basata su **pila**.
- Istruzioni di basso livello (gestione della pila) ed alto livello (oggetti).
- **Garbage collector** (la memoria inutilizzata viene reclamata automaticamente).

Uso tipico della JVM



In questo corso



- Il file `.j` contiene bytecode JVM in formato mnemonico (facile da produrre/leggere).
- Il file `.class` contiene bytecode JVM in formato binario.
- Usiamo **Jasmin** per tradurre il bytecode dal formato mnemonico a quello binario.

Componenti della JVM

- Un'area metodi che contiene il bytecode corrispondente ai metodi di tutte le classi usate da un'applicazione.
- Un insieme di **registri** che contengono informazioni essenziali sullo stato della macchina virtuale. Tra tutti, il **program counter** contiene l'indirizzo della prossima istruzione da eseguire.
- Una **pila di frame**, uno per ogni metodo in esecuzione. A sua volta, ogni frame è composto da:
 - una **pila degli operandi** usata per la valutazione di risultati temporanei;
 - un **array di variabili locali** usato per la memorizzazione delle variabili locali e degli argomenti del metodo.
- Un **heap** in cui vengono allocati gli oggetti.

Nota

Nelle applicazioni con thread multipli in esecuzione esistono copie distinte della pila dei frame e di alcuni registri per ciascun thread.

Struttura di un frame della JVM

Ogni **frame** corrisponde a un metodo in esecuzione e contiene:

- **argomenti e variabili locali** (indirizzati a partire da 0);
- **pila degli operandi** (cresce/cala durante l'esecuzione del metodo).

Nome	Slot n.	Valore
a	0	42
b	1	true
x	2	7
y	3	23
z	4	'c'
—	—	5
—	—	7
		:

```
static void m(int a, boolean b) {  
    int x, y;  
    char z;  
  
    ... 5 * x ...  
}
```

Nota

- Nei metodi non statici il primo argomento è il riferimento all'oggetto ricevente (this).

Gestione della pila degli operandi

Istruzione	Prima	Dopo	Descrizione
ldc v		v	carica v sulla pila
iload &x		v	carica il valore di x sulla pila
istore &x	v		assegna v a x
pop	v		rimuove il valore in cima alla pila
dup	v	$v\ v$	duplica il valore in cima alla pila
swap	$v_1\ v_2$	$v_2\ v_1$	scambia i due valori in cima alla pila

Note

- Il valore in cima alla pila è quello più a destra.
- Le istruzioni **iload** e **istore** hanno come argomento l'indirizzo – e non il nome – della variabile x nel frame del metodo corrente.

Operazioni aritmetiche e su bit

Istruzione	Prima	Dopo	Descrizione
ineg	v	v	negazione
iadd	$v_1 \ v_2$	v	somma $v_1 + v_2$
isub	$v_1 \ v_2$	v	sottrazione $v_1 - v_2$
imul	$v_1 \ v_2$	v	moltiplicazione $v_1 \times v_2$
idiv	$v_1 \ v_2$	v	divisione v_1/v_2
irem	$v_1 \ v_2$	v	resto della divisione v_1/v_2
iand	$v_1 \ v_2$	v	congiunzione bit a bit
ior	$v_1 \ v_2$	v	disgiunzione bit a bit

Note

- Il valore in cima alla pila è quello più a destra.
- Nelle operazioni binarie (es. **isub**) il secondo operando è quello in cima alla pila.

Gestione degli array

Istruzione	Prima	Dopo	Descrizione
newarray	n	a	crea un array di n elementi
arraylength	a	n	dimensione dell'array a
iaload	$a\ i$	v	carica $a[i]$ sulla pila
iastore	$a\ i\ v$		assegna v ad $a[i]$

Nota

- a è un riferimento all'array nell'heap.

Controllo di flusso

Istruzione	Prima	Dopo	Descrizione
goto <i>l</i>			salta a <i>l</i>
if_icmp eq <i>l</i>	<i>v</i> ₁ <i>v</i> ₂		salta a <i>l</i> se <i>v</i> ₁ = <i>v</i> ₂
if_icmp ne <i>l</i>	<i>v</i> ₁ <i>v</i> ₂		salta a <i>l</i> se <i>v</i> ₁ ≠ <i>v</i> ₂
if_icmp le <i>l</i>	<i>v</i> ₁ <i>v</i> ₂		salta a <i>l</i> se <i>v</i> ₁ ≤ <i>v</i> ₂
if_icmp ge <i>l</i>	<i>v</i> ₁ <i>v</i> ₂		salta a <i>l</i> se <i>v</i> ₁ ≥ <i>v</i> ₂
if_icmp lt <i>l</i>	<i>v</i> ₁ <i>v</i> ₂		salta a <i>l</i> se <i>v</i> ₁ < <i>v</i> ₂
if_icmp gt <i>l</i>	<i>v</i> ₁ <i>v</i> ₂		salta a <i>l</i> se <i>v</i> ₁ > <i>v</i> ₂
invokestatic <i>m</i>	<i>v</i> ₁ … <i>v</i> _{<i>n</i>}	<i>v?</i>	invoca <i>m</i> (<i>v</i> ₁ , …, <i>v</i> _{<i>n</i>})
return			termina il metodo
ireturn	<i>v</i>		termina il metodo restituendo <i>v</i>

Nota

- *v?* è presente solo se il metodo invocato ha un tipo di ritorno diverso da void.

Linguaggi Formali e Traduttori

5.4 Traduzione di espressioni aritmetiche

- Sommario
- Grammatica delle espressioni aritmetiche
- SDD per espressioni aritmetiche
- SDT per la grammatica ambigua
- SDT “on-the-fly” per la grammatica LL(1)
- 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

- Traduzione delle espressioni aritmetiche.

In questa lezione

- Definiamo SDD e SDT per la traduzione di espressioni aritmetiche.

Riferimenti esterni

- Java Language and Virtual Machine Specifications
- JVM Instruction set

Grammatica delle espressioni aritmetiche

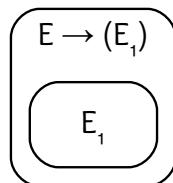
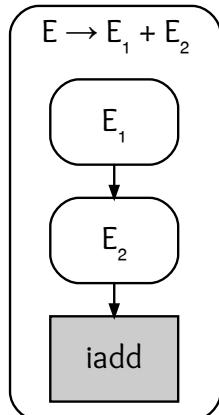
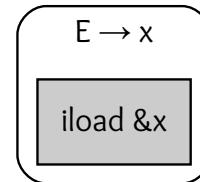
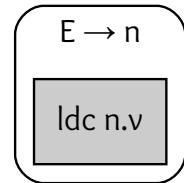
Produzioni	Descrizione
$E \rightarrow E_1 + E_2$	Somma
$E \rightarrow E_1 - E_2$	Sottrazione
$E \rightarrow E_1 * E_2$	Moltiplicazione
$E \rightarrow E_1 / E_2$	Divisione intera
$E \rightarrow E_1 \% E_2$	Resto della divisione intera
$E \rightarrow (E_1)$	Stesso valore di E_1
$E \rightarrow n$	Costante
$E \rightarrow x$	Variabile

Attenzione

Al fine di presentare la traduzione in codice intermedio adotteremo spesso grammatiche ambigue o comunque non LL(1). Disambiguazione, fattorizzazione, eliminazione della ricorsione spesso richiedono modifiche significative anche agli SDT corrispondenti, come l'introduzione di attributi ereditati.

SDD per espressioni aritmetiche

Produzioni	Regole semantiche
$E \rightarrow n$	$E.\text{code} = \text{ldc } n.v$
$E \rightarrow x$	$E.\text{code} = \text{iload } \&x$
$E \rightarrow E_1 + E_2$	$E.\text{code} = E_1.\text{code}$ $E_2.\text{code}$ iadd
$E \rightarrow (E_1)$	$E.\text{code} = E_1.\text{code}$



Attributi

- $E.\text{code}$ = codice che calcola il valore di E e lo lascia in cima alla pila.

SDT per la grammatica ambigua

SDT con accumulo del codice

Produzioni	Azioni semantiche
$E \rightarrow E_1 + E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{iadd}\}$
$E \rightarrow E_1 - E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{isub}\}$
$E \rightarrow E_1 * E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{imul}\}$
$E \rightarrow E_1 / E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{idiv}\}$
$E \rightarrow E_1 \% E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{irem}\}$
$E \rightarrow (E_1)$	$\{E.\text{code} = E_1.\text{code}\}$
$E \rightarrow \mathbf{n}$	$\{E.\text{code} = \text{ldc } \mathbf{n}.v\}$
$E \rightarrow x$	$\{E.\text{code} = \text{iload } \&x\}$

SDT per la grammatica ambigua

SDT con accumulo del codice

Produzioni	Azioni semantiche
$E \rightarrow E_1 + E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{iadd}\}$
$E \rightarrow E_1 - E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{isub}\}$
$E \rightarrow E_1 * E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{imul}\}$
$E \rightarrow E_1 / E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{idiv}\}$
$E \rightarrow E_1 \% E_2$	$\{E.\text{code} = E_1.\text{code} \parallel E_2.\text{code} \parallel \text{irem}\}$
$E \rightarrow (E_1)$	$\{E.\text{code} = E_1.\text{code}\}$
$E \rightarrow \text{n}$	$\{E.\text{code} = \text{ldc n. } v\}$
$E \rightarrow x$	$\{E.\text{code} = \text{iload } \&x\}$

SDT “on-the-fly”

Azioni semantiche
$\{\text{emit}(\text{iadd})\}$
$\{\text{emit}(\text{isub})\}$
$\{\text{emit}(\text{imul})\}$
$\{\text{emit}(\text{idiv})\}$
$\{\text{emit}(\text{irem})\}$
$\{\text{emit}(\text{ldc n. } v)\}$
$\{\text{emit}(\text{iload } \&x)\}$

SDT “on-the-fly” per la grammatica LL(1)

Produzioni	Azioni semantiche
$E \rightarrow TE'$	
$E' \rightarrow \epsilon$	
$E' \rightarrow +T$	{ <i>emit(iadd)</i> }
	E'
$E' \rightarrow -T$	{ <i>emit(isub)</i> }
	E'
$T \rightarrow FT'$	
$T' \rightarrow \epsilon$	
$T' \rightarrow *F$	{ <i>emit(imul)</i> }
	T'
...	
$F \rightarrow n$	{ <i>emit(ldc n.v)</i> }
$F \rightarrow x$	{ <i>emit(iload &x)</i> }
$F \rightarrow (E)$	

```
private void E'() {
    switch (peek()) {
        case '+': // E' → +TE'
            match('+');
            T();
            emit("iadd");
            E'();
            break;
        case '-': // E' → -TE'
            match('-');
            T();
            emit("isub");
            E'();
            break;
        case ')':
        case '$': // E' → ε
            break;
        default:
            throw error("E'");
    }
}
```

Esercizi

1. Calcolare il codice generato per l'espressione $x * x + 2 * x + 1$.
2. Scrivere le regole semantiche per tradurre la negazione $E \rightarrow \neg E_1$.
3. Scrivere le regole semantiche per tradurre l'accesso ad array $E \rightarrow E_1[E_2]$.
4. Scrivere le regole semantiche per tradurre l'invocazione di un metodo statico

$$E \rightarrow m(E_{list}) \quad E_{list} \rightarrow \varepsilon \mid E_{listp} \quad E_{listp} \rightarrow E \mid E, E_{listp}$$

usando l'istruzione **invokestatic** m per invocare m .

5. Scrivere le regole semantiche per tradurre l'assegnamento $E \rightarrow x = E_1$ ricordando che, in Java, tale comando è anche un'espressione il cui valore coincide con quello di E_1 . Suggerimento: usare l'istruzione dup per evitare di valutare E_1 due volte.
6. Scrivere le regole semantiche per tradurre pre- e post-incremento di variabili

$$E \rightarrow ++x \mid x++$$

tenendo presente la differente semantica delle due forme.

7. Scrivere le regole semantiche per la traduzione del post-incremento dell'elemento di un array $E \rightarrow E_1[E_2]++$ ricordando che il valore di tale espressione è quello dell'elemento prima dell'incremento. **DIFFICILE!** Per risolvere l'esercizio **usare dup2 e dup_x2**.

Linguaggi Formali e Traduttori

5.5 Traduzione di espressioni logiche

- Sommario
- Espressioni logiche con cortocircuito
- Costanti e relazioni
- Congiunzione e negazione
- Esempi
- 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

- In questa lezione presentiamo le SDD per la traduzione di espressioni logiche.

Riferimenti esterni

- Java Language and Virtual Machine Specifications
- JVM Instruction set

Espressioni logiche con cortocircuito

Produzioni	Descrizione
$B \rightarrow \text{true}$	Sempre vero
$B \rightarrow \text{false}$	Sempre falso
$B \rightarrow E_1 R E_2$	Confronto
$B \rightarrow B_1 \&\& B_2$	Congiunzione logica
$B \rightarrow B_1 B_2$	Disgiunzione logica
$B \rightarrow !B_1$	Negazione logica
$B \rightarrow (B_1)$	Stesso valore di B_1

Produzioni	Descrizione
$R \rightarrow ==$	Uguale
$R \rightarrow !=$	Diverso
$R \rightarrow <$	Minore
$R \rightarrow >$	Maggiore
$R \rightarrow <=$	Minore o uguale
$R \rightarrow >=$	Maggiore o uguale

Intuizione

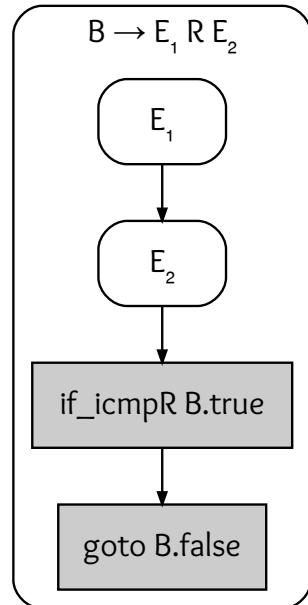
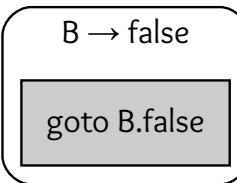
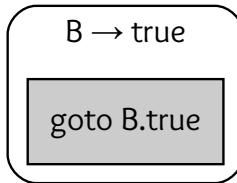
- I valori logici non hanno una rappresentazione concreta.
- Deve essere nota dal contesto l'etichetta a cui saltare a seconda del valore di B .

Attributi

- $B.\text{true}$ = etichetta a cui saltare se B è vera (ereditato)
- $B.\text{false}$ = etichetta a cui saltare se B è falsa (ereditato)
- $B.\text{code}$ = codice che salta a $B.\text{true}$ se B è vera o a $B.\text{false}$ se B è falsa.

Costanti e relazioni

Produzioni	Regole semantiche
$B \rightarrow \text{true}$	$B.\text{code} = \text{goto } B.\text{true}$
$B \rightarrow \text{false}$	$B.\text{code} = \text{goto } B.\text{false}$
$B \rightarrow E_1 R E_2$	$B.\text{code} = E_1.\text{code}$ $E_2.\text{code}$ $\text{if_icmpR } B.\text{true}$ $\text{goto } B.\text{false}$

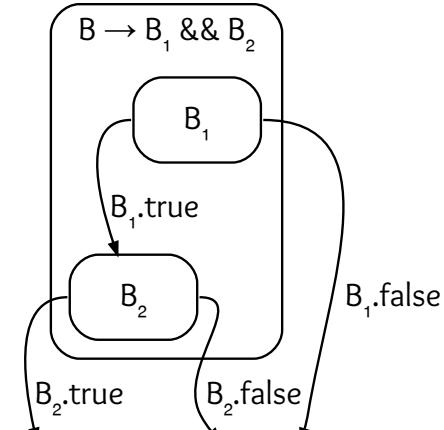


Congiunzione e negazione

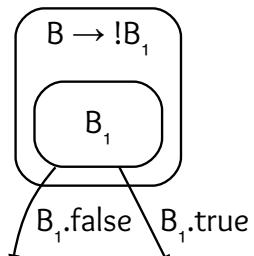
Produzioni	Regole semantiche
$B \rightarrow B_1 \&& B_2$	$B_1.\text{true} = \text{newlabel}()$ $B_1.\text{false} = B.\text{false}$ $B_2.\text{true} = B.\text{true}$ $B_2.\text{false} = B.\text{false}$ $B.\text{code} = B_1.\text{code}$ $\parallel B_1.\text{true} : B_2.\text{code}$
$B \rightarrow !B_1$	$B_1.\text{true} = B.\text{false}$ $B_1.\text{false} = B.\text{true}$ $B.\text{code} = B_1.\text{code}$

Nota

- La disgiunzione è duale della congiunzione, basta scambiare gli attributi **true** e **false** nella traduzione della congiunzione.



B.true: B.false:



B.true: B.false:

Esempi

Codice per $x < y$

```
iload &x  
iload &y  
if_icmplt B.true  
goto B.false
```

Codice per $0 \leq x \&& x < 5$

```
ldc 0  
iload &x  
if_icmpge L1  
goto B.false  
L1: iload &x  
ldc 5  
if_icmplt B.true  
goto B.false
```

Nota

- L_1 è una nuova etichetta

Osservazioni

- Il secondo operando di una congiunzione viene valutato solo se il primo è **true**.
- Il secondo operando di una disgiunzione viene valutato solo se il primo è **false**.
- La valutazione delle espressioni logiche è cortocircuitata.

Esercizi

1. Calcolare il codice generato traducendo l'espressione $!(x < y \ \&\& \ y < z)$.
2. Scrivere le regole semantiche per tradurre l'implicazione logica $B \rightarrow B_1 \Rightarrow B_2$ che è vera se e solo se B_1 è falsa o B_2 è vera.
3. Scrivere le regole semantiche per tradurre l'espressione ternaria $E \rightarrow B ? E_1 : E_2$ il cui valore è quello di E_1 o E_2 a seconda che B sia **true** o **false**, rispettivamente.

Linguaggi Formali e Traduttori

5.6 Traduzione di comandi

- Sommario
- Grammatica dei comandi
- Assegnamento
- Comandi condizionali
- Comandi iterativi
- Composizione sequenziale
- Esempio: somma dei primi 10 numeri naturali
- 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

- In questa lezione presentiamo le SDD per la traduzione di comandi.

Riferimenti esterni

- Java Language and Virtual Machine Specifications
- JVM Instruction set

Grammatica dei comandi

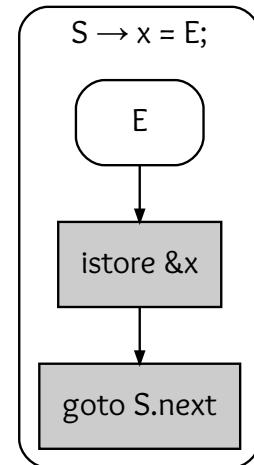
Produzioni	Descrizione
$S \rightarrow x = E;$	Assegna il valore di E ad x
$S \rightarrow \text{if } (B) S_1$	Esegue S_1 se B è vera, non fa nulla altrimenti
$S \rightarrow \text{if } (B) S_1 \text{ else } S_2$	Esegue S_1 se B è vera e S_2 altrimenti
$S \rightarrow \text{while } (B) S_1$	Esegue S_1 finché B è vera
$S \rightarrow \text{do } S_1 \text{ while } (B);$	Esegue S_1 finché B è vera e <u>almeno una volta</u>
$S \rightarrow \{S_{list}\}$	Esegue in sequenza i comandi in S_{list}
$S_{list} \rightarrow \epsilon$	Non fa nulla
$S_{list} \rightarrow SS_{list}$	Esegue S e poi i comandi in S_{list}

Attributi

- $S.\text{next}$ = etichetta a cui saltare dopo che S è stato eseguito (ereditato)
- $S.\text{code}$ = codice che esegue S e salta a $S.\text{next}$

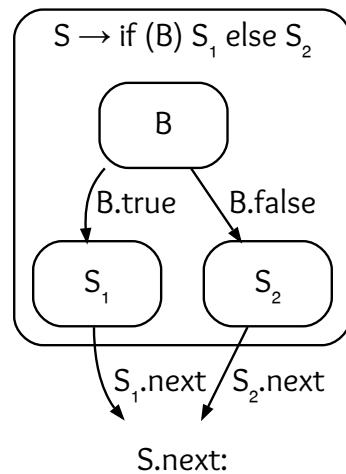
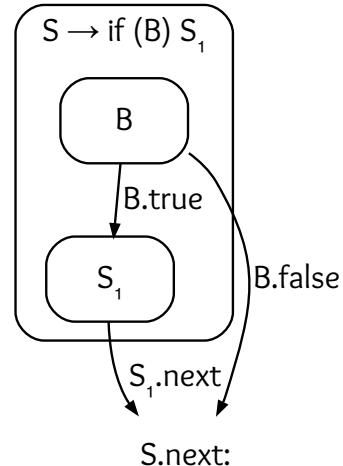
Assegnamento

Produzioni	Regole semantiche
$S \rightarrow x = E;$	$S.\text{code} = E.\text{code}$
	 istrong &x
	 goto S.next



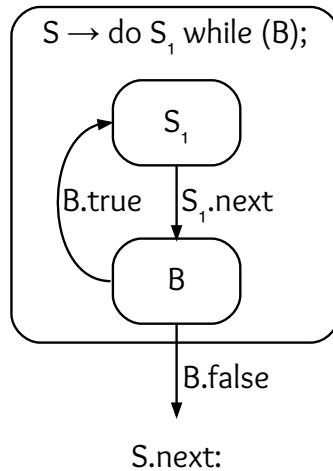
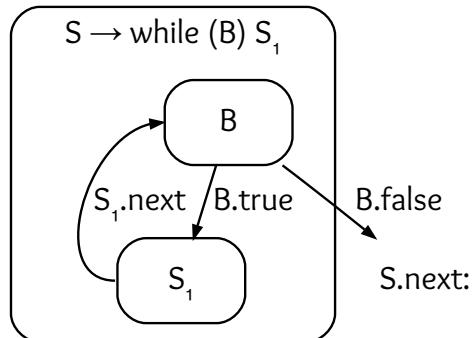
Comandi condizionali

Produzioni	Regole semantiche
$S \rightarrow \text{if } (B) S_1$	$B.\text{true} = \text{newlabel}()$ $B.\text{false} = S.\text{next}$ $S_1.\text{next} = S.\text{next}$ $S.\text{code} = B.\text{code}$ $\parallel B.\text{true} : S_1.\text{code}$
$S \rightarrow \text{if } (B) S_1 \text{ else } S_2$	$B.\text{true} = \text{newlabel}()$ $B.\text{false} = \text{newlabel}()$ $S_1.\text{next} = S.\text{next}$ $S_2.\text{next} = S.\text{next}$ $S.\text{code} = B.\text{code}$ $\parallel B.\text{true} : S_1.\text{code}$ $\parallel B.\text{false} : S_2.\text{code}$



Comandi iterativi

Produzioni	Regole semantiche
$S \rightarrow \text{while } (B) S_1$	$B.\text{true} = \text{newlabel}()$ $B.\text{false} = S.\text{next}$ $S_1.\text{next} = \text{newlabel}()$ $S.\text{code} = S_1.\text{next} : B.\text{code}$ $\parallel B.\text{true} : S_1.\text{code}$
$S \rightarrow \text{do } S_1 \text{ while } (B);$	$S_1.\text{next} = \text{newlabel}()$ $B.\text{true} = \text{newlabel}()$ $B.\text{false} = S.\text{next}$ $S.\text{code} = B.\text{true} : S_1.\text{code}$ $\parallel S_1.\text{next} : B.\text{code}$

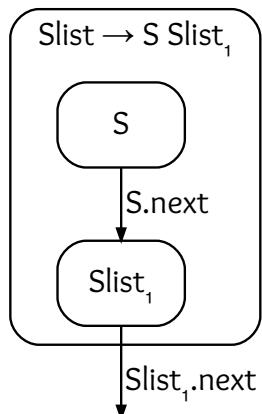
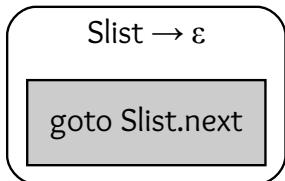


Composizione sequenziale

Produzioni	Regole semantiche
$S \rightarrow \{S_{list}\}$	$S_{list}.next = S.next$ $S.code = S_{list}.code$
$S_{list} \rightarrow \epsilon$	$S_{list}.code = \text{goto } S_{list}.next$
$S_{list} \rightarrow SS_{list_1}$	$S.next = \text{newlabel}()$ $S_{list_1}.next = S_{list}.next$ $S_{list}.code = S.code$ $\parallel S.next : S_{list_1}.code$

Attributi

- $S_{list}.next$ = etichetta a cui saltare dopo che tutti i comandi in S_{list} sono stati eseguiti (ereditato)
- $S_{list}.code$ = codice che esegue tutti i comandi in S_{list}



Slist.next:

Esempio: somma dei primi 10 numeri naturali

```
{  
    x = 10;  
    y = 0;  
    while (x > 0) {  
        y = x + y;  
        x = x - 1;  
    }  
}
```

Note

- L_0 è l'etichetta presente nell'attributo ereditato $S.\text{next}$ della variabile S da cui è stato generato il comando.
- Nel codice generato ci sono diversi goto ridondanti che possono essere eliminati nella fase di ottimizzazione del codice.

```
ldc 10  
istore &x  
goto L1  
L1: ldc 0  
istore &y  
goto L2  
L2:  
L3: iload &x  
ldc 0  
if_icmpgt L4  
goto L0  
L4: iload &x  
iload &y  
iadd  
istore &y  
goto L5  
L5: iload &x  
ldc 1  
isub  
istore &x  
goto L3  
L0:
```

Esercizi

1. Scrivere le regole semantiche per la traduzione del doppio assegnamento simultaneo $S \rightarrow x, y = E_1, E_2;$, che può essere utile per scambiare il contenuto di due variabili senza usare una variabile di appoggio (es. $x, y = y, x;$). Cosa contiene la variabile m al termine dell'esecuzione del seguente programma?

```
{ m, n = 0, 1;
  k = 10;
  while (k > 0) {
    m, n = n, m + n;
    k = k - 1;
  }
}
```

2. Scrivere le regole semantiche per tradurre il ciclo for $S \rightarrow \text{for } (S_1; B; S_2) \ S_3$

Linguaggi Formali e Traduttori

5.7 Analisi statica e traduzione di metodi

- Sommario
- Grammatica
- Note
- Esempio
- SDD per invocazione di un metodo e ritorno
- Verifica della presenza di return
- SDD per la verifica della presenza di return
- Allocazione delle variabili locali
- SDD per il calcolo degli slot di variabili locali
- Esempio
- Calcolo della dimensione massima della pila
- SDD per pila di espressioni aritmetiche
- Esempio: associatività degli operatori
- SDD per pila di comandi e metodi
- Traduzione di metodi statici
- 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

- La compilazione di un metodo comporta il calcolo della dimensione del suo **frame**, che comprende le **variabili locali** e la **pila degli operandi**.
- Il compilatore deve determinare questi valori **senza eseguire il codice del metodo**.
- Il compilatore deve assicurarsi che un metodo con tipo di ritorno diverso da `void` restituisca sempre un valore.

In questa lezione

- Presentiamo le SDD per la definizione e l'invocazione di metodi.
- Introduciamo alcune semplici forme di **analisi statica del codice** per calcolare la dimensione dei frame e individuare metodi errati.

Riferimenti esterni

- Java Language and Virtual Machine Specifications
- JVM Instruction set

Grammatica

Produzioni	Descrizione
$E \rightarrow \dots$	Come in precedenza
$E \rightarrow m(E_{list})$	Invocazione di metodo
$E_{list} \rightarrow \varepsilon$	Nessun argomento
$E_{list} \rightarrow E_{listp}$	Uno o più argomenti
$E_{listp} \rightarrow E$	Un argomento
$E_{listp} \rightarrow E, E_{listp}$	Due o più argomenti
$S \rightarrow \dots$	Come in precedenza
$S \rightarrow \mathbf{return} E;$	Ritorno da metodo
$S_{list} \rightarrow \dots$	Come in precedenza
$S_{list} \rightarrow T \ x = E; S_{list}$	Variabile locale
$M \rightarrow T \ m(T_1 \ x_1, \dots, T_n \ x_n) \ S$	Definizione di metodo
$T \rightarrow \mathbf{void} \mid \mathbf{int} \mid \dots$	Tipo

Note

SDD

- Le SDD presentate vanno integrate a quelle già presentate in precedenza per la generazione del codice di **espressioni e comandi**.

Metodi non statici

- Consideriamo solo **metodi statici**.
- I metodi **non statici** hanno un argomento implicito `this` (**l'oggetto ricevente**).

Tipo di ritorno

- Consideriamo solo metodi che restituiscono un valore intero.
- I metodi con tipo di ritorno `void` non possono essere invocati all'interno di un'espressione.

Esempio

```
int fibo(int n) {
    if (n <= 1) return n;
    else return fibo(n-1) + fibo(n-2);
}

int fibo(int k) {
    int m = 0;
    int n = 1;
    while (k >= 0) {
        int t = m;
        m = n;
        n = m + n;
        k = k - 1;
    }
    return m;
}
```

```
.method fibo(I)I
    .limit stack 3
    .limit locals 1
    ...
    ...

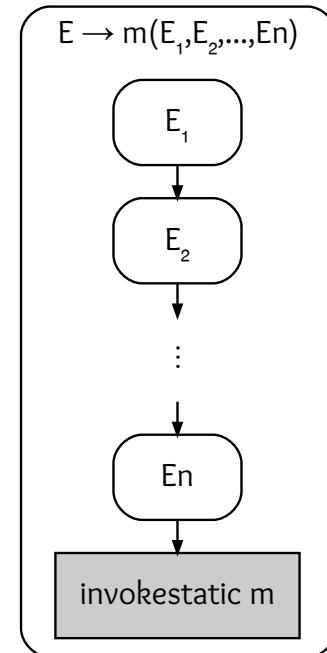
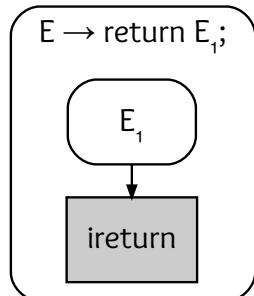
.method fibo(I)I
    .limit stack 2
    .limit locals 4
    ...
    ...
    ...
```

Dimensione del frame di un metodo

- Numero di argomenti e variabili locali del metodo (locals=1 e locals=4).
 - Dimensione massima della pila degli operandi (stack=3 e stack=2).

SDD per invocazione di un metodo e ritorno

Produzioni	Regole semantiche
$E \rightarrow m(E_{list})$	$E.\text{code} = E_{list}.\text{code} \parallel \text{invokestatic } m$
$E_{list} \rightarrow \epsilon$	$E_{list}.\text{code} = []$
$E_{list} \rightarrow E_{listp}$	$E_{list}.\text{code} = E_{listp}.\text{code}$
$E_{listp} \rightarrow E$	$E_{listp}.\text{code} = E.\text{code}$
$E_{listp} \rightarrow E, E_{listp1}$	$E_{listp}.\text{code} = E.\text{code} \parallel E_{listp1}.\text{code}$
$S \rightarrow \text{return } E;$	$S.\text{code} = E.\text{code} \parallel \text{ireturn}$



Verifica della presenza di return

Problema

- Un metodo con tipo di ritorno diverso da void deve restituire un **risultato** al chiamante per mezzo del comando return.

Strategia per l'analisi

- Si analizza il codice del metodo per verificare che ogni cammino di esecuzione porti dall'inizio del metodo a una istruzione return.
- L'analisi è **statica**, dunque non tiene conto dell'effettivo flusso di esecuzione del metodo e non garantisce che return sia davvero eseguito, per esempio se l'esecuzione entra in un **ciclo infinito** o se viene **lanciata un'eccezione** (es. divisione per zero).

Approssimazioni

- Non teniamo conto del valore delle espressioni logiche in comandi condizionali e cicli, anche quando sono banali (es. true). In generale questo problema è **indecidibile**.
- Non controlliamo se il valore restituito da return è del tipo giusto. Questo controllo richiede una forma aggiuntiva di analisi statica detta **controllo dei tipi**.

SDD per la verifica della presenza di return SDD

Attributi

- $S.\text{ret}$ = se l'esecuzione di S termina, è perché esegue **return**

Produzioni	Regole semantiche
$S \rightarrow x = E;$	$S.\text{ret} = \text{false}$
$S \rightarrow \text{if } (B) S_1$	$S.\text{ret} = \text{false}$
$S \rightarrow \text{if } (B) S_1 \text{ else } S_2$	$S.\text{ret} = S_1.\text{ret} \wedge S_2.\text{ret}$
$S \rightarrow \text{while } (B) S_1$	$S.\text{ret} = \text{false}$
$S \rightarrow \text{do } S_1 \text{ while } (B);$	$S.\text{ret} = S_1.\text{ret}$
$S \rightarrow \{S_{list}\}$	$S.\text{ret} = S_{list}.\text{ret}$
$S \rightarrow \text{return } E;$	$S.\text{ret} = \text{true}$
$S_{list} \rightarrow \epsilon$	$S_{list}.\text{ret} = \text{false}$
$S_{list} \rightarrow S S_{list1}$	$S_{list}.\text{ret} = S.\text{ret} \vee S_{list1}.\text{ret}$
$S_{list} \rightarrow T x = E; S_{list1}$	$S_{list}.\text{ret} = S_{list1}.\text{ret}$

SDD per la verifica della presenza di return SDD

Produzioni	Regole semantiche
$S \rightarrow x = E;$	$S.\text{ret} = \text{false}$
$S \rightarrow \text{if } (B) S_1$	$S.\text{ret} = \text{false}$
$S \rightarrow \text{if } (B) S_1 \text{ else } S_2$	$S.\text{ret} = S_1.\text{ret} \wedge S_2.\text{ret}$
$S \rightarrow \text{while } (B) S_1$	$S.\text{ret} = \text{false}$
$S \rightarrow \text{do } S_1 \text{ while } (B);$	$S.\text{ret} = S_1.\text{ret}$
$S \rightarrow \{S_{list}\}$	$S.\text{ret} = S_{list}.\text{ret}$
$S \rightarrow \text{return } E;$	$S.\text{ret} = \text{true}$
$S_{list} \rightarrow \epsilon$	$S_{list}.\text{ret} = \text{false}$
$S_{list} \rightarrow S S_{list1}$	$S_{list}.\text{ret} = S.\text{ret} \vee S_{list1}.\text{ret}$
$S_{list} \rightarrow T x = E; S_{list1}$	$S_{list}.\text{ret} = S_{list1}.\text{ret}$

Attributi

- $S.\text{ret}$ = se l'esecuzione di S termina, è perché esegue **return**

Note

- In una sequenza $S S_{list}$ in cui $S.\text{ret} = \text{true}$ la continuazione S_{list} non viene mai eseguita ed è detta codice morto.
- La presenza di codice morto non impedisce la compilazione ma è probabilmente sintomo di un errore.
- Il compilatore lo segnala con un avvertimento (warning) o un errore (es. javac).

Allocazione delle variabili locali

Problema

- Determinare il **più piccolo numero di slot** necessari all'interno di un frame per la memorizzazione di argomenti e variabili locali (“più piccolo” = risparmio di memoria).

Strategia

- Determinare il **numero massimo** di variabili che sono **contemporaneamente attive**.

Esempi

```
void sequenza() {  
    int x = 42;  
}  
  
{  
    int y = 15;  
}
```

```
void alternativa() {  
    if (true) {  
        int x = 42;  
    } else {  
        int y = 15;  
    }  
}
```

- In entrambi i casi x ed y non sono **mai** attive contemporaneamente e possono **condividere lo stesso slot** nel frame del metodo.

SDD per il calcolo degli slot di variabili locali

Produzioni	Regole semantiche
$S \rightarrow x = E;$	$S.\text{locals} = 0$
$S \rightarrow \text{if } (B) S_1$	$S.\text{locals} = S_1.\text{locals}$
$S \rightarrow \text{if } (B) S_1 \text{ else } S_2$	$S.\text{locals} = \max\{S_1.\text{locals}, S_2.\text{locals}\}$
$S \rightarrow \text{while } (B) S_1$	$S.\text{locals} = S_1.\text{locals}$
$S \rightarrow \text{do } S_1 \text{ while } (B);$	$S.\text{locals} = S_1.\text{locals}$
$S \rightarrow \{S_{list}\}$	$S.\text{locals} = S_{list}.\text{locals}$
$S \rightarrow \text{return } E;$	$S.\text{locals} = 0$
$S_{list} \rightarrow \epsilon$	$S_{list}.\text{locals} = 0$
$S_{list} \rightarrow S S_{list1}$	$S_{list}.\text{locals} = \max\{S.\text{locals}, S_{list1}.\text{locals}\}$
$S_{list} \rightarrow T x = E; S_{list1}$	$S_{list}.\text{locals} = 1 + S_{list1}.\text{locals}$

- $S.\text{locals}$ = max numero di variabili contemporaneamente attive durante esecuzione di S

Esempio

```
void sequenza() {  
    {  
        int x = 42;  
    }  
    {  
        int y = 15;  
    }  
}  
  
.method sequenza()V  
    .limit locals 1  
    ldc 42  
    istore 0 ; x  
    goto L1  
L1: ldc 15  
    istore 0 ; y  
    goto L0  
L0: return  
.end method
```

```
void alternativa() {  
    if (true) {  
        int x = 42;  
    } else {  
        int y = 15;  
    }  
}
```

```
.method alternativa()V  
    .limit locals 1  
    goto L3  
L3: ldc 42  
    istore 0 ; x  
    goto L2  
L4: ldc 15  
    istore 0 ; y  
    goto L2  
L2: return  
.end method
```

Nota: x ed y condividono lo stesso slot 0 nei frame dei due metodi.

Calcolo della dimensione massima della pila

Problema

- Determinare il **numero massimo di slot** occupati sulla pila degli operandi durante l'esecuzione di un metodo.

Strategia

- Tenendo conto del codice prodotto dalla traduzione di espressioni e comandi, **approssimare per eccesso** la dimensione massima della pila.

Esempio

```
int metodo() {  
    if (true) return 0;  
    else return 1 + 2 * 3;  
}
```

```
.method metodo() I  
    .limit stack 3  
    .limit locals 0  
    goto L1  
L1: ldc 0  
    ireturn  
L2: ldc 1  
    ldc 2  
    ldc 3  
    imul  
    iadd  
    ireturn  
.end method
```

SDD per pila di espressioni aritmetiche

Produzioni	Regole semantiche
$E \rightarrow E_1 + E_2$	$E.\text{stack} = \max\{E_1.\text{stack}, 1 + E_2.\text{stack}\}$
$E \rightarrow (E_1)$	$E.\text{stack} = E_1.\text{stack}$
$E \rightarrow \mathbf{n}$	$E.\text{stack} = 1$
$E \rightarrow x$	$E.\text{stack} = 1$
$E \rightarrow m(E_{list})$	$E.\text{stack} = \max\{1, E_{list}.\text{stack}\}$
$E_{list} \rightarrow \epsilon$	$E_{list}.\text{stack} = 0$
$E_{list} \rightarrow E_{listp}$	$E_{list}.\text{stack} = E_{listp}.\text{stack}$
$E_{listp} \rightarrow E$	$E_{listp}.\text{stack} = E.\text{stack}$
$E_{listp} \rightarrow E, E_{listp1}$	$E_{listp}.\text{stack} = \max\{E.\text{stack}, 1 + E_{listp1}.\text{stack}\}$

- $E.\text{stack}$ = dimensione massima della pila durante la valutazione di E ($E.\text{stack} \geq 1$)
- $E_{list}.\text{stack}$ = dimensione massima della pila durante la valutazione cumulata di tutte le espressioni generate da E_{list} ($E_{list}.\text{stack} \geq 0$)
- Nel caso $E \rightarrow m(E_{list})$, l'1 serve per tenere conto del valore restituito dal metodo.

Esempio: associatività degli operatori

```
int sinistra() {  
    return 1 + 2 + 3 + 4 + 5;  
}
```

```
.method sinistra()  
    .limit stack 2  
    .limit locals 0  
    ldc 1  
    ldc 2  
    iadd  
    ldc 3  
    iadd  
    ldc 4  
    iadd  
    ldc 5  
    iadd  
    ireturn  
.end method
```

```
int destra() {  
    return 1 + (2 + (3 + (4 + 5)));  
}
```

```
.method destra()  
    .limit stack 5  
    .limit locals 0  
    ldc 1  
    ldc 2  
    ldc 3  
    ldc 4  
    ldc 5  
    iadd  
    iadd  
    iadd  
    iadd  
    iadd  
    ireturn  
.end method
```

Osservazione

- L'associatività a sinistra mantiene la pila piccola perché le sottoespressioni vengono valutate man mano che si incontrano, da sinistra verso destra.

SDD per pila di comandi e metodi

Produzioni	Regole semantiche
$S \rightarrow x = E;$	$S.\text{stack} = E.\text{stack}$
$S \rightarrow \text{if } (B) S_1$	$S.\text{stack} = \max\{B.\text{stack}, S_1.\text{stack}\}$
$S \rightarrow \text{if } (B) S_1 \text{ else } S_2$	$S.\text{stack} = \max\{B.\text{stack}, S_1.\text{stack}, S_2.\text{stack}\}$
$S \rightarrow \text{while } (B) S_1$	$S.\text{stack} = \max\{B.\text{stack}, S_1.\text{stack}\}$
$S \rightarrow \text{do } S_1 \text{ while } (B);$	$S.\text{stack} = \max\{S_1.\text{stack}, B.\text{stack}\}$
$S \rightarrow \{S_{list}\}$	$S.\text{stack} = S_{list}.\text{stack}$
$S \rightarrow \text{return } E;$	$S.\text{stack} = E.\text{stack}$
$S_{list} \rightarrow \varepsilon$	$S_{list}.\text{stack} = 0$
$S_{list} \rightarrow S S_{list1}$	$S_{list}.\text{stack} = \max\{S.\text{stack}, S_{list1}.\text{stack}\}$
$S_{list} \rightarrow T x = E; S_{list1}$	$S_{list}.\text{stack} = \max\{E.\text{stack}, S_{list1}.\text{stack}\}$

- $B.\text{stack}$ = dimensione massima della pila durante la valutazione di B (definire SDD come esercizio)
- $S.\text{stack}$ = dimensione massima della pila durante l'esecuzione di S

Traduzione di metodi statici

Produzioni	Regole semantiche
$M \rightarrow \mathbf{void} \ m(T_1 \ x_1, \dots, T_n \ x_n) \ S$	$S.\text{next} = \text{newlabel}()$ $M.\text{code} = \text{.method } m$.limit stack $S.\text{stack}$.limit locals $n + S.\text{locals}$ $S.\text{code}$ $S.\text{next} : \mathbf{return}$ (se $S.\text{ret} = \mathit{false}$) .end method
$M \rightarrow \mathbf{int} \ m(T_1 \ x_1, \dots, T_n \ x_n) \ S$	$S.\text{next} = \text{newlabel}()$ (etichetta inutilizzata) $M.\text{code} = \text{.method } m$.limit stack $S.\text{stack}$.limit locals $n + S.\text{locals}$ $S.\text{code}$.end method (se $S.\text{ret} = \mathit{false}$ errore)

Esercizi

1. Tradurre i seguenti metodi:

```
int min(int x, int y) {  
    if (x < y) return x;  
    else return y;  
}
```

```
int euclide(int a, int b) {  
    if (a == 0) return b;  
    while (b != 0)  
        if (a > b) a = a - b;  
        else b = b - a;  
    return a;  
}
```

```
int primo(int n) {  
    int i = 2;  
    while (i < n) {  
        if (n % i == 0)  
            return 0;  
        i = i + 1;  
    }  
    if (n >= 2) return 1;  
    else return 0;
```

2. Scrivere le regole semantiche per calcolare $E.\text{stack}$ nel caso della produzione
 $E \rightarrow B ? E_1 : E_2$.
3. Scrivere le regole semantiche per calcolare $S.\text{locals}$ e $S.\text{stack}$ per il ciclo for
 $S \rightarrow \text{for } (S_1; B; S_2) S_3$.