Traduzione guidata dalla sintassi

Attributi e definizioni guidate dalla sintassi

Maria Rita Di Berardini

Dipartimento di Matematica e Informatica
Università di Camerino
mariarita.diberardini@unicam.it

Introduzione

- Analisi sintattica: il flusso di token (analisi lessicale) viene raggruppato in frasi grammaticali rappresentate tramite alberi di derivazione di una grammatica libera da contesto.
- Questo tipo di rappresentazione intermedia ci permette di identificare operatori ed operandi delle espressioni e statements
- Costiuisce l'input della fase di analisi semantica:
  - verifica dell'esistenza di eventuali errori semantici
  - acquisizione di informazioni sui tipi utilizzate nelle fasi successive del processo di compilazione
  - type checking
- Come possiamo definire la semantica ai costrutti dei linguaggi di programmazione?

Introduzione

- Consideriamo la grammatica per il linguaggio delle espressioni naturali:

$$E \rightarrow E + T \mid E - T \mid T$$
$$T \rightarrow T * F \mid T/F \mid F$$
$$F \rightarrow (E) \mid \textbf{number}$$

- Definire la semantica di questo linguaggio significa definire il valore di ogni possibile espressione costruita applicando le produzioni della grammatica
- Associamo ad ogni non terminale della grammatica, e quindi ad ogni nodo del albero di derivazione per una data stringa, un attributo val che rappresenta appunto il suo valore: E.val, T.val, e così via
- A questo punto non ci resta che associare a ciascuna produzione della grammatica un regola, regola semantica, che ci dice come calcolare il valore di una espressione a partire da quello delle sue sottoespressioni

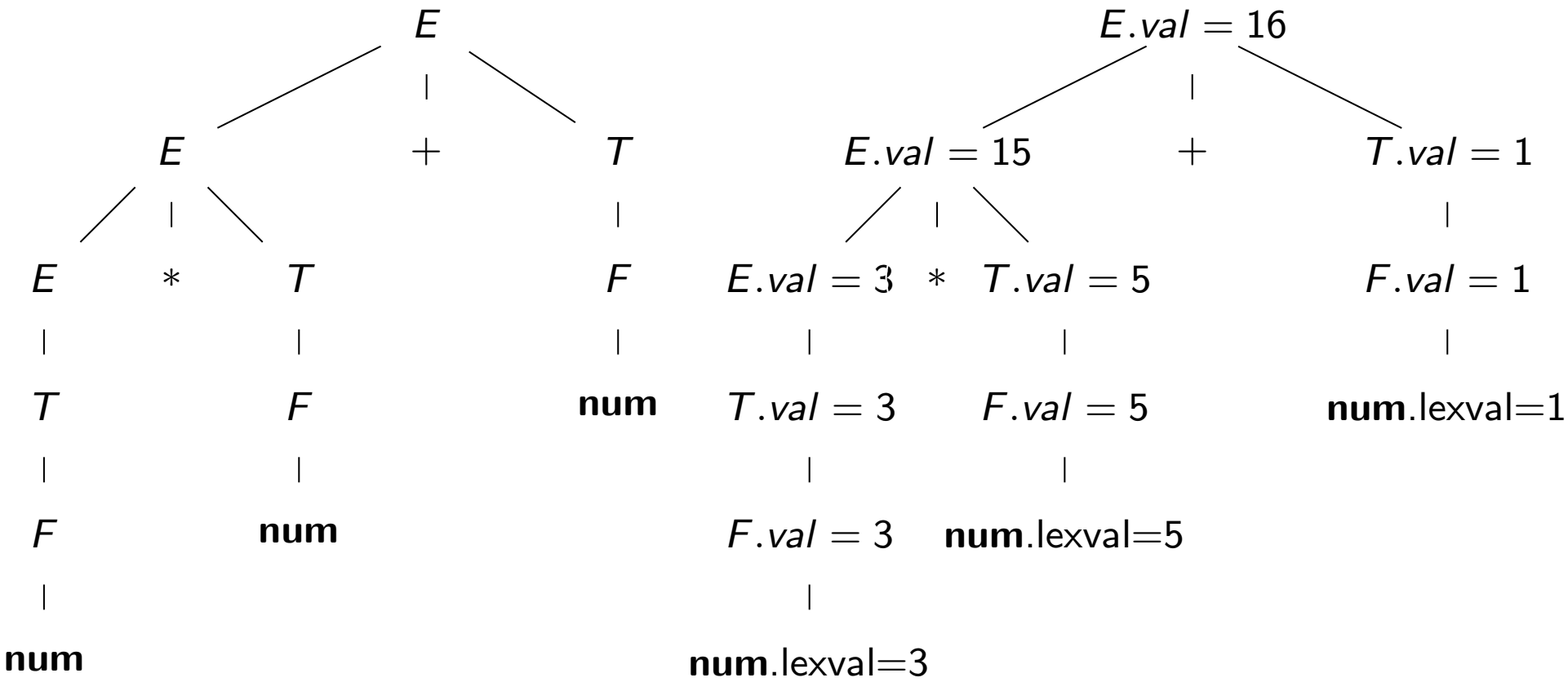
Introduzione

PRODUZIONI	REGOLE SEMANTICHE
$E \rightarrow E + T$	$E.val = E.val + T.val$
$E \rightarrow E - T$	$E.val = E.val - T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T * F$	$T.val = T.val * F.val$
$T \rightarrow T/F$	$T.val = T.val/F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow \mathbf{number}$	$F.val = \mathbf{number}.lexval$

Fornire un insieme di regole non è però sufficiente; bisogna anche fornire un ordine di valutazione: e quali regole applicare, e in che ordine, per calcolare il valore in questione

L'ordine di valutazione è definito dall'albero di derivazione per una data stringa

Albero di derivazione (annotato) della stringa $3 * 5 + 1$



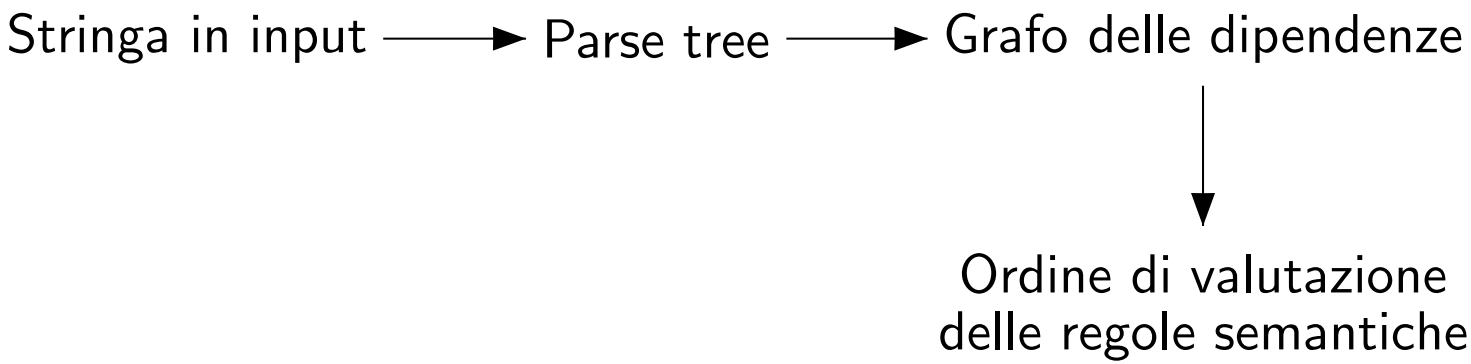
Obiettivi

- In questa ultima parte del corso vediamo, in breve, una tecnica che permette di effettuare analisi semantiche e traduzione usando la struttura sintattica data dalla grammatica di un linguaggio
- L'idea chiave è quella di associare, ad ogni costrutto del linguaggio, alcune informazioni utili per il nostro scopo
- L'informazione di ogni costrutto è rappresentata dal valore di diversi **attributi** associati a simboli non terminali della grammatica
- Il valore di ogni attributo è calcolato tramite delle **regole semantiche** associate con le produzioni della grammatica

Due diverse notazioni

- Esistono due diversi formalismi per definire le regole semantiche: definizioni guidate dalla sintassi e schemi di traduzione
- Le definizioni guidate dalla sintassi sono specifiche di alto livello: nascondono i dettagli implementativi e non richiedono di specificare l'ordine di valutazione che la traduzione deve seguire
- Gli schemi di traduzione, invece, indicano l'ordine in cui le regole semantiche devono essere valutate e quindi permettono la specifica di alcuni dettagli di implementazione
- Noi vedremo soprattutto le definizioni guidate dalla sintassi

Flusso concettuale dei dati



- Dalla stringa di input viene costruito il parse tree, il parse tree viene poi attraversato secondo l'ordine di valutazione delle regole semantiche che si trovano sui nodi
- L'ordine di valutazione delle regole semantiche è definito da un **grafo delle dipendenze**: definisce come attraversare l'albero di derivazione

Definizioni guidate dalla sintassi

- Sono generalizzazioni delle grammatiche in cui ad ogni simbolo della grammatica è associato un insieme di attributi
- Gli attributi possono essere di due tipi: **sintetizzati** ed **ereditati**
- Possiamo pensare ad ogni nodo del parse tree come ad un record i cui campi sono i nomi degli attributi
- Ogni attributo può stringhe, numeri, tipi, locazioni di memoria, etc.
- Il valore di ogni attributo ad ogni nodo è determinato da una regola semantica associata alla produzione che si usa nel nodo

Attributi sintetizzati ed ereditati

- Il valore di attributi **sintetizzati** di un dato nodo n è calcolato a partire dai valori degli **attributi dei nodi figli** di n
- Il valore di attributi **ereditati** di un dato nodo n è calcolato a partire dai valori degli attributi dei **nodi fratelli** e del **nodo padre** di n

Attributi sintetizzati: un esempio

L'attributo *val* per la grammatica delle espressioni è un tipico esempio di attributo sintetizzato

PRODUZIONI	REGOLE SEMANTICHE
$E \rightarrow E + T$	E.val = E.val + T.val
$E \rightarrow E - T$	E.val = E.val - T.val
$E \rightarrow T$	E.val = T.val
$T \rightarrow T * F$	T.val = T.val * F.val
$T \rightarrow T/F$	T.val = T.val / F.val
$T \rightarrow F$	T.val = F.val
$F \rightarrow (E)$	F.val = E.val
$F \rightarrow$ **number**	F.val = **number**.lexval

Attributi ereditati: un esempio

- Un attributo ereditato può essere usato per distribuire informazioni sul tipo fra i vari identificatori di una dichiarazione
- Una dichiarazione è costituita (in molti linguaggi di programmazione: C, Java, ...) da un identificare di tipo T seguito da una lista L di identificatori

PRODUZIONI	REGOLE SEMANTICHE
$D \rightarrow T\ L$	L.type := T.type
$T \rightarrow$ **int**	T.type=int
$T \rightarrow$ **real**	T.type = real
$L \rightarrow L_1$, **id**	L_1.type = L.type addtype(id.entry, L.type)
$L \rightarrow$ **id**	addtype(id.entry, L.type)

Attributi ereditati: un esempio

- L'attributo type è sintetizzato per T ed ereditato per L
- Inizialmente il valore di type è passato da T ad L (mediante la regola L.type := T.type)
- Durante la costruzione della lista ogni elemento passa al successivo il valore di type (mediante la regola L_1.type = L.type)
- La procedura addtype, data un entrata per la tabella dei simboli per un qualche identificare (id.entry) ed un tipo (L.type), aggiunge al record corrente informazioni riguardo il tipo

Dipendenze tra attributi

- Le regole semantiche inducono delle dipendenze tra il valore degli attributi che possono essere rappresentate con dei grafi (delle dipendenze)
- La valutazione, nel giusto ordine, delle regole semantiche determina il valore per tutti gli attributi dei nodi del parse tree di una stringa data
- La valutazione può avere anche side-effects (effetti collaterali) come la stampa di valori o l'aggiornamento di una variabile globale
- Un parse tree che mostri i valori degli attributi ad ogni nodo è detto **parse tree annotato**
- Il processo di calcolo di questi valori si dice **annotazione** o **decorazione del parse tree**

Forma di una definizione

- È una grammatica libera da contesto estesa in cui ogni produzione $A \longrightarrow \alpha$ ha associato un insieme di regole semantiche della forma

$$b ::= f(c_1, c_2, \ldots, c_k)$$

  dove f è una funzione (di solito espressa con delle espressioni) e b, $c_1, c_2, \ldots, c_k$ sono degli attributi
- Se b è un attributo sintetizzato di A allora $c_1, c_2, \ldots, c_k$ sono attributi dei simboli in α (figli di b)
- Se b è un attributo ereditato di un di simbolo di α allora $c_1, c_2, \ldots, c_k$ sono attributi di simboli di α oppure di A (fratelli e padre di b)
- In ogni caso l'attributo b **dipende dagli attributi** $c_1, c_2, \ldots, c_k$

Un esempio

Le regole semantiche possono avere degli effetti collaterali espressi mediante chiamate di procedura o frammenti di codice

Produzioni	Reg. Semantiche
$L \rightarrow E\mathbf{n}$	println(E.val);
$E \rightarrow E_1 + T$	$E.val = E_1.val \oplus T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val \otimes F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow \mathbf{num}$	$F.val = \mathbf{num}.lexval$

Un esempio

- La grammatica genera le espressioni aritmetiche tra cifre seguite dal carattere **n** di newline
- Ogni simbolo non terminale ha un **attributo sintetizzato** val
- Il simbolo terminale **num** ha un attributo sintetizzato lexval il cui valore è fornito dall'analizzatore lessicale
- La regola associata al simbolo iniziale L è una chaimata di procedura che stampa un valore intero (side-effect) mentre tutte le altre regole servono per il calcolo del valore degli attributi

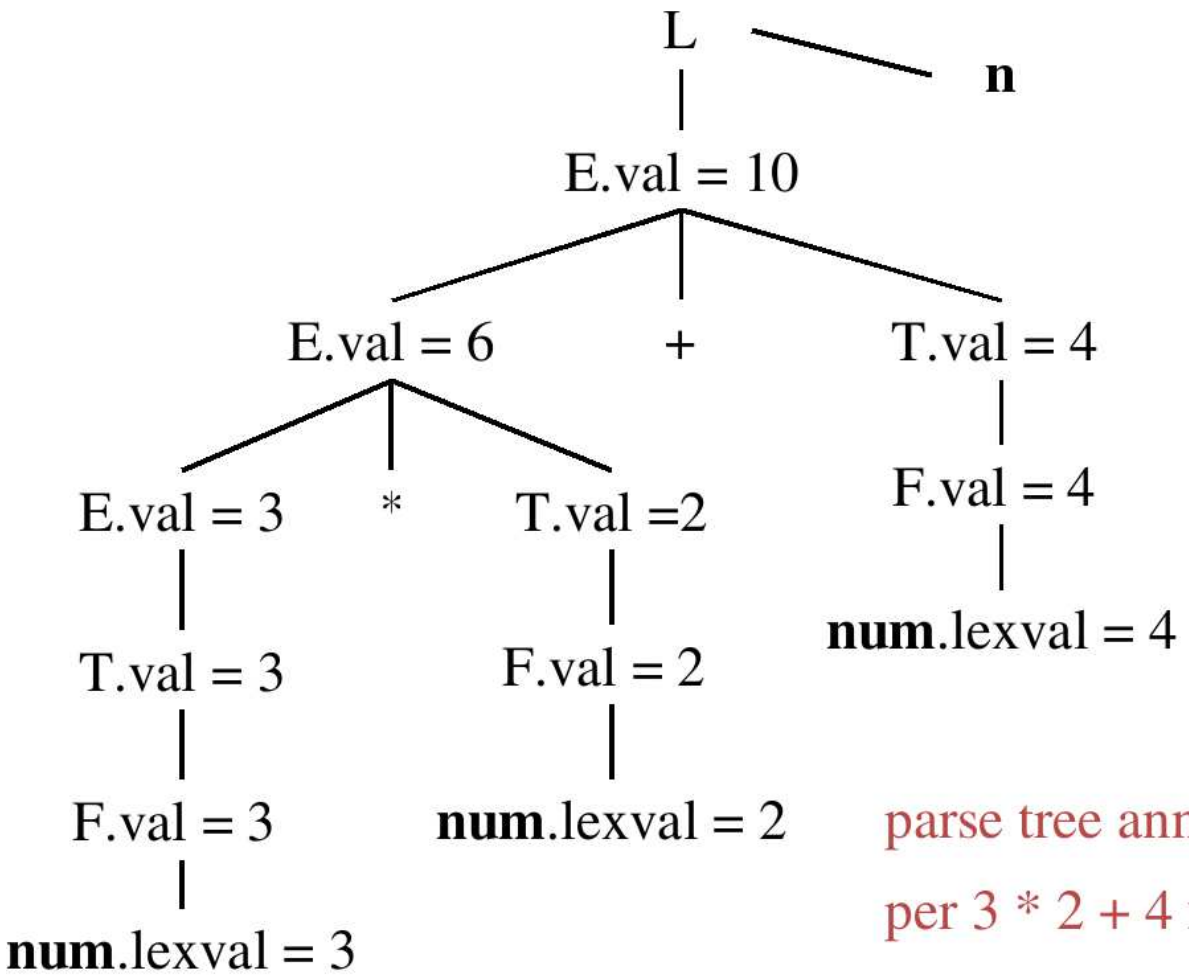
Assunzioni e convenzioni:

- I simboli terminali hanno solo attributi sintetizzati, i cui valori sono, in genere, forniti dall'analizzatore lessicale
- Il simbolo iniziale, se non diversamente specificato, non ha attributi ereditati

Definizioni S-attributed

- In pratica gli attributi sintetizzati sono i più usati
- Una definizione che usa solo attributi sintetizzati è detta **S-attributed**
- Un parse tree di una defizione S-attributed può sempre essere annotato valutando le regole semantiche per gli attributi in maniera bottom-up (dalle foglie alla radice)
- Possono quindi essere implementate facilmente durante il parsing LR
- Generatori automatici di LR-parser possono essere modificati per implementare una definizione S-attributed basata su una grammatica LR

Valutazione degli attributi in unq definizione S-attributed



parse tree annotato

per 3 * 2 + 4 **n**

Valutazione degli attributi in unq definizione S-attributed

- Consideriamo il nodo interno più in basso e più a sinistra per cui è usata la produzione $F \rightarrow \mathbf{num}$
- Dato che il valore dell'attributo lexval del nodo figlio **num** è 3, la corrispondente regola semantica F.val := num.lexval pone l'attributo val per F a 3
- Allo stesso modo nel nodo padre il valore di T.val è 3 (si applica la regola semantica T.val := F.val)
- Consideriamo il nodo con la produzione $T \rightarrow T * F$
- La regola semantica è T.val := T_1.val $\otimes$ F.val; il $\otimes$ è il corrispondente semantico dell'operatore sintattico * (una possibile implementazione della moltiplicazione fra interi)
- T_1.val è il valore dell'attributo alertval del primo figlio (quello più a sinistra) T, cioè 3

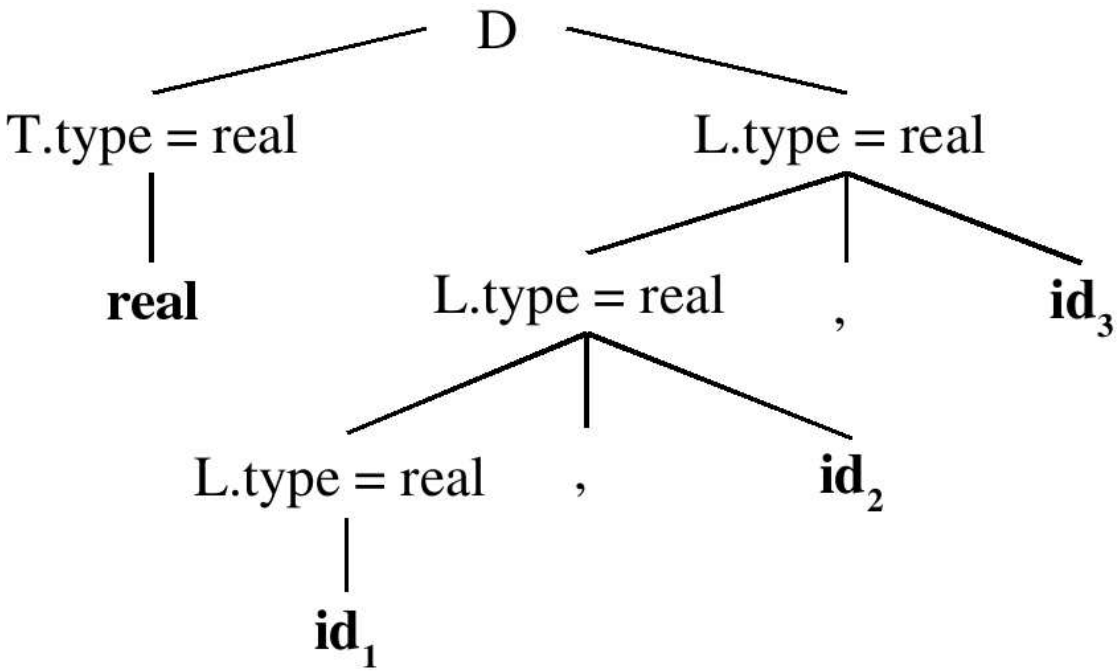
Attributi ereditati: un esempio

- Questa procedura non può essere utilizzata nel caso in cui la definizione (guidata dalla sintassi) contiene anche qualche attributo eriditato
- Consideriamo la definizione per le dichiarazioni

PRODUZIONI	REGOLE SEMANTICHE
$D \rightarrow T\ L$	L.type := T.type
$T \rightarrow$ **int**	T.type=int
$T \rightarrow$ **real**	T.type = real
$L \rightarrow L_1$, **id**	L_1.type = L.type addtype(id.entry, L.type)
$L \rightarrow$ **id**	addtype(id.entry, L.type)

Parse tree annotato per **real** id_1, id_2, id_3

- Questa procedura non può essere utilizzata nel caso in cui la definizione (guidata dalla sintassi) contiene anche qualche attributo eriditato
- Consideriamo la definizione per le dichiarazioni



Grafi delle dipendenze

- Se un attributo b in un nodo dipende da un attributo c allora la regola semantica per b deve essere valutata dopo la regola semantica che definisce c
- Le interdipendenze fra gli attributi ereditati e sintetizzati nei nodi di un parse tree possono essere agevolmente rappresentate da grafi (delle dipendenze)

Costruzione dei grafi delle dipendenze

- Prima di tutto rendiamo uniformi le regole semantiche ponendole tutte nella forma $b := f(c_1, c_2, \ldots, c_k)$
- Per le chiamate di procedure introduciamo un attributo fittizio, ad esempio la regola addtype(id.entry, L.type) può essere riscritta come b_{fitt} := addtype(id.entry, L.type)
- Il grafo ha un nodo per ogni attributo e un arco da c a b se l'attributo b dipende dal valore dell'attributo c

Algoritmo

for each nodo n nel parse tree **do**
    for each attributo a del nodo n **do**
        costruisci un nodo per a nel grafo;

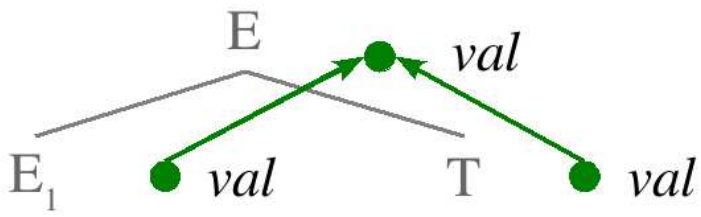
for each nodo n nel parse tree **do**
    for each regola semantica $b := f(c_1, c_2, \ldots, c_k)$ associata
        con una produzione usata in n **do**
        for i := 1 **to** k **do**
            aggiungi un arco dal nodo per c_i al nodo per b

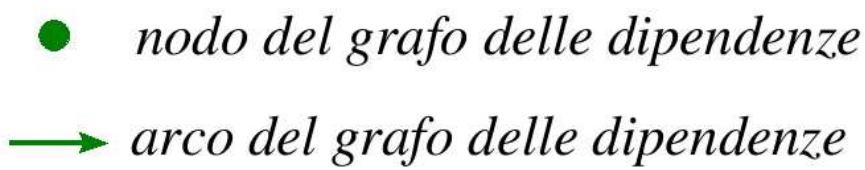
Algoritmo

- $A \rightarrow X\ Y$ con regola semantica associata $A.a := f(X.x, Y.y)$
- L'attributo sintetizzato a associato al non terminale A dipende dagli attributi x ed y di X ed Y risp.
- Se questa produzione è usata nel parse tree allora nel grafo ci sono tre nodi (uno per a, uno per x ed uno per y) e due archi: uno da x ad a e l'altro da y ad a
- $A \rightarrow X\ Y$ con regola semantica associata $X.x := g(A.a, Y.y)$
- L'attributo ereditato a associato al non terminale X dipende dagli attributi a ed y di A ed Y risp.
- Se questa produzione è usata nel parse tree allora nel grafo ci sono tre nodi (uno per a, uno per x ed uno per y) e due archi: uno da a ad x e l'altro da y ad x

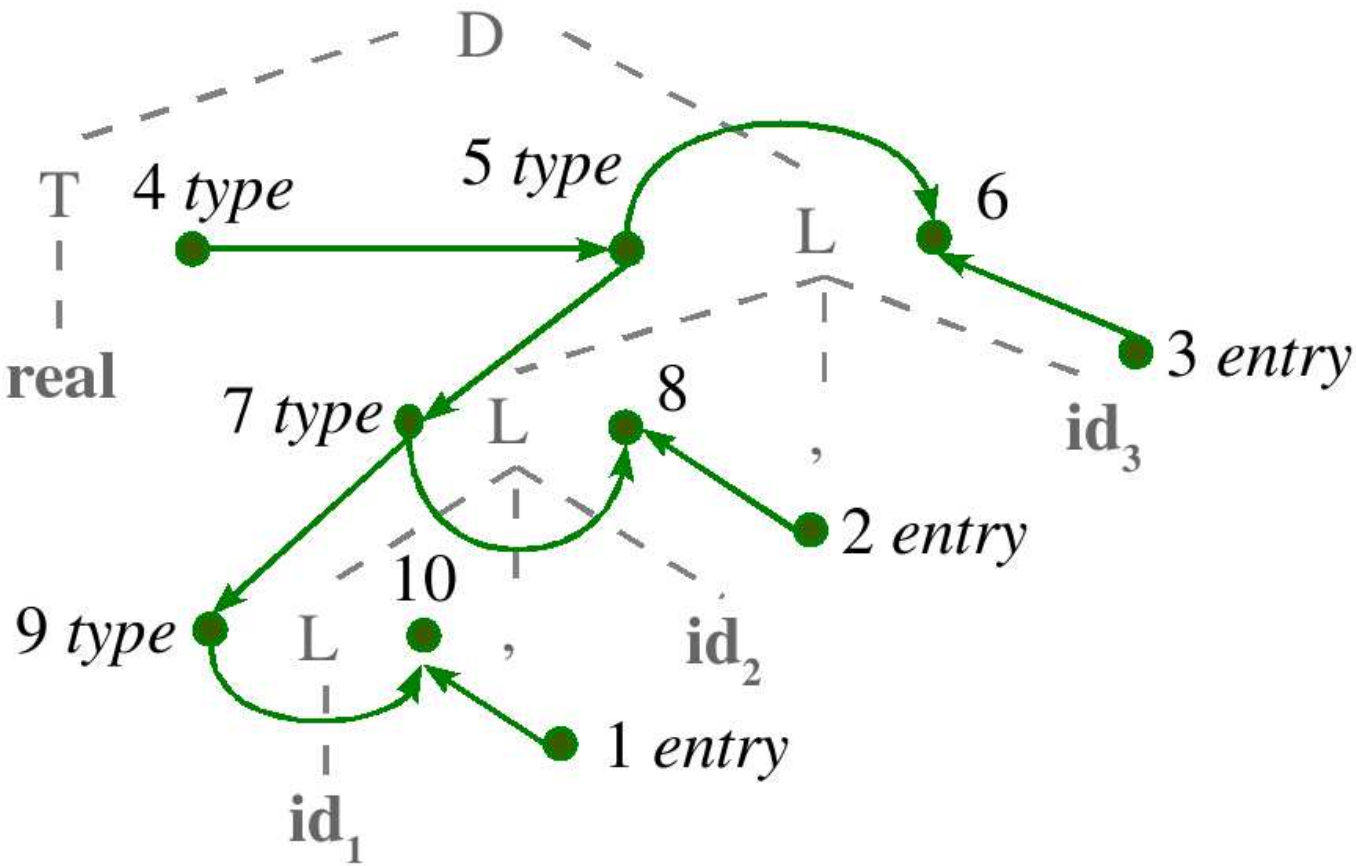
Algoritmo

- Ad esempio, la produzione $E \rightarrow E_1 + T$ con regola semantica E.val = E_1.val $\oplus$ T.val da origine al seguente frammento di grafo





Esempio


D
T
4 type
5 type
L
6
real
7 type
L
8
3 entry
id3
,
2 entry
10
9 type
L
,
id2
1 entry
id1

Ordine di valutazione

- Un **ordinamento topologico** di un grafo diretto aciclico è un qualsiasi ordinamento dei nodi $n_1 \preceq n_2 \preceq \ldots \preceq n_k$ tale che se esiste un arco nel grafo dal nodo n_i al nodo n_j (se la coppia $(n_i, n_j) \in A$) allora n_i precedete n_j nell'ordinamento ($n_i \preceq n_j$) ogni coppia i, j
- Nell'esempio precedente l'ordinamento definito dai numeri associati ai nodi $1 \preceq 2 \preceq \ldots \preceq 10$ è un ordinamento topologico: se esiste un arco dal nodo i al nodo j allora $i \preceq j$
- Un qualsiasi ordinamento topologico del grafo delle dipendenze dà un ordine valido in cui le regole semantiche possono essere valutate
- Nell'ordinamento topologico i valori degli attributi $c_1, c_2, \ldots, c_k$ di una regola $b := f(c_1, c_2, \ldots, c_k)$ sono disponibili sempre prima che f sia valutata

Ordine di valutazione

La traduzione specificata da una qualsiasi definizione guidata dalla sintassi può essere sempre e comunque implementata nel seguente modo:

1. si costruisce il parse tree
2. si costruisce il grafo delle dipendenze
3. si trova un ordinamento topologico del grafo
4. si valutano le regole semantiche dei nodi secondo l'ordinamento

Ordine di valutazione

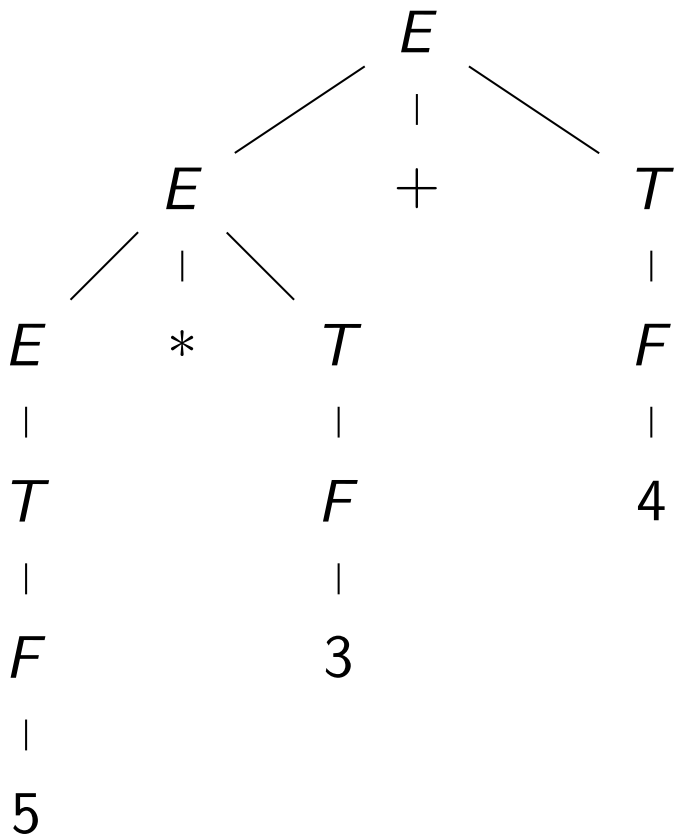
Consideriamo di nuovo il solito esempio, la valutazione della dichiarazione real d_1, d_2, d_3. Abbiamo: (1) costruito il parse tree dalla stringa, (2) costruito il grafo delle dipendenze e (3) identificato un ordinamento topologico del grafo. Non ci resta che valutare le regole semantiche in base all'ordinamento, ottenendo il seguente frammento di codice:

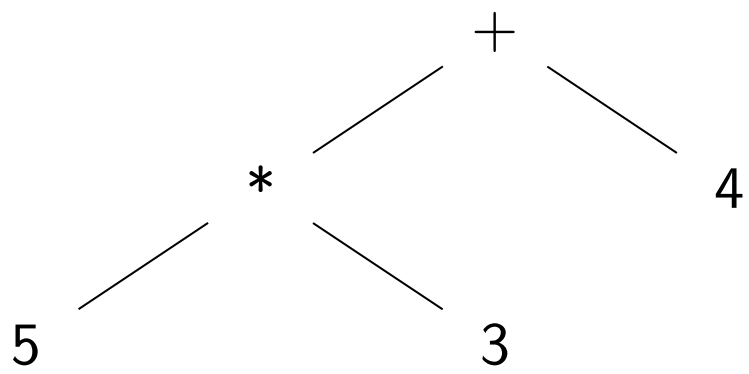
$$
\begin{aligned}
&a_4 := real;\\
&a_5 := a_4;\\
&addtype(id_3.entry, a_5);\\
&a_7 := a_5;\\
&addtype(id_2.entry, a_7);\\
&a_9 := a_7;\\
&addtype(id_1.entry, a_9);
\end{aligned}
$$

Syntax Tree

- Vediamo ora come sia possibile usare le definizioni guidate dalla sintassi per specificare (implementare) la costruzione di syntax tree
- Abbiamo parlato di syntax tree all'inizio del corso: un albero sintattico (astratto) è una forma condensata di un parse tree che è utile per rappresentare i costrutti dei linguaggi
- Operatori e le parole chiave non appaiono come foglie, ma sono associati a nodi interni; sulle foglie troviamo invece gli operandi
- Un'altra semplificazione è che le catene di applicazione di una singola produzione vengono collassate

Syntax Tree





Alcuni esempi di Syntax Tree

- La traduzione guidata dalla sintassi potrebbe essere basata su alberi sintattici piuttosto che su parse tree
- L'approccio è sempre lo stesso: associamo degli attributi ai nodi dell'albero

Costruzione di Syntax Tree

- Vediamo ora in dettaglio come costruire gli alberi sintattici per le espressioni aritmetiche; la procedura è chiaramente ricorsiva
- Costruiamo, inanzitutto, i sottoalberi per le sottoespressioni creando un nodo per ogni operatore ed operando
- I figli di un nodo operatore altro non sono che le radici dei sottoalberi che rappresentano le sottoespressioni che definiscono lespressione principale
- Ogni nodo di un syntree può essere implementato mediante un record con diversi campi

Costruzione di Syntax Tree

- In un nodo operatore un campo identifica l'operatore stesso e i campi rimanenti son i puntatori ai nodi operandi; l'operatore è spesso detto **etichetta** del nodo
- I nodi in un syntree possono avere campi addizionali per gli attributi che sono stati definiti
- In un nodo operando un campo identifica l'operando che può essere un identificatore o un numero
- I campi rimanenti possono rappresentare un entrata alla symbol table (nel caso in cui l'operando sia un identificatore) o un valore (nel caso in cui l'operando sia un numero)

Costruzione di Syntax Tree

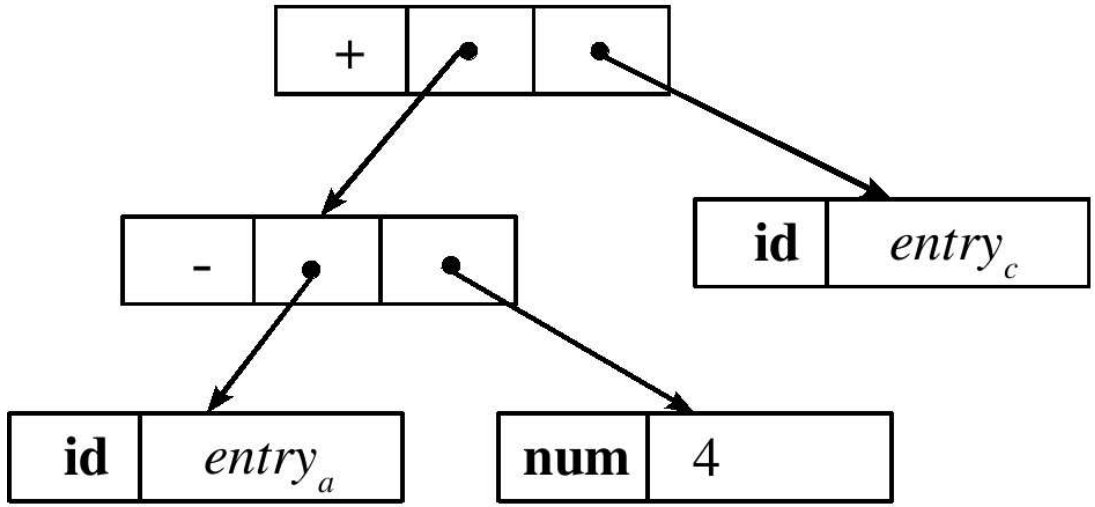
Usiamo le seguenti funzioni per costruire i nodi dei syntree per espressioni con operatori **binari**:

1. *mknode*(*op*, *left*, *right*) crea un nodo operatore con etichetta *op* e due campi puntatore all'operando destro e sinistro
2. *mkleaf* (**id**, *entry*) crea un nodo identificatore con etichetta **id** ed un puntatore *entry* alla tabella dei simboli
3. *mkleaf* (**num**, *val*) crea un nodo numero con etichetta **num** e un campo *val* contentente il valore

Costruzione di Syntax Tree

Il seguente frammento di programma crea (in maniera bottom-up) un syntax tree per lespressione $a - 4 + c$

1. $p_1 = mkleaf(\mathbf{id}, entry_a)$;
2. $p_2 = mkleaf(\mathbf{num}, 4)$;
3. $p_3 = mknode(`-', p_1, p_2)$;
4. $p_4 = mkleaf(\mathbf{id}, entry_c)$;
5. $p_3 = mknode(`+', p_3, p_5)$;



Usiamo una definizione

- Diamo una definizione guidata dalla sintassi S-attributed per la costruzione dell'albero sintattico di una espressione contenente gli operatori + e -
- Introduciamo un attributo *nptr* per ogni simbolo non terminale
- Esso deve tenere traccia dei puntatori ritornati dalle funzioni di creazione dei nodi

Produzioni	Regole Semantiche
$E \rightarrow E_1 + T$	$E.nptr := mknode(`+', E_1.nptr, T.nptr)$
$E \rightarrow E_1 - T$	$E.nptr := mknode(`-', E_1.nptr, T.nptr)$
$E \rightarrow T$	$E.nptr := T.nptr$
$T \rightarrow (E)$	$T.nptr := E.nptr$
$T \rightarrow \mathbf{num}$	$E.nptr := mkleaf(\mathbf{id}, \mathbf{id}.entry)$
$T \rightarrow \mathbf{id}$	$E.nptr := mkleaf(\mathbf{num}, \mathbf{num}.val)$

Albero annotato