Traduzione guidata dalla sintassi

Attributi e Definizioni guidate dalla sintassi

Intro

- 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

Attributi

- L'informazione di ogni costrutto è rappresentata dal valore di diversi **attributi** associati al simbolo non terminale della grammatica usato per generare il costrutto
- Il valore di ogni attributo è calcolato tramite **regole semantiche** associate con le produzioni della grammatica

Due notazioni a diversi livelli

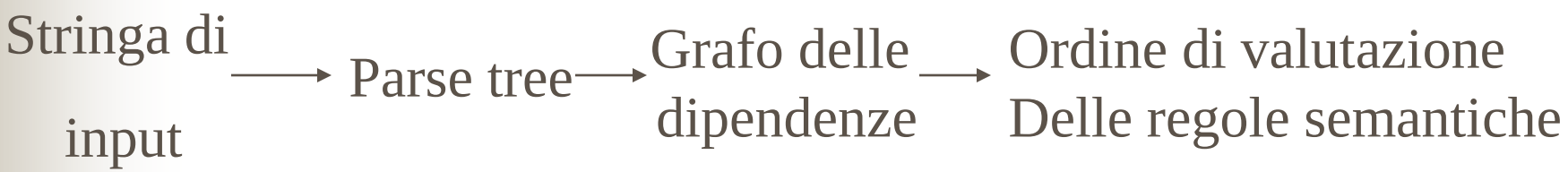
- Ci sono due diversi tipi di notazione per scrivere le regole semantiche:
  1. Definizioni guidate dalla sintassi
  2. Schemi di traduzione
- Le prime sono specifiche di alto livello: nascondono i dettagli implementativi e non richiedono di specificare **l'ordine di valutazione** che la traduzione deve seguire

Due notazioni a diversi livelli

- 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

Il flusso concettuale

- Anche per questa fase della compilazione vediamo il flusso concettuale dei dati



Il flusso concettuale

- Dalla stringa di input viene costruito il parse tree e poi l'albero viene attraversato nella maniera adatta (data dal grafo delle dipendenze) per valutare le regole semantiche che si trovano sui nodi
- Come al solito, comunque, una reale implementazione non segue questo flusso concettuale, ma esegue tutto durante il parsing senza costruire il parse tree esplicitamente, né il grafo delle dipendenze

Definizioni guidate dalla sintassi

- Sono generalizzazioni delle grammatiche in cui ad ogni simbolo della grammatica è associato un insieme di attributi
- Ogni insieme di attributi è partizionato in due sottoinsiemi:
  - Gli attributi **sintetizzati**
  - Gli attributi **ereditati**

Definizioni guidate dalla sintassi

- Possiamo pensare ad ogni nodo del parse tree come ad un record i cui campi sono i nomi degli attributi
- Ogni attributo può rappresentare qualunque cosa vogliamo: 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 degli attributi sintetizzati ad ogni nodo *n* è calcolato a partire dai valori degli attributi dei **nodi figli** di *n* nel parse tree
- Il valore degli attributi erediati è ad ogni nodo *n* è calcolato a partire dai valori degli attributi dei nodi fratelli e del nodo padre di *n*
- Le regole semantiche inducono delle dipendenze tra il valore degli attributi che possono essere rappresentate con dei grafi (delle dipendenze)

Regole semantiche

- La valutazione, nell'ordine giusto, 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 veriabile globale

Decorazioni

- Un parse tree che mostri i valori degli attributi ad ogni nodo è chiamato *parse tree annotato*
- Il processo di calcolo di questi valori si dice annotazione o *decorazione del parse tree*

Forma di una definizione

- Ogni produzione $A \rightarrow \alpha$ ha associato un insieme di regole semantiche della forma:

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

- f è una funzione
- $c_1, \ldots, c_k$ sono attributi dei simboli di α
- b è
  - Un attributo sintetizzato di A
  - Un attributo ereditato di uno dei simboli di α o di A

Forma di una definizione

- In ogni caso diciamo che b dipende dagli attributi $c_1,\ldots, c_k$
- Una grammatica di attributi è una definizione guidata dalla sintassi in cui le funzioni delle regole semantiche non hanno side-effects
- Le funzioni sono in genere scritte come espressioni
- I side-effects sono espressi usando chiamate di procedura o frammenti di codice

Esempio

PRODUZIONE	REGOLE SEMANTICHE
$L \rightarrow E$ **n**	print(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
$F \rightarrow$ **digit**	F.val := **digit**.*lexval*

Esempio: spiegazioni

- 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 digit ha un attributo sintetizzato il cui valore si assume essere fornito dall'analizzatore lessicale
- La regola associata al simbolo iniziale L è una procedura che stampa un valore intero (side-effect) mentre tutte le altre regole servono per il calcolo del valore degli attributi

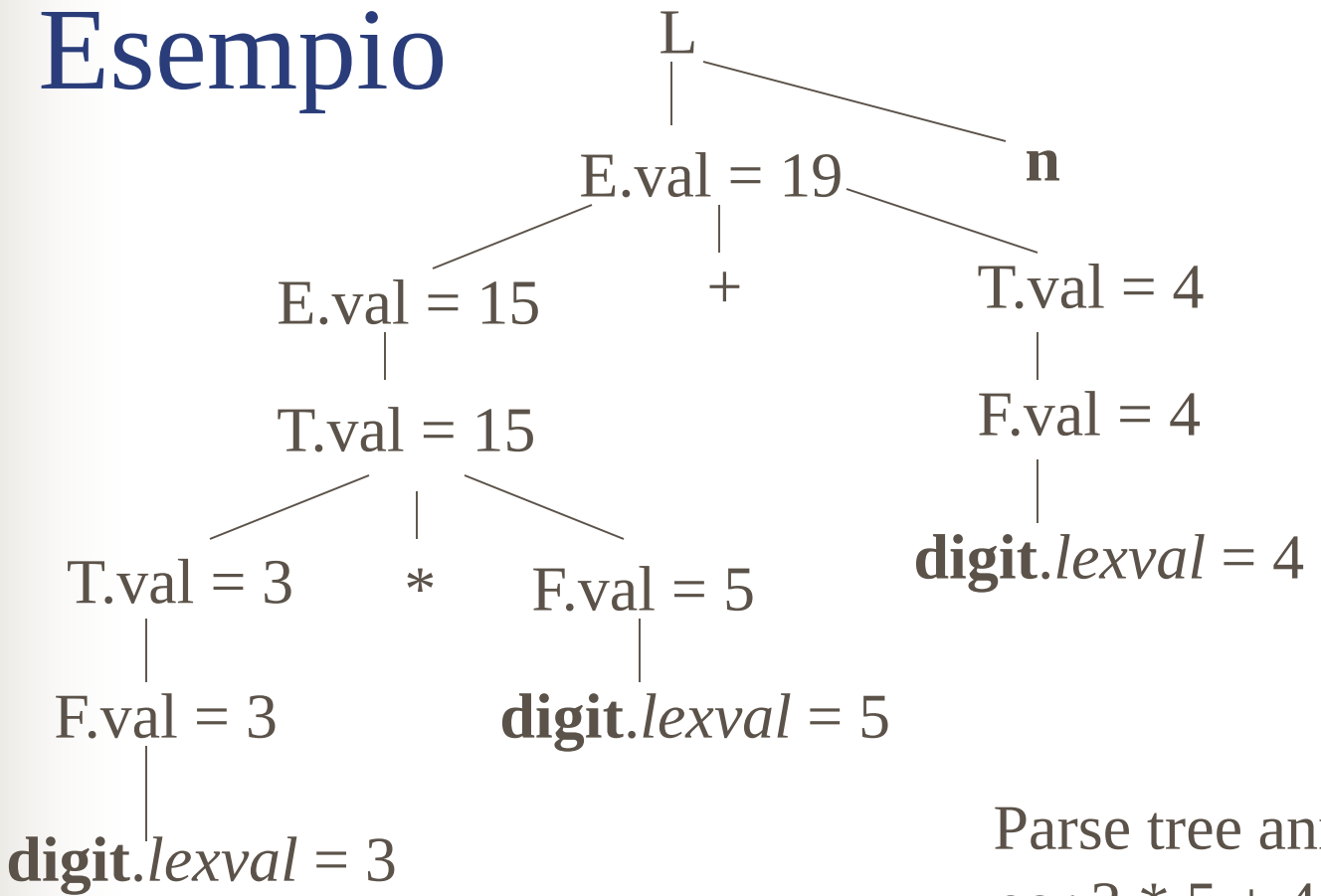
Assunzioni e convenzioni

- In una definizione guidata dalla sintassi si assume che i simboli terminali abbiano solo attributi sintetizzati
- I valori per questi attributi sono in genere forniti dall'analizzatore lessicale
- Il simbolo iniziale, se non diversamente specificato, non ha attributi ereditati

Definizioni S-attributed

- Nella pratica gli attributi sintetizzati sono i più usati
- Una definizione che usi solo attributi sintetizzati è chiamata S-attributed
- Un parse tree di una def. S-attributed può sempre essere annotato valutando le regole semantiche per gli attributi ad ogni nodo in maniera bottom-up dalle foglie alla radice
- Possono quindi essere implementate facilmente durante il parsing LR

Esempio



Parse tree annotato per 3 * 5 + 4 **n**

Esempio: valutazione

- Consideriamo il nodo interno più in basso e più a sinistra per cui è usata la produzione F $\rightarrow$ **digit**
- La corrispondente regola semantica F.val := **digit**.*lexval* pone in questo nodo l'attributo val per F a 3 poiché il valore dell'attributo *lexval* del nodo figlio **digit** è uguale a 3
- Allo stesso modo nel nodo padre il valore di T.val è 3 poiché si ha la regola semantica T.val := F.val

Esempio: valutazione

- Consideriamo il nodo con la produzione $T \rightarrow T * F$
- La regola semantica è $T.val := T_1.val \otimes F.val$
- L'operatore $\otimes$ è il corrispondente semantico dell'operatore sintattico * : è una implementazione della moltiplicazione fra interi
- $T_1.val$ è il valore dell'attributo val del primo figlio (più a sinistra) T, cioè 3

Attributi ereditati

- Ricordiamo che un attributo è ereditato se il suo valore dipende da quelli associati al padre e/o ai fratelli
- Sono utili per esprimere le dipendenze di un costrutto di un linguaggio di programmazione rispetto al suo contesto
- Ad esempio possiamo usare un attributo ereditato per tenere traccia del fatto che un certo identificatore compare a sinistra o a destra di un assegnamento (nel primo caso ci serve il suo indirizzo, nel secondo il suo valore)

Attributi ereditati

- È importante sapere, comunque, che è sempre possibile riscrivere una definizione guidata dalla sintassi in modo da avere solo attributi sintetizzati
- Tuttavia scrivere definizioni usando attributi ereditati risulta essere molto più naturale
- Vediamo un esempio in cui un attributo ereditato è usato per distribuire l'informazione sul tipo fra i vari identificatori di una dichiarazione.

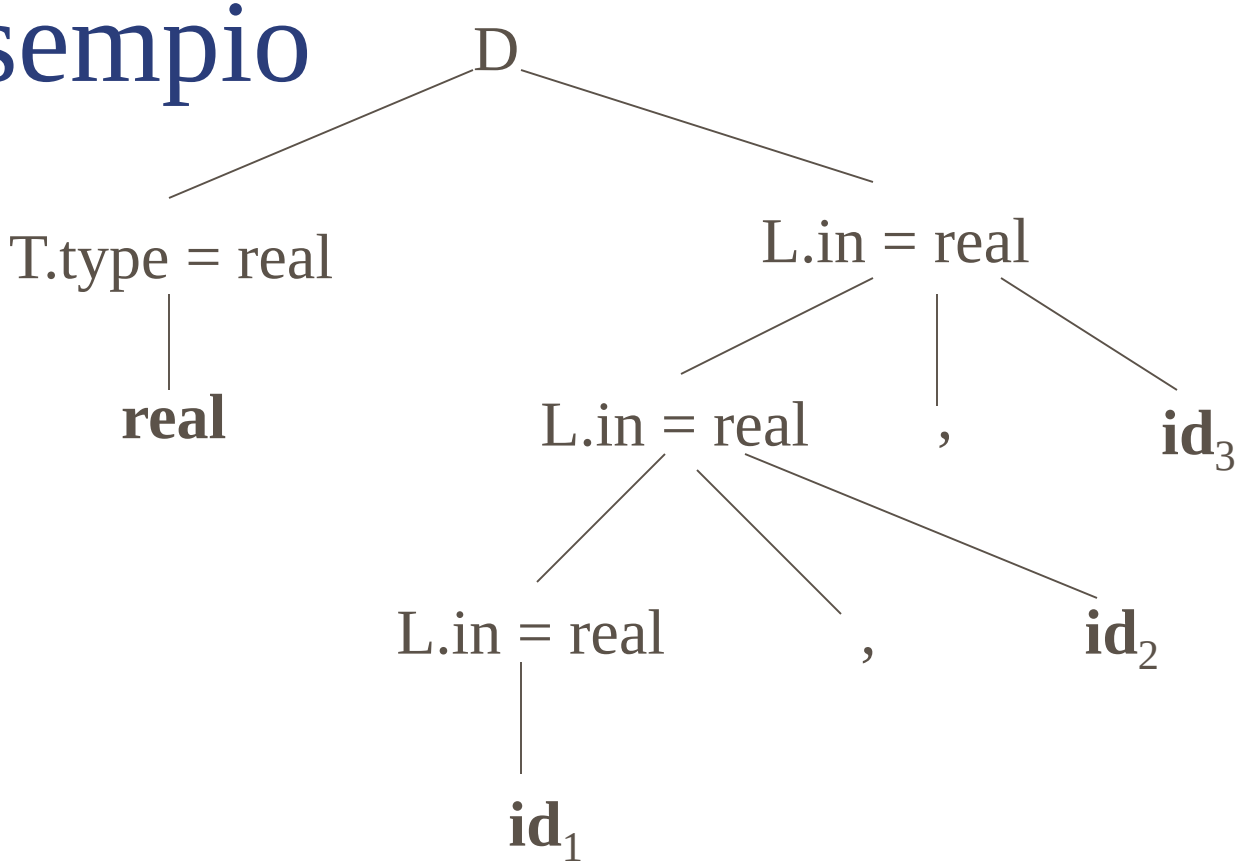
Esempio

PRODUZIONI	REGOLE SEMANTICHE
D → T L	L.in := T.type
T → **int**	T.type := integer
T → **real**	T.type := real
L → L1 , **id**	L1.in := L.in addtype(**id**.entry, L.in)
L → **id**	addtype(**id**.entry, L.in)

Esempio: spiegazioni

- Una dichiarazione D è costituita (ad esempio in C, o in Java) dal nome del tipo T seguito da una lista L di identificatori
- in è un attributo ereditato di L
- All'inizio il valore è passato ad L da T
- Durante la costruzione della lista ogni elemento passa al successivo il valore in
- La procedura addtype semplicemente riempie la tabella dei simboli, all'entrata per l'identificatore che si sta trattando, con l'informazione sul tipo

Esempio



Parse tree annotato per **real** $\mathbf{id}_1$, $\mathbf{id}_2$, $\mathbf{id}_3$

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

- Prima di tutto rendiamo uniformi le regole semantiche ponendole tutte nella forma $b := f(c_1,c_2,...,c_k)$
- Per le chiamate di procedure introduciamo un attributo fittizio
- Il grafo ha un nodo per ogni attributo e un arco da c a b se b dipende dall'attributo c

Algoritmo

for each nodo *n* nel parse tree **do**

    for each attributo *a* del simbolo in *n* do

        costruisci un nodo per *a* nel grafo;

for each nodo *n* nel parse tree **do**

    for each regola semantica b := $f(c_1,c_2,...,c_k)$

        associata con la produzione usata in *n* **do**

        for i := 1 to *k* **do**

            costruisci un arco dal nodo per c_i al nodo per b

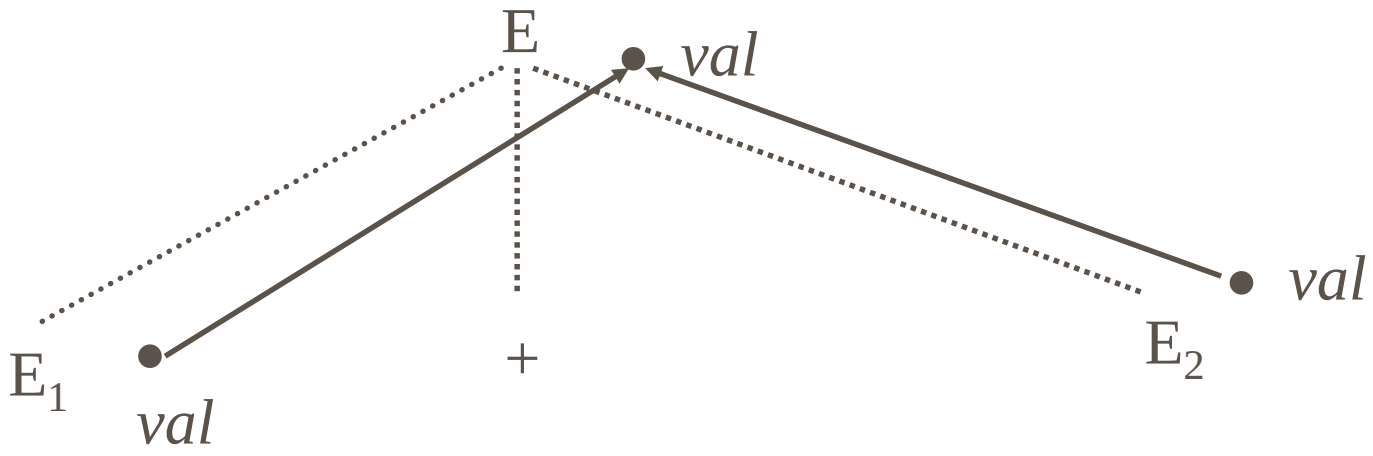
Algoritmo

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

Algoritmo

- $A \rightarrow XY$ con regola semantica X.i := g(A.a, Y.y)
- L'attributo ereditato i ha, per il suo nodo corrispondente, due archi entranti: uno da a e uno da y
- Esempio: $E \rightarrow E_1 + E_2$ con E.val := E_1.val $\oplus$ E_2.val

Esempio

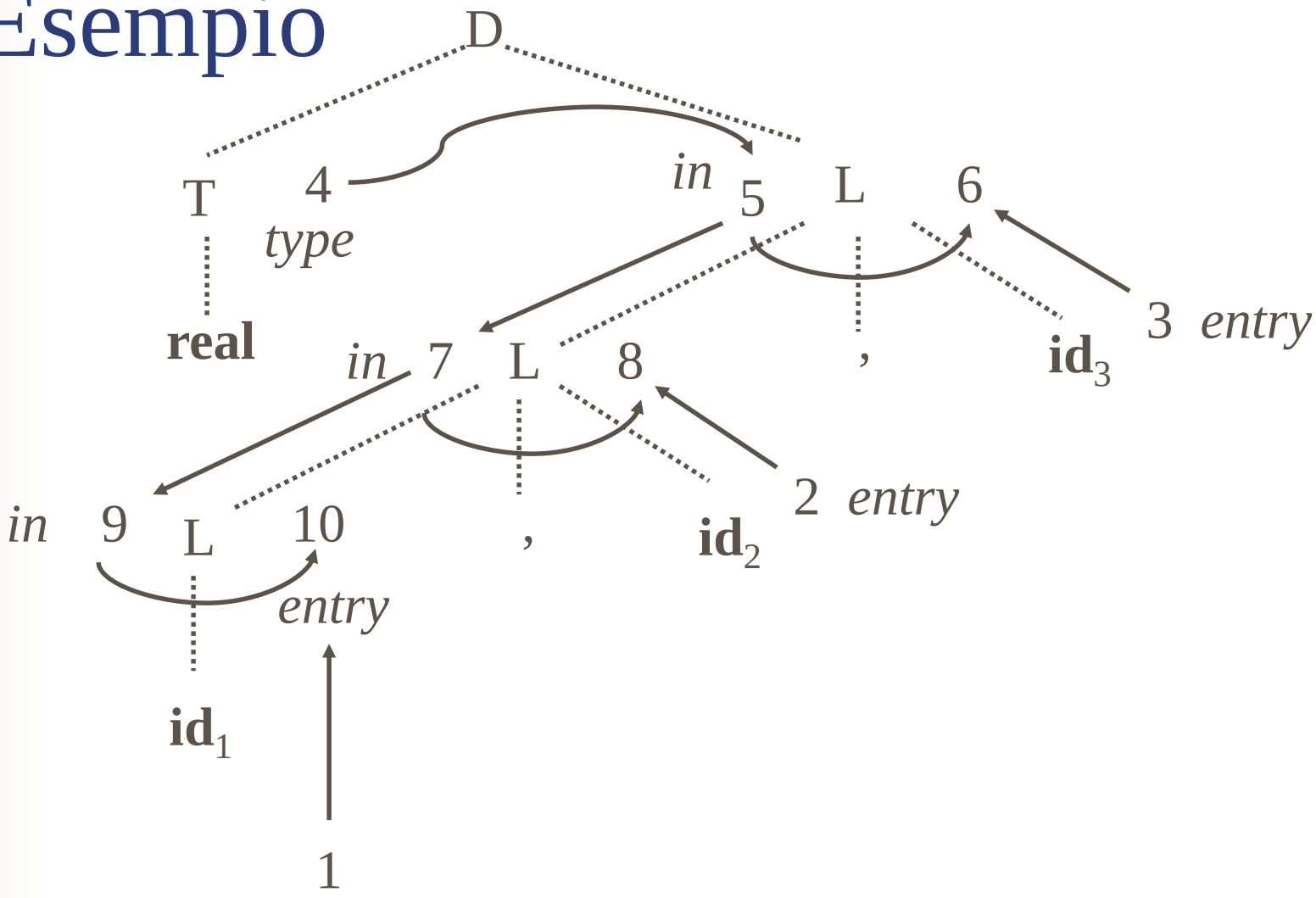


..... Parse tree

——→ Arco del grafo delle dipendenze

• Nodo del grafo delle dipendenze

Esempio


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

Ordine di valutazione

- Un **ordinamento topologico** di un grafo diretto aciclico è un qualsiasi ordinamento dei nodi m_1, m_2,..., m_k tale che gli archi vanno da nodi che vengono prima nell'ordinamento a nodi che vengono dopo
- In altre parole: se $m_i \rightarrow m_j$ nel grafo allora m_i viene prima di m_j nell'ordinamento, per ogni coppia i,j

Ordine di valutazione

- 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 $c_1,c_2,...,c_k$ di una regola b := $f(c_1,c_2,...,c_k)$ sono disponibili sempre ai vari nodi *n* 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 dalla grammatica
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

- I numeri dell'esempio sulle dichiarazioni sono un ordinamento topologico
- Se scriviamo in ordine le regole semantiche otteniamo il seguente programma:

a4 := real;

a5:= a4;

addtype(**id**$_3$.entry, a5);

a7:= a5;

addtype(**id**$_2$.entry, a7);

a9:= a7;

addtype(**id**$_1$.entry, a9);

• La valutazione degli attributi sintetizzati dei simboli terminali non viene considerata

• Infatti questi valori sono già disponibili dall'analisi lessicale

Costruzione di alberi sintattici

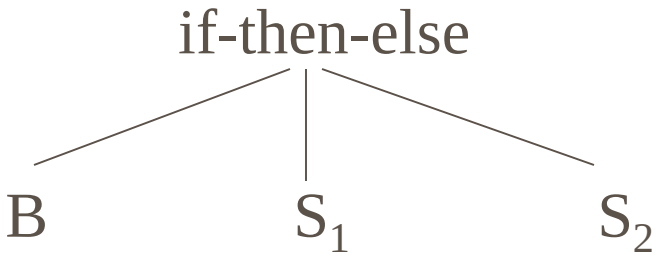
- Vediamo come utilizzare le definizioni guidate dalla sintassi per specificare la costruzione degli alberi sintattici
- L'uso degli alberi sintattici come rappresentazione intermedia divide il problema del parsing da quello della traduzione
- Infatti le routine di traduzione che vengono invocate durante il parsing hanno delle limitazioni

Limitazioni

1. Una grammatica che sia adatta per il parsing potrebbe non riflettere la naturale struttura gerarchica dei costrutti del linguaggio
2. Il metodo di parsing vincola l'ordine in cui i nodi del parse tree sono considerati e questo ordine può non corrispondere con quello in cui l'informazione sui costrutti diventa disponibile

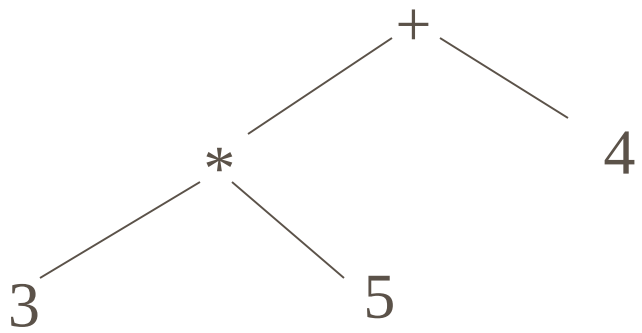
Syntax trees

- Un albero sintattico (astratto) è una forma condensata di un parse tree che è utile per rappresentare i costrutti dei linguaggi
- Ad esempio, la produzione S $\rightarrow$ **if** B **then** S_1 **else** S_2 potrebbe apparire in un albero sintattico come:



Syntax trees

- Negli alberi sintattici gli operatori e le parole chiave non appaiono come foglie, ma sono associati ad un nodo interno
- Inoltre un'altra semplificazione è che le catene di applicazione di una singola produzione vengono collassate:



Syntax trees

- La traduzione guidata dalla sintassi può benissimo 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 (syntree)

- Vediamo come costruire gli alberi sintattici per le espressioni aritmetiche:
- Costruiamo sottoalberi per le sottoespressioni creando un nodo per ogni operatore ed operando
- I figli di un nodo operatore sono le radici dei sottoalberi che rappresentano le sottoespressioni con le quali è costruita l'espressione principale
- Ogni nodo di un syntree può essere implementato come un record con diversi campi

Costruzione di un syntree

- In un nodo operatore un campo identifica l'operatore stesso e i campi rimanenti son i puntatori ai nodi operandi
- L'operatore è spesso chiamato *l'etichetta* del nodo
- Quando vengono usati per la traduzione, i nodi in un syntree possono avere campi addizionali per gli attributi che sono stati definiti

Costruzione di un syntree

- In questo esempio usiamo le seguenti funzioni per costruire i nodi degli alberi sintattici per espressioni con operatori *binari*:

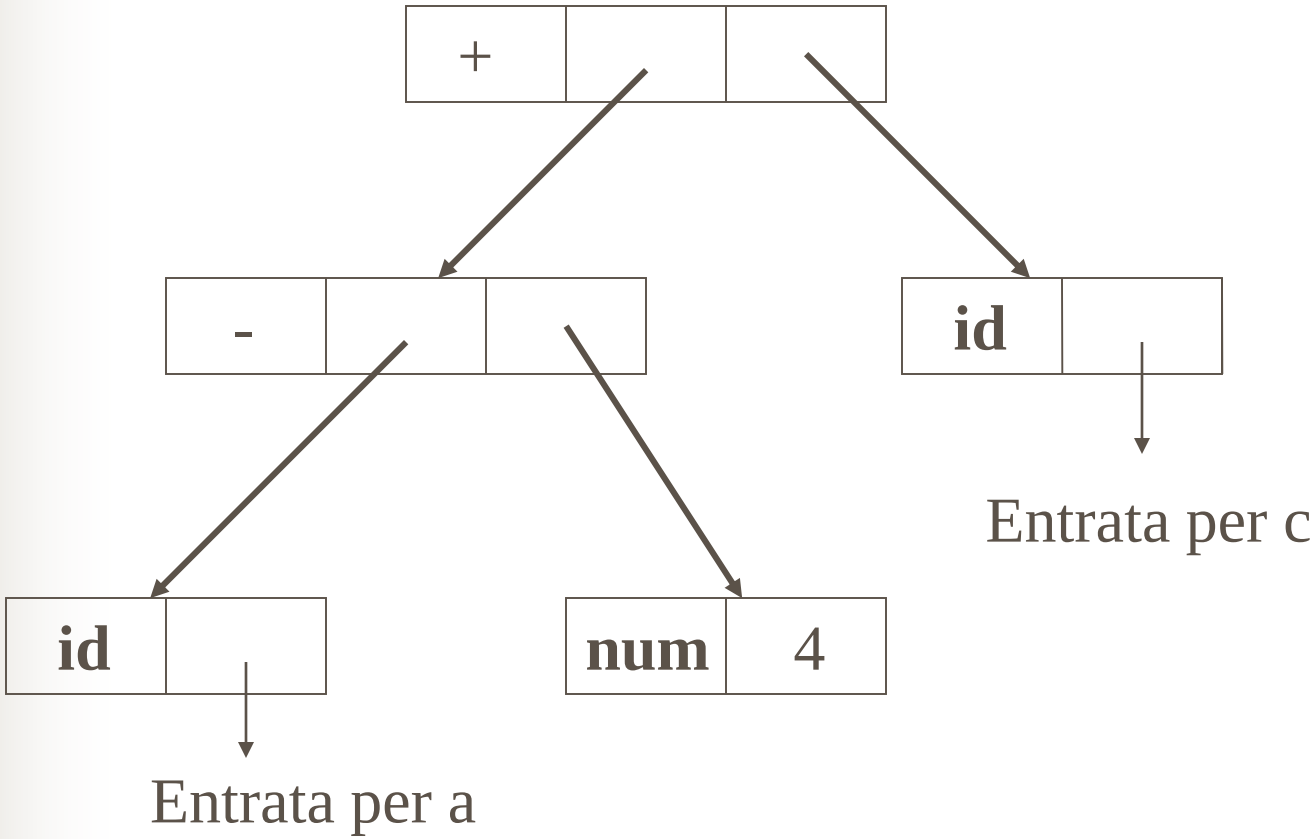
1. *mknode*(*op*, *left*, *right*) crea un nodo operatore con etichetta *op* e due campi puntatore all'operando destro e sinistro
2. *mkleaf*(**id**, *entry*) crea un nodo identificatore con etichetta **id** e puntatore *entry* alla tabella dei simboli
3. *mkleaf*(**num**,*val*) crea un nodo numero con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

Costruzione di un syntree

- Ad esempio il seguente frammento di programma crea (bottom-up) un syntax tree per l'espresione a – 4 + c

1) p_1 := *mkleaf*(**id**, *entrya*);
2) p_2 := *mkleaf*(**num**, 4);
3) p_3 := *mknode*('-', p_1, p_2);
4) p_4 := *mkleaf*(**id**, *entryc*);
5) p_5 := *mknode*('+', p_3, p_4);

Syntree per a – 4 + c



Usiamo una definizione

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

Syntree

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*
$T \rightarrow$ **id**	T.*nptr* := *mkleaf*(**id**, **id**.*entry*)
$T \rightarrow$ **num**	T.*nptr* := *mkleaf*(**num**, **num**.*val*)

L’albero annotato