Linguaggi formali e compilazione

Corso di Laurea in Informatica

A.A. 2012/2013

Linguaggi formali e compilazione

Analisi sintattica (parte II)

- Alberi di derivazione (parse tree) e ambiguità
- Grammatiche di tipo 1 e di tipo 3

Equivalenza di grammatiche

- In Informatica (e non solo, naturalmente) esistono sempre molti modi per risolvere un problema, alcuni più efficienti altri meno.
- Per definire un linguaggio, ad esempio, si possono usare grammatiche equivalenti (cioè che generano lo stesso insieme di stringhe terminali) anche molto diverse.
- Le seguenti grammatiche definiscono $L_5 = \mathrm{a}^*\mathrm{b}^*$:

$$\begin{array}{lcl} S & \to & \epsilon \mid A \\ A & \to & \mathrm{a} \mid \mathrm{a}A \mid B \\ B & \to & \mathrm{b}B \mid \mathrm{b} \end{array} \qquad \begin{array}{lcl} S & \to & AB \\ A & \to & \epsilon \mid \mathrm{a}A \\ B & \to & \epsilon \mid \mathrm{b}B \end{array}$$

- Le seguenti grammatiche definiscono lo stesso semplice linguaggio per le espressioni aritmetiche:

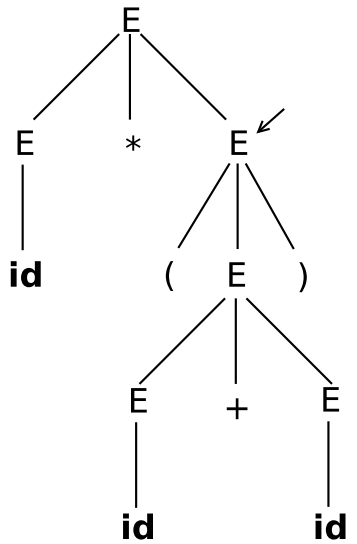
$$\begin{array}{lcl} E & \to & E+E \mid E*E \\ E & \to & (E) \mid \mathtt{id} \end{array} \qquad \begin{array}{lcl} E & \to & E+T \mid T \\ T & \to & T*F \mid F \\ F & \to & \mathtt{id} \mid (E) \end{array}$$

Quale grammatica usare?

- Poiché l'obiettivo è di costruire un riconoscitore (parser), il criterio che deve guidare il progettista è proprio quello di rendere possibile, e agevole, il parsing.
- Fondamentalmente un parser deve "trovare" una derivazione dall'assioma alla stringa in input.
- In generale, se una stringa appartiene al linguaggio, essa può essere derivata in molti modi possibili.
- Tale ridondanza non è un fatto positivo e può avere ripercussioni sull'interpretazione da dare alla stringa (si ricordi che le "stringhe" più importanti per noi sono programmi in un qualche linguaggio di programmazione).
- Cominceremo a lavorare proprio su questo.

Alberi di derivazione (parse tree)

- Ci sono derivazioni diverse che possono essere descritte mediante uno stesso *parse tree*.
- Ad esempio, il seguente parse tree descrive “molte” possibili derivazioni per la stringa **id** $*$ (**id** $+$ **id**) nella grammatica G_1:



- Infatti il parse tree lascia indeterminato l’ordine di applicazione di tutte quelle riscritture che non coinvolgano, nell’albero stesso, nodi interni su uno stesso cammino radice-foglia.

Alberi di derivazione (parse tree)

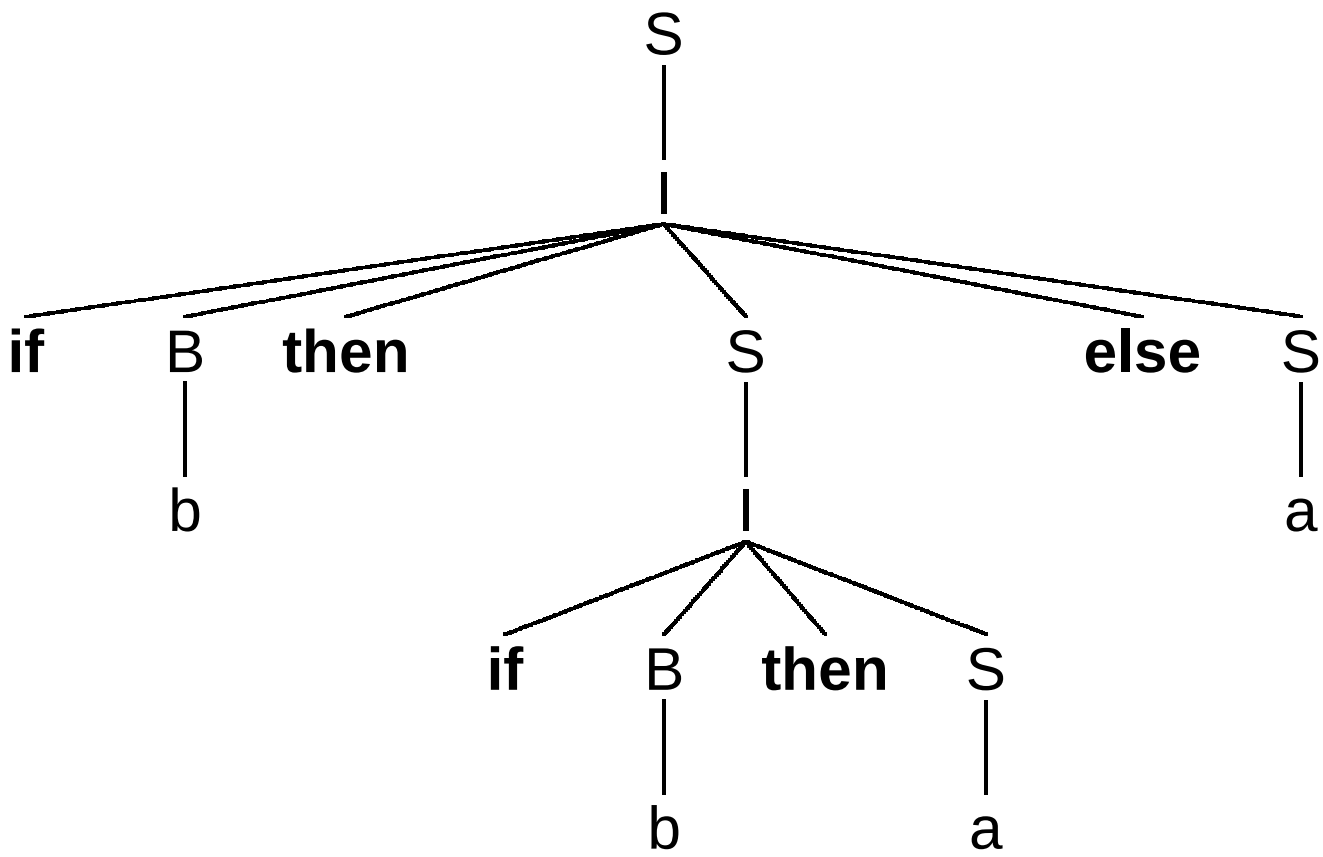
- Formalmente, un parse tree per una grammatica libera $G = (\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$ è un albero radicato ed etichettato che soddisfa le seguenti proprietà:
  - i nodi interni sono etichettati con simboli di $\mathcal{N}$ e, in particolare, la radice è etichettata con l'assioma;
  - le foglie sono etichettate con simboli di $\mathcal{T}$ o con il simbolo ϵ;
  - se $A \in \mathcal{N}$ etichetta un nodo interno e $X_1, \ldots, X_k$ sono le etichette dei figli di (il nodo con etichetta) A, con $X_i \in \mathcal{V}$, allora deve esistere in $\mathcal{P}$ la produzione $A \to X_1 X_2 \ldots X_k$;
  - Se $A \in \mathcal{N}$ etichetta un nodo interno il cui unico figlio è un nodo con etichetta ϵ, allora deve esistere in $\mathcal{P}$ la produzione $A \to \epsilon$.

Un secondo esempio

- Un parse tree per "**if** *b* **then if** *b* **then** *a* **else** *a*" nella grammatica G_I:

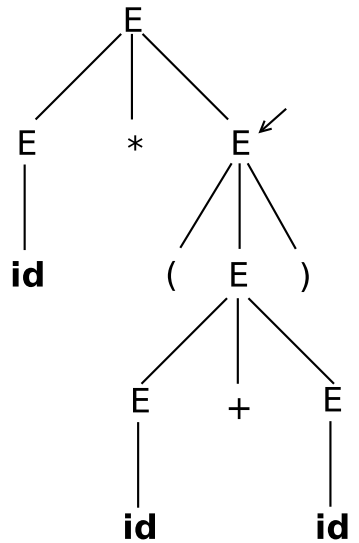
$$S \rightarrow I \mid A, \quad A \rightarrow \mathrm{a}, \quad B \rightarrow \mathrm{b}$$
$$I \rightarrow \textbf{if } B \textbf{ then } S \mid \textbf{if } B \textbf{ then } S \textbf{ else } S$$

è



Alberi di derivazione (parse tree)

- I parse tree sono importanti perché impongono una struttura sintattica (gerarchica) alle frasi e ciò costituisce un primo passo per attribuire ad esse un significato.
- La struttura gerarchica consente di decomporre una frase in sottofrasi, corrispondenti ai sottoalberi e al simbolo non terminale che ne rappresenta la radice.
- Nel parse tree per "**id** $*$ (**id** $+$ **id**)", il sottoalbero indicato con la freccia rappresenta la frase, che potremmo definire "di senso compiuto", (**id** $+$ **id**):



Derivazioni canoniche

- Le derivazioni distinte alle quali corrisponde uno stesso parse tree sono in qualche modo “la stessa derivazione”, proprio perché impongono alla frase derivata la stessa struttura.
- Per tutte queste derivazioni possiamo risolvere il problema della “ridondanza” considerando solo derivazioni cosiddette *canoniche*.
- $\mathcal{S} \Rightarrow_G^* \alpha$ è una *derivazione canonica sinistra/destra* della frase α in G, se ad ogni passo viene riscritto il simbolo non terminale più a sinistra/destra.
- Ad esempio, delle due seguenti derivazioni per **id** $+$ (**id**) in G_1:

  $E \Rightarrow_{G_1} E+E \Rightarrow_{G_1} E+(E) \Rightarrow_{G_1} \texttt{id}+(E) \Rightarrow_{G_1} \texttt{id}+(id)$

  e

  $E \Rightarrow_{G_1} E+E \Rightarrow_{G_1} E+(E) \Rightarrow_{G_1} E+(\texttt{id}) \Rightarrow_{G_1} \texttt{id}+(id)$

  la seconda è una derivazione (canonica) destra, mentre la prima non è una derivazione canonica.

Derivazioni canoniche

- È tuttavia possibile che, per una data frase, esistano non solo derivazioni differenti (circostanza molto probabile) bensì parse-tree differenti e quindi derivazioni (canoniche) destre o sinistre differenti.
- Ad esempio, in G_1 la frase $\mathbf{id} + \mathbf{id} + \mathbf{id}$ può essere derivata così

$$
\begin{aligned}
E &\Rightarrow_{G_1} E+\underline{E} \\
&\Rightarrow_{G_1} E+E+\underline{E} \\
&\Rightarrow_{G_1} E+\underline{E}+\mathbf{id} \\
&\Rightarrow_{G_1} \underline{E}+\mathbf{id}+\mathbf{id} \\
&\Rightarrow_{G_1} \mathbf{id}+\mathbf{id}+\mathbf{id}
\end{aligned}
$$

o così

$$
\begin{aligned}
E &\Rightarrow_{G_1} E+\underline{E} \\
&\Rightarrow_{G_1} \underline{E}+\mathbf{id} \\
&\Rightarrow_{G_1} E+\underline{E}+\mathbf{id} \\
&\Rightarrow_{G_1} \underline{E}+\mathbf{id}+\mathbf{id} \\
&\Rightarrow_{G_1} \mathbf{id}+\mathbf{id}+\mathbf{id}
\end{aligned}
$$

Ambiguità

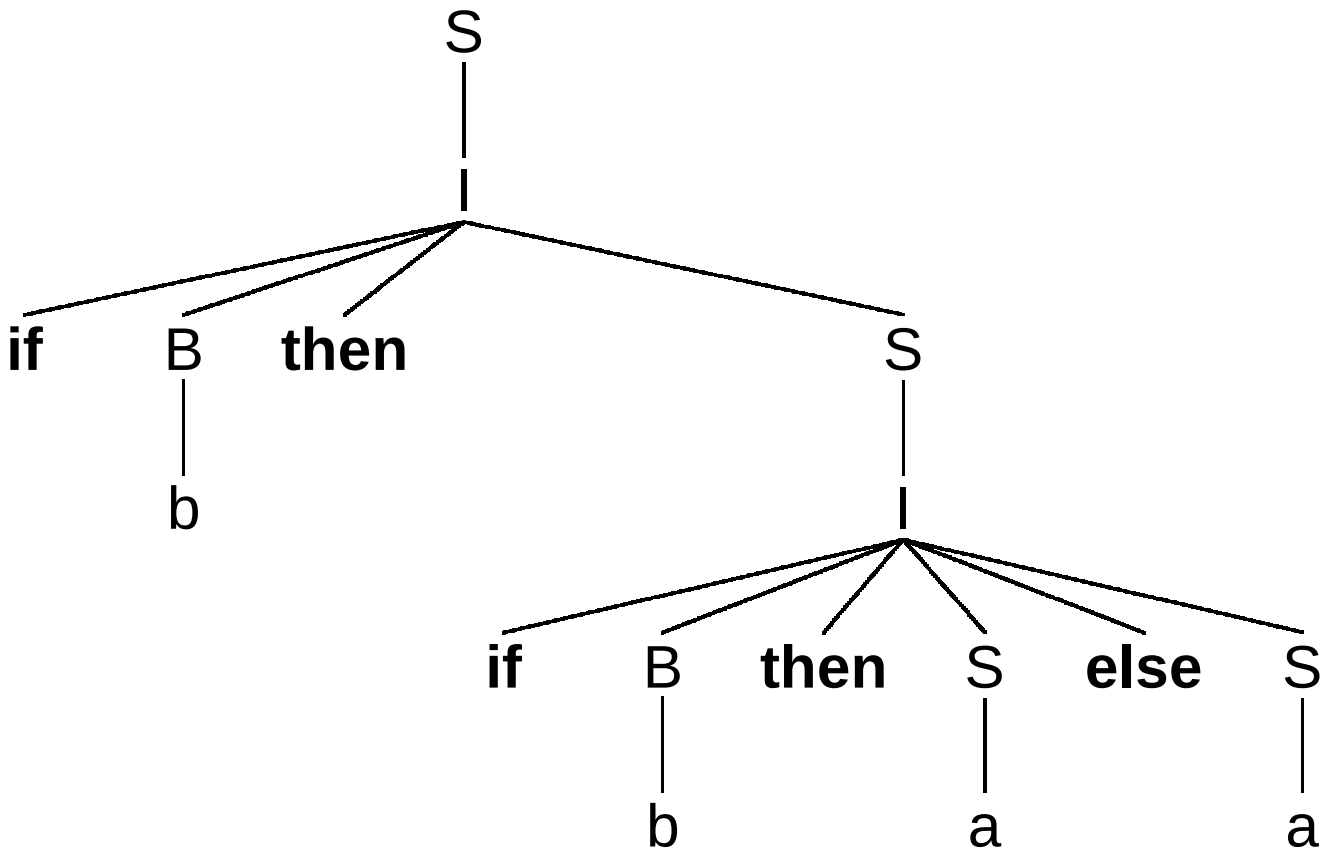
- Il fatto che esistano più derivazioni canoniche per una frase, e quindi parse tree diversi, è un fenomeno che che viene definito con il termine di *ambiguità* della grammatica.
- Si definisce cioè *ambigua* una grammatica in cui almeno una frase ammetta differenti parse tree.
- L'ambiguità è un ostacolo alla realizzazione di un parser e, prima ancora, alla attribuzione di un significato alla frase, visto che alberi diversi impongono strutture sintattiche diverse.
- Vediamo subito due esempi importanti.

Esempio 1

- La grammatica G_I è ambigua in quanto alla frase

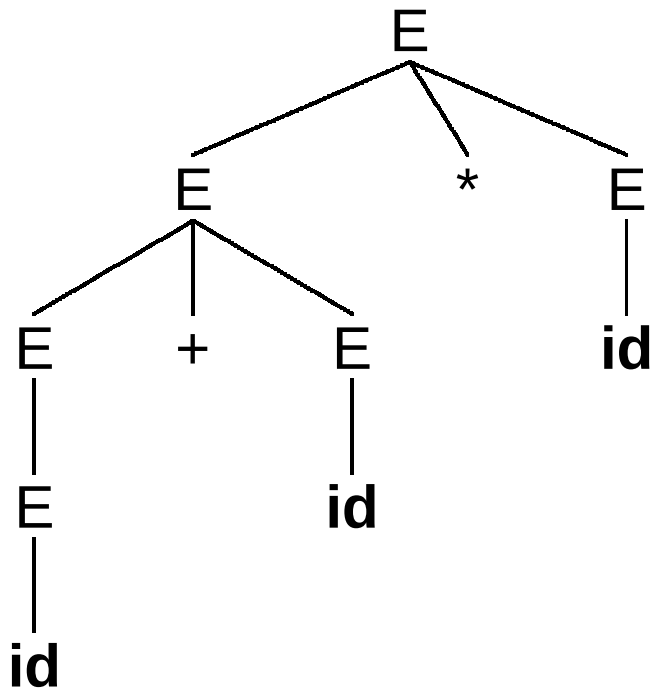
  if b **then if** b **then** a **else** a

  corrisponde un parse tree diverso da quello già visto, e precisamente



Esempio 2

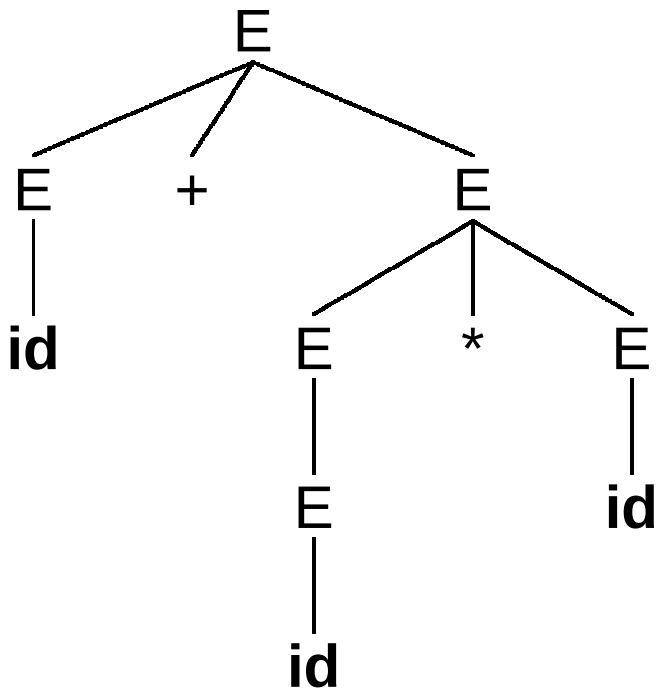
- Alla stringa **id** $+$ **id** $*$ **id** corrisponde in G_1 un primo parse tree illustrato di seguito:



- Si noti come esso suggerisca un'*interpretazione* dell'espressione, e precisamente (**id** $+$ **id**) $*$ **id**, che non coincide con quella che universalmente si dà alla frase in Matematica.

Esempio 2

- Il secondo parse tree che corrisponde a **id** $+$ **id** $*$ **id** in G_1 è:



- Questa volta l'albero suggerisce l'interpretazione "corretta" **id** $+$ (**id** $*$ **id**).

Ambiguità

- Come già accennato, se una grammatica è ambigua il parsing (e poi l'intepretazione) diviene problematico.
- È quindi importante che la grammatica non sia ambigua e, nel caso lo sia, sapere come disambiguare le "frasi ambigue".
- Per certe classi di grammatiche l'eventuale ambiguità può essere stabilita (e rimossa) mediante un algoritmo.
- Si tratta però di un approccio che, quand'anche possibile, risulta macchinoso.
- In certi casi si preferisce utilizzare una grammatica ambigua e implementare una regola di disambiguazione direttamenente nel parser.

Esempio

- Riconsideriamo il frammento di grammatica:

$$\begin{aligned} S &\rightarrow I \mid A \\ I &\rightarrow \textbf{if } B \textbf{ then } S \mid \textbf{if } B \textbf{ then } S \textbf{ else } S \end{aligned}$$

- Questa grammatica è semplice ma, come sappiamo, ambigua per **if** b **then if** b **then** a **else** a.
- Una grammatica non ambigua che "forza" l'interpretazione classica è:

$$\begin{aligned} S &\rightarrow O \mid C \\ C &\rightarrow \textbf{if } B \textbf{ then } C \textbf{ else } C \mid A \\ O &\rightarrow \textbf{if } B \textbf{ then } S \mid \textbf{if } B \textbf{ then } C \textbf{ else } O \end{aligned}$$

- L'unica derivazione possibile è:

$$\begin{aligned} S &\Rightarrow O \Rightarrow \textbf{if } B \textbf{ then } S \\ &\Rightarrow \textbf{if } B \textbf{ then } C \Rightarrow \textbf{if } B \textbf{ then if } B \textbf{ then } C \textbf{ else } C \\ &\overset{*}{\Rightarrow} \textbf{if } b \textbf{ then if } b \textbf{ then } a \textbf{ else } a \end{aligned}$$

Esempio (continua)

- La grammatica modificata è più ostica da usare.
- Si preferisce quindi lasciare la sintassi ambigua e fare in modo che il parser risolva i conflitti (associando ogni `else` all'ultimo `then` incontrato).

Altri aspetti problematici

- Oltre all'ambiguità, esistono altre caratteristiche che, per una data grammatica libera, possono ostacolare il processo di parsing.
- I più importanti sono:
  - presenza di cicli (derivazioni del tipo $A \Rightarrow^+ A$) o comunque di *left recursion* (derivazioni del tipo $A \Rightarrow^+ A\alpha$, con $\alpha \neq \epsilon$);
  - presenza di *prefissi comuni* a due o più produzioni relative allo stesso non terminale, ad esempio $A \to \alpha\beta_1$ e $A \to \alpha\beta_2$.

Esempi

- La grammatica

$$S \rightarrow Sa \mid b$$

presenta una ricorsione immediata a sinistra in quanto $S \rightarrow Sa$.

- La grammatica

$$\begin{aligned} S &\rightarrow A \mid b \\ A &\rightarrow Aa \mid Sa \end{aligned}$$

presenta una ricorsione a sinistra in quanto $S \Rightarrow^{+} Sa$.

- La grammatica G_I (oltre ad essere ambigua) presenta un prefisso comune alla due produzioni che riscrivono il simbolo I.

Esempi

- Nella grammatica

$$\begin{aligned} S &\rightarrow SA \\ A &\rightarrow \mathsf{a} \mid \epsilon \end{aligned}$$

c'è una produzione $A \rightarrow \epsilon$ e questa provoca il ciclo $S \Rightarrow^{+} S$.

Precedenza degli operatori

- Come sappiamo, la grammatica G_1:

$$E \rightarrow E+E \mid E*E \mid \mathbf{id} \mid (E)$$

è ambigua

- In questo caso per eliminare l'ambiguità è sufficiente introdurre un nuovo simbolo non terminale E'

$$\begin{aligned} E &\rightarrow E+E' \mid E*E' \mid E' \\ E' &\rightarrow \mathbf{id} \mid (E) \end{aligned}$$

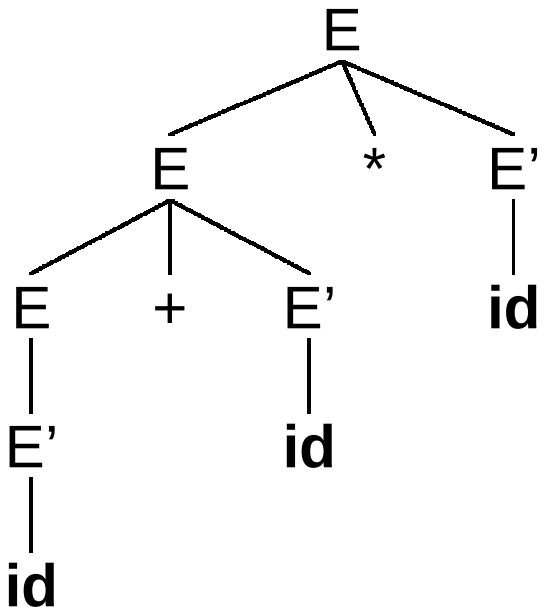
- La nuova grammatica non è più ambigua ma, come anche la stessa grammatica G_1 di partenza, presenta un problema legato alla precedenza degli operatori.

Precedenza degli operatori

- Ad esempio, per la frase **id** $+$ **id** $*$ **id** l'unica derivazione canonica destra possibile è:

$$\begin{aligned} E &\Rightarrow E * E' \Rightarrow E * \mathbf{id} \\ &\Rightarrow E + E' * \mathbf{id} \Rightarrow E + \mathbf{id} * \mathbf{id} \\ &\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id} \end{aligned}$$

alla quale corrisponde l'albero di derivazione "erroneo":



Precedenza degli operatori (continua)

- Le “usuali” precedenze di operatore possono essere forzate introducendo differenti simboli non terminali per i diversi livelli di precedenza.
- Ad esempio, nel caso di somma e prodotto possiamo introdurre due livelli di precedenza, rappresentati dai non terminali E e T, oltre ad un simbolo (useremo F) per la catogoria delle espressioni di base (nel nostro caso identificatori ed espressioni tra parentesi):

$$\begin{aligned} E &\rightarrow E+T \mid T \\ T &\rightarrow T*F \mid F \\ F &\rightarrow \mathbf{id} \mid (E) \end{aligned}$$

Precedenza degli operatori (continua)

- La grammatica precedente (con le ovvie estensioni) “gestisce” senza ambiguità anche il caso degli operatori di divisione e sottrazione, inseriti nelle giuste categorie sintattiche:

$$
\begin{aligned}
E &\rightarrow E+T \mid E-T \mid T \\
T &\rightarrow T*F \mid T/F \mid F \\
F &\rightarrow \mathbf{id} \mid (E)
\end{aligned}
$$

Precedenza degli operatori (continua)

- Volendo inserire anche l'operatore di esponenziazione (che ha precedenza maggiore), è necessario prevedere un'ulteriore variabile sintattica e ricordarsi che l'operatore di esponenziazione (qui useremo il simbolo ^) è associativo a destra:

$$
\begin{aligned}
E &\rightarrow E+T \mid E-T \mid T \\
T &\rightarrow T*P \mid T/P \mid P \\
P &\rightarrow F\hat{}P \mid F \\
F &\rightarrow \mathbf{id} \mid (E)
\end{aligned}
$$

Esercizi

- Si consideri il linguaggio $L_{\mathtt{a}>\mathtt{b}}$ delle stringhe su $\{\mathtt{a}, \mathtt{b}\}$ che contengono più a che b. Si dica, giustificando la risposta, se la seguente grammatica libera genera $L_{\mathtt{a}>\mathtt{b}}$

$$\begin{aligned} S &\rightarrow \mathtt{a} \mid \mathtt{a}S \mid S\mathtt{a} \mid \mathtt{ab}S \mid \mathtt{a}S\mathtt{b} \mid S\mathtt{ab} \mid \\ &\quad \mathtt{ba}S \mid \mathtt{b}S\mathtt{a} \mid S\mathtt{ba} \end{aligned}$$

- Si fornisca una grammatica libera per il linguaggio su $\mathcal{B}$ contenente tutte e sole le stringhe con un diverso numero di `0` e `1`.
- Si dica se la grammatica

$$\begin{aligned} S &\rightarrow \mathtt{a}A \\ A &\rightarrow AB \mid B \\ B &\rightarrow \mathtt{b} \end{aligned}$$

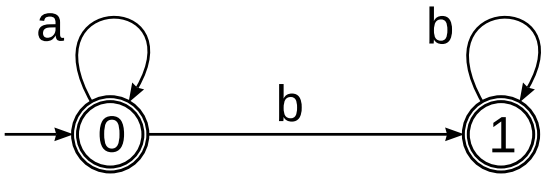
è ambigua.

Automi finiti e grammatiche lineari destre

- Dato un automa a stati finiti M che riconosce il linguaggio L è immediato definire una grammatica lineare destra G_M che genera lo stesso linguaggio, cioè tale che $L(G_M) = L$.
- La costruzione è molto semplice:
  - per ogni stato q dell'automa la grammatica conterrà un simbolo non terminale A_q;
  - l'assioma iniziale è A_{q_0};
  - per ogni transizione $\delta(q, \mathrm{a}) = q'$, la grammatica conterrà la produzione $A_q \to \mathrm{a}A_{q'}$;
  - se q' è uno stato finale, la grammatica conterrà la produzione $A_{q'} \to \epsilon$.

Esempio

- La grammatica corrispondente all'automa M_5:

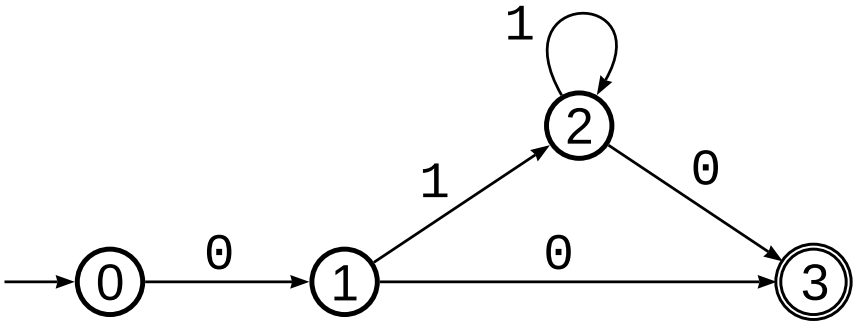


è

$$\begin{aligned} A_{q_0} &\rightarrow \mathrm{a}A_{q_0} \mid \mathrm{b}A_{q_1} \mid \epsilon \\ A_{q_1} &\rightarrow \mathrm{b}A_{q_1} \mid \epsilon \end{aligned}$$

Esempio

- La grammatica corrispondente all'automa M_3:

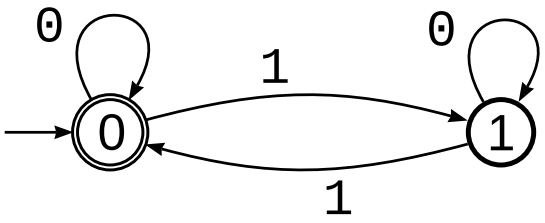


è

$$
\begin{aligned}
A_{q_0} &\rightarrow 0A_{q_1} \\
A_{q_1} &\rightarrow 1A_{q_2} \mid 0A_{q_3} \\
A_{q_2} &\rightarrow 1A_{q_2} \mid 0A_{q_3} \\
A_{q_3} &\rightarrow \epsilon
\end{aligned}
$$

Esempio

- La grammatica corrispondente all'automa M_{parity}:



è

$$A_{q_0} \rightarrow 0A_{q_0} \mid 1A_{q_1} \mid \epsilon$$
$$A_{q_1} \rightarrow 1A_{q_0} \mid 0A_{q_1}$$

Automi finiti e grammatiche lineari destre (continua)

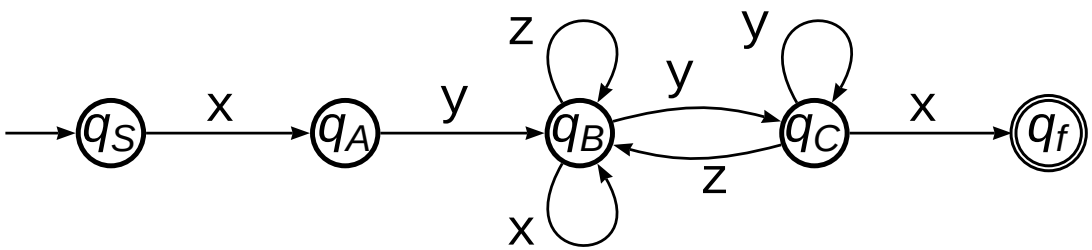
- La costruzione appena vista è completamente reversibile.
- Data una grammatica lineare destra G, possiamo facilmente costruire un automa non deterministico M che riconosce il linguaggio generato da G.
  - per ogni non terminale A della grammatica definiamo uno stato q_A dell'automa;
  - per ogni produzione $A \to \mathrm{a}B$ inseriamo la transizione $\delta(q_A, \mathrm{a}) = q_B$ e per ogni produzione $A \to B$ inseriamo la transizione $\delta(q_A, \epsilon) = q_B$
  - lo stato iniziale è q_S;
  - si definisce uno stato finale q_f, non corrispondente ad alcun simbolo non terminale;
  - per ogni produzione $A \to \mathrm{a}$ si pone $\delta(q_A, \mathrm{a}) = q_f$ e, analogamente, per ogni produzione $A \to \epsilon$ si pone $\delta(q_A, \epsilon) = q_f$.

Esempio

- L'automa corrispondente alla grammatica:

$$
\begin{aligned}
S &\rightarrow \mathrm{x}A \\
A &\rightarrow \mathrm{y}B \\
B &\rightarrow \mathrm{x}B \mid \mathrm{z}B \mid \mathrm{y}C \\
C &\rightarrow \mathrm{x} \mid \mathrm{z}B \mid \mathrm{y}C
\end{aligned}
$$

è

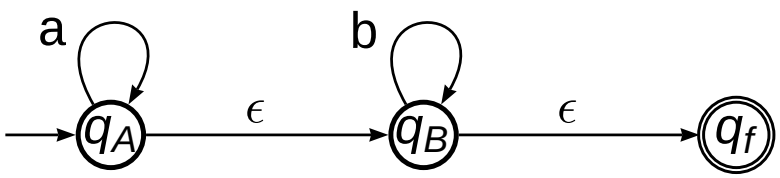


Esempio

- L'automa corrispondente alla grammatica:

$$\begin{aligned} A &\rightarrow \mathrm{a}A \mid B \\ B &\rightarrow \mathrm{b}B \mid \epsilon \end{aligned}$$

è



Grammatiche dipendenti dal contesto

- La forma generale delle produzioni di una grammatica di tipo 1 (dipendente dal contesto) è:

$$\alpha A \gamma \rightarrow \alpha \beta \gamma$$

  dove $\alpha, \beta, \gamma \in \mathcal{V}^*$, $\beta \neq \epsilon$ e $A \in \mathcal{N}$.
- La forma delle produzioni "spiega" il nome di queste grammatiche.
- Infatti, il simbolo *A* può essere riscritto come β solo se appare nel *contesto* rappresentato dalle stringhe α e γ.

Linguaggi dipendenti dal contesto

- Sono così detti i linguaggi generabili da grammatiche dipendenti dal contesto.
- Poiché la forma generale delle produzioni di una grammatica context-dependent sono più generali di quelle di una grammatica context-free, l'insieme dei linguaggi dipendenti dal contesto "contiene" l'insieme dei linguaggi liberi.
- Viceversa, ci sono linguaggi dipendenti dal contesto che non sono liberi.
- Un esempio è il linguaggio

$$L_{13} = \{a^n b^n c^n | n \geq 0\}$$

Costrutti non liberi nei linguaggi di programmazione

- Alcuni aspetti della sintassi dei linguaggi di programmazione non sono catturabili da produzioni di grammatiche libere.
- Ad esempio, il fatto che una variabile debba essere dichiarata prima dell'uso non è esprimibile mediante una grammatica libera.
- Tale caratteristica è catturata dal cosiddetto *linguaggio delle repliche* (che non è libero):

$$L_R = \{\alpha\alpha | \alpha \in \mathcal{T}^*\}.$$

Costrutti non liberi nei linguaggi di programmazione

- Un altro aspetto non esprimibile con grammatiche libere è che il numero e il tipo degli argomenti di una funzione coincida ordinatamente con il numero e il tipo dei parametri formali.
- Anziché utilizzare grammatiche più potenti (che renderebbero difficoltoso, quando non impossibile, il parsing), l'approccio consiste nell'ignorare il problema a livello sintattico.
- La verifica di correttezza viene completata invece durante l'*analisi semantica*