# Grammatiche generative libere da contesto - II

*Prof. A. Morzenti*

*aa 2008-2009*

## ALBERI SINTATTICI E DERIVAZIONI CANONICHE

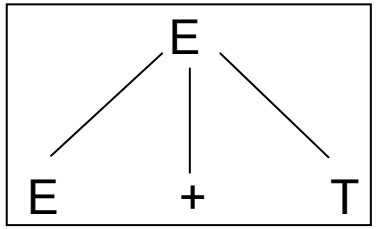
ALBERO SINTATTICO: grafo orientato e ordinato (figli ordinati da sx a dx) privo di cicli, tale che per ogni coppia di nodi esiste un solo cammino che li congiunge

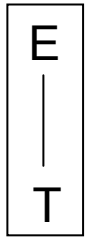
- rappresenta graficamente il processo di derivazione
- relazione padre-figlio / nodo radice / nodi foglia-terminali
- frontiera (seq. foglie lette da sinistra a destra)
- grado di un nodo
- radice contiene assioma S
- frontiera contiene frase generata

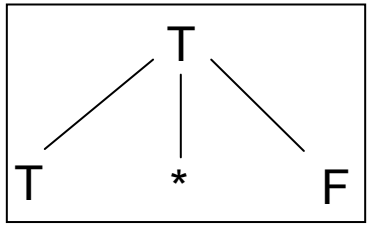
SOTTOALBERO di radice N: albero avente N come radice e comprendente tutti i discendenti di N (i figli di N, i figli dei figli, ecc.)

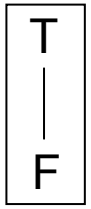
Espressioni con somme e prodotti: E espressione, T termine, F fattore

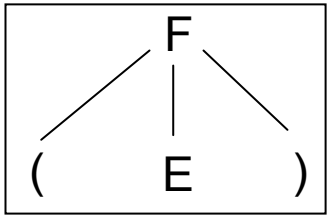
1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow i$

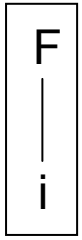


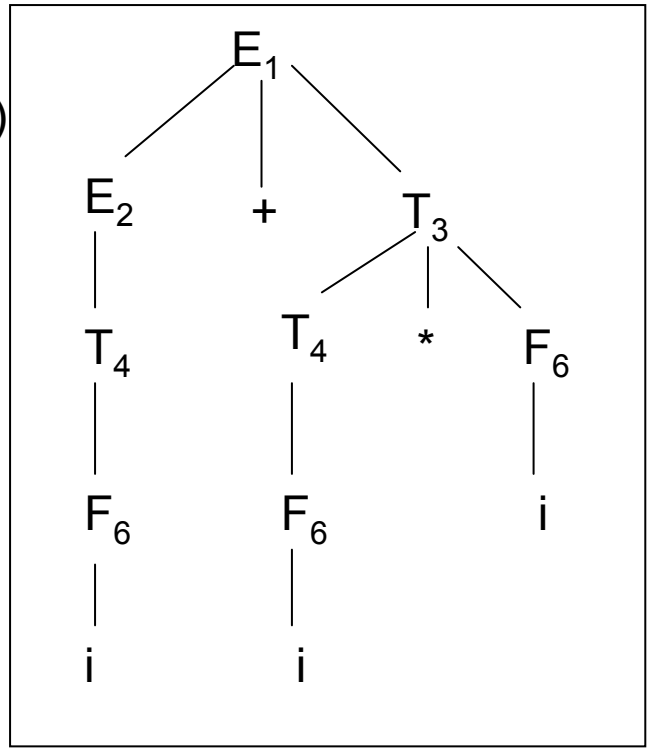



Albero sintattico per la frase i + i * i
(n.t. etichettati con la regola applicata)




Rappresentazione lineare dell'albero
(pedici sono n.t. radici dei sottoalberi)

$$[[[[i]_F]_T]_E + [[[i]_F]_T * [i]_F]_T]_E$$

DERIVAZIONE SINISTRA

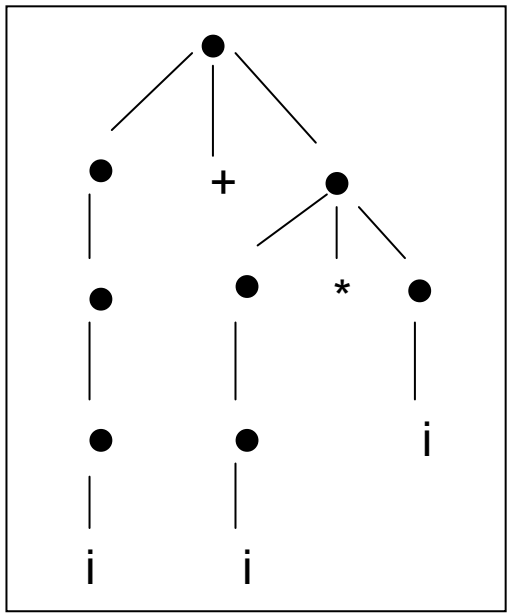
$$E \underset{1}{\Rightarrow} E + T \underset{2}{\Rightarrow} T + T \underset{4}{\Rightarrow} F + T \underset{6}{\Rightarrow} i + T \underset{3}{\Rightarrow} i + T * F \underset{4}{\Rightarrow}$$
$$\underset{4}{\Rightarrow} i + F * F \underset{6}{\Rightarrow} i + i * F \underset{6}{\Rightarrow} i + i * i$$

DERIVAZIONE DESTRA

$$E \underset{1}{\Rightarrow} E + T \underset{3}{\Rightarrow} E + T * F \underset{6}{\Rightarrow} E + T * i \underset{4}{\Rightarrow} E + F * i \underset{6}{\Rightarrow} E + i * i \underset{2}{\Rightarrow}$$
$$\underset{2}{\Rightarrow} T + i * i \underset{4}{\Rightarrow} F + i * i \underset{6}{\Rightarrow} i + i * i$$

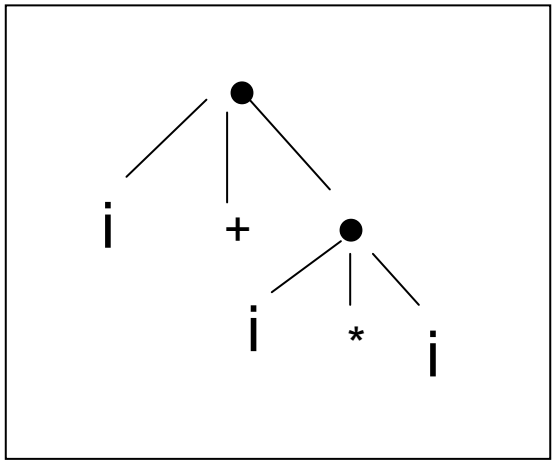
ALBERO SCHELETRICO (solo la frontiera e la struttura)

$$[[[[i]]]+[[[i]]*[i]]]$$




ALBERO SCHELETRICO CONDENSATO (si fondono i nodi interni che stanno su un cammino privo di biforcazioni)

$$[[i]+[[i]*[i]]]$$



# DERIVAZIONI SINISTRE E DESTRE / GRAMMATICHE LIBERE

Derivazione ne` destra ne` sinistra

$$E \underset{s,d}{\Rightarrow} E+T \underset{d}{\Rightarrow} E+T*F \underset{s}{\Rightarrow} T+T*F \Rightarrow T+F*F \underset{d}{\Rightarrow} T+F*i \underset{s}{\Rightarrow}$$
$$\underset{s}{\Rightarrow} F+F*i \underset{d}{\Rightarrow} F+i*i \underset{d}{\Rightarrow} i+i*i$$

OGNI FRASE DI UNA GRAMMATICA LIBERA PUÒ ESSERE GENERATA MEDIANTE UNA DERIVAZIONE SINISTRA (oppure DESTRA) – proprietà importante per gli algoritmi di analisi sintattica. Consente di organizzare nel modo più opportuno l'ordine di costruzione della derivazione di una frase.

NB: in questo caso tutte le derivazioni corrispondono a un unico albero

LINGUAGGI A PARENTESI: nei linguaggi artificiali si incontrano frequentemente strutture annidate (a parentesi) caratterizzate dalla presenza di coppie di elementi che aprono e chiudono un inciso (o stringa subordinata); all'interno di una coppia di elementi possono poi aprirsi e chiudersi altri incisi.

| | |
|---|---|
| Pascal: | begin ...end |
| C: | {…} |
| XML: | <title> … </title> |
| LaTeX: | \begin{equation}…\end{equation} |

Astraendo dalla particolare codifica delle marche, il paradigma dei linguaggi a parentesi è noto come *Linguaggio di Dyck*.

Alfabeto:

$$\Sigma = \{')','(',']','['\}$$

Frasi:

$$()[[()[]]()]$$

Frasi caratterizzate dalla regola di cancellazione: Si applica ripetutamente alla stringa data la cancellazione che sostituisce la stringa vuota a una coppia di parentesi concordi e adiacenti

$$[\ ] \Rightarrow \varepsilon \qquad (\ ) \Rightarrow \varepsilon$$

LINGUAGGIO DI DYCK: parentesi aperte *a,b,...,* parentesi chiuse *a',b',...*
(linguaggio non lineare)

$$\Sigma = \{a, a', b, b'\}$$

$$S \to aSa'S \mid bSb'S \mid \varepsilon$$

$$a\,a\,\underbrace{a\,\underbrace{a'}}\,a'\,a\,a\,\underbrace{a\,\underbrace{a'}}\,a'\,a'\,a'$$

LINGUAGGIO LINEARE NON REGOLARE:

$$L_1 = \{a^n c^n \mid n \geq 1\} = \{ac, aacc, \ldots\}$$

$$S \to aSc \mid ac$$

$L_1$ è un sottoinsieme del Linguaggio di Dyck: non ammette più di un nido di parentesi allo stesso livello (non ammesse stringhe del tipo aababb ).

## COMPOSIZIONE REGOLARE DI LINGUAGGI LIBERI

Applicando le operazioni basilari (unione, concatenamento, stella) a linguaggi liberi, si ottengono ancora linguaggi liberi. *La famiglia dei linguaggi liberi è chiusa rispetto alle operazioni di unione, concatenamento e stella.*

$$G_1 = \left(\Sigma_1, V_{N_1}, P_1, S_1\right) \text{ e } G_2 = \left(\Sigma_2, V_{N_2}, P_2, S_2\right)$$

$$V_{N_1} \bigcap V_{N_2} = \varnothing \qquad S \notin (V_{N_1} \bigcup V_{N_2})$$

NB: necessari ins. n.t. disgiunti e nuovo assioma

UNIONE: $G = (\Sigma_1 \bigcup \Sigma_2, \{S\} \bigcup V_{N_1} \bigcup V_{N_2}, \{S \to S_1 \mid S_2\} \bigcup P_1 \bigcup P_2, S)$

CONCATENAMENTO: $G = (\Sigma_1 \bigcup \Sigma_2, \{S\} \bigcup V_{N_1} \bigcup V_{N_2}, \{S \to S_1 S_2\} \bigcup P_1 \bigcup P_2, S)$

STELLA: G di $(L_1)^*$ è ottenuta aggiungendo a $G_1$ le regole $S \to SS_1 | \varepsilon$

CROCE (non necessario perche` derivato, aggiunto per comodita):
G di $(L1)^+$ è ottenuta aggiungendo a $G_1$ le regole $S \to SS_1 | S_1)$

ESEMPIO: Unione di linguaggi

$$L = \{a^i b^j c^k \mid i = j \vee j = k\} = \{a^i b^i c^* \mid i \geq 0\} \cup \{a^* b^i c^i \mid i \geq 0\} = L_1 \cup L_2$$

$L_1$ ed $L_2$ generabili dalle due grammatiche $G_1$ e $G_2$

genera $a^i b^i c^*$ →

$G_1$

$S_1 \rightarrow XC$

$X \rightarrow aXb \mid \varepsilon$

$C \rightarrow cC \mid \varepsilon$

$G_2$

$S_2 \rightarrow AY$

$Y \rightarrow bYc \mid \varepsilon$

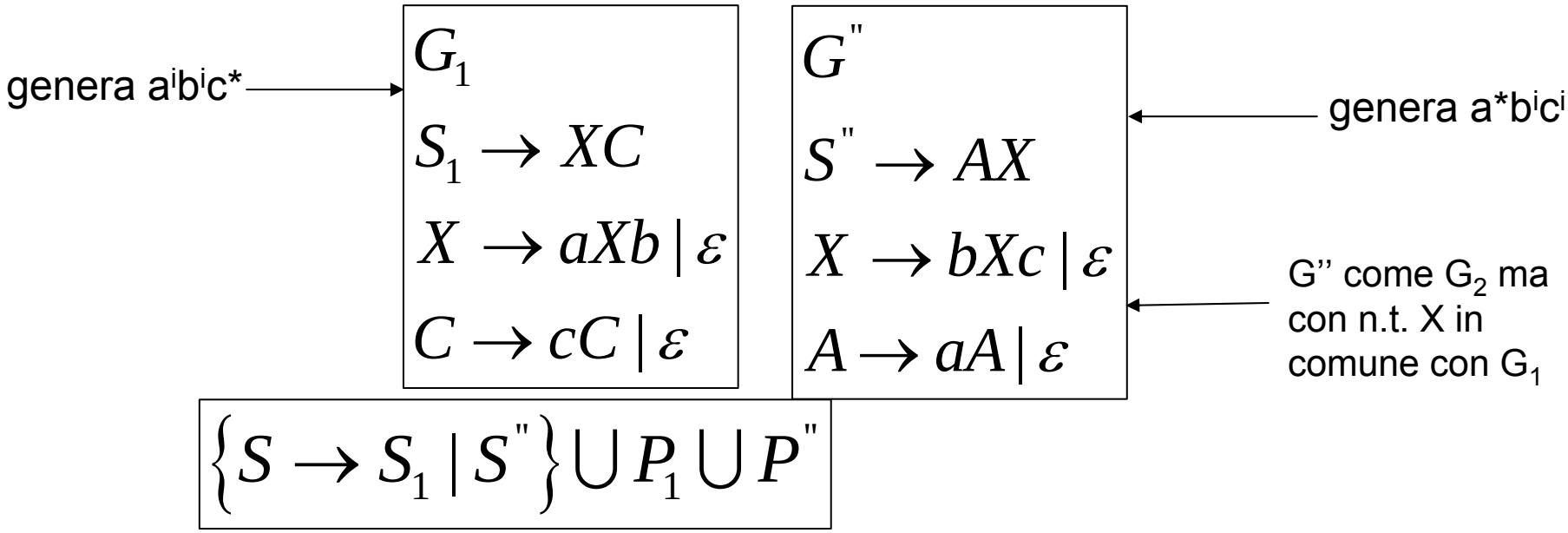
$A \rightarrow aA \mid \varepsilon$

← genera $a^* b^i c^i$

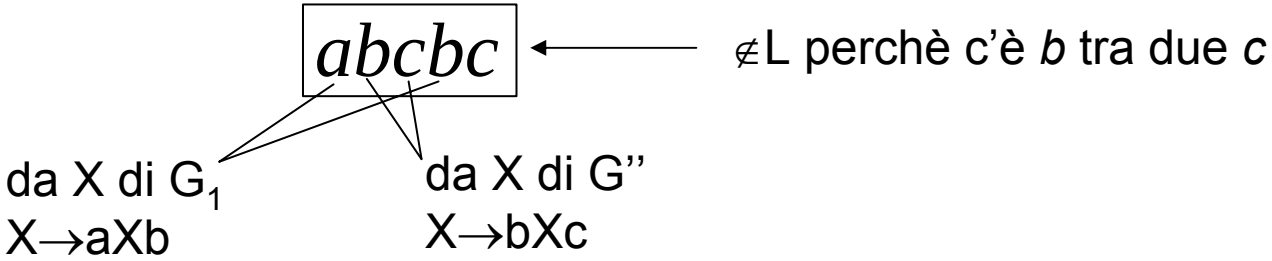
Si noti che i nonterminali dalle due grammatiche $G_1$ e $G_2$ sono DISGIUNTI

ESEMPIO: Unione di linguaggi (segue)

Che succederebbe se i nonterm. non fossero disgiunti? Usiamo G" al posto di $G_2$




Con alfabeti nonterminali non disgiunti, grammatica genera un sovrainsieme dell'unione dei due linguaggi: vengono introdotte frasi estranee come



La famiglia LIB è CHIUSA RISPETTO ALLA TRASF. SPECULARE  del linguaggio

Facile dimostrazione:

Si ottiene la grammatica del linguaggio speculare invertendo specularmente la parte destra di ogni regola.

AMBIGUITÀ : semantica / sintattica
“Ho visto un uomo in giardino con il cannocchiale” - “La pesca è bella” –
“half backed chicken” - ...

Anche nei linguaggi artificiali le ambiguità vanno controllate. Consideriamo AMBIGUITÀ SINTATTICHE:

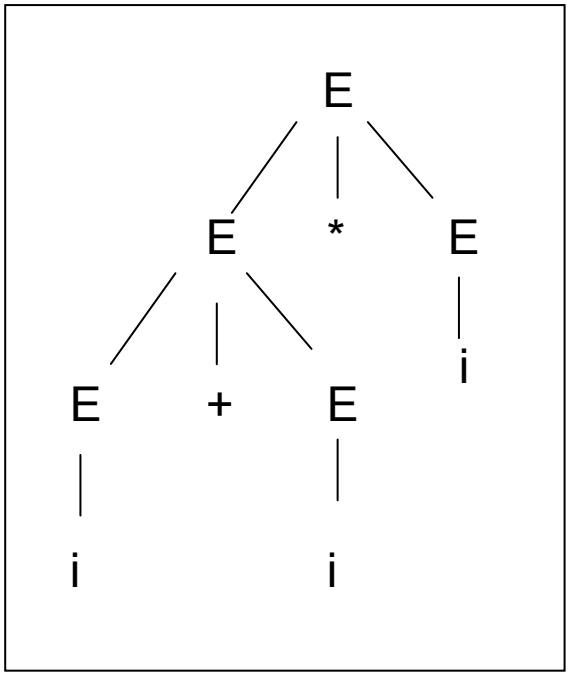
Una frase x del linguaggio definito dalla grammatica G è ambigua se ammette alberi sintattici distinti. Anche la grammatica G è detta, in tal caso, ambigua.

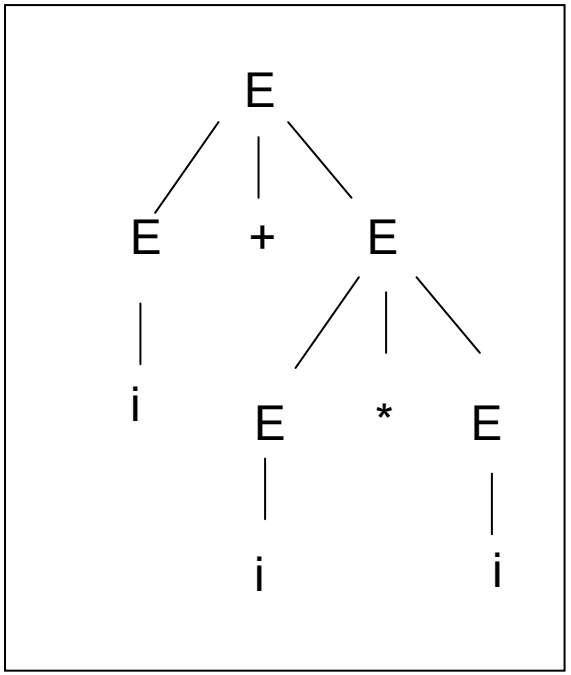
ESEMPIO:

$$E \to E + E \mid E * E \mid (E) \mid i$$

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow i + E * E \Rightarrow i + i * E \Rightarrow i + i * i$$

$$E \Rightarrow E + E \Rightarrow i + E \Rightarrow i + E * E \Rightarrow i + i * E \Rightarrow i + i * i$$




Anche la frase *i+i+i* è ambigua. La grammatica G' è dunque ambigua e non rispetta la tradizionale precedenza del prodotto sulla somma.

NB: G' è più piccola della G vista precedentemente (p.3): ha 1 solo n.t. e 4 regole

Il GRADO DI AMBIGUITÀ DI UNA FRASE x del linguaggio L(G) è il numero di alberi distinti con cui essa è generata da G

PER UNA GRAMMATICA IL GRADO DI AMBIGUITÀ è il massimo tra i gradi delle frasi. Il grado di ambiguità di una grammatica può risultare illimitato.

PROBLEMA IMPORTANTE: accertare l'ambiguità di una grammatica.

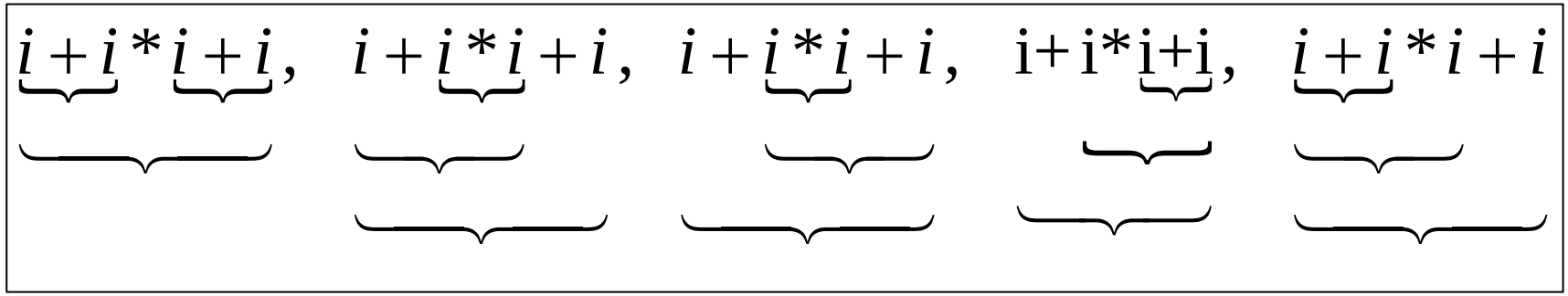
Il PROBLEMA è INDECIDIBILE: non esiste un algoritmo generale che,
data una grammatica libera,
si fermi dopo un numero finito di passi con la risposta: è (non è) ambigua.

La procedura che ricerca eventuali ambiguità sarebbe (in generale) trascinata ad esaminare derivazioni sempre più lunghe. Ma l'ambiguità di una particolare grammatica può essere esaminata come un problema a sé, caso per caso, facendo uso di RAGIONAMENTI INDUTTIVI. L'analisi di un numero ridotto di derivazioni è spesso adeguato per convincersi della inambiguità di una grammatica

MEGLIO EVITARE L'AMBIGUITÀ IN FASE DI PROGETTAZIONE

ESEMPIO: riprendiamo l'esempio precedente

Il grado di ambiguità vale 2 per *i* + *i* + *i* e vale 5 per *i* + *i* * *i* + *i*



Con l'allungarsi della frase il grado di ambiguità cresce senza limite.

CATALOGO DI FORME AMBIGUE E RIMEDI

1) AMBIGUITÀ NELLA RICORSIONE BILATERALE. Un simbolo non terminale A ricorsivo a sinistra e a destra dà luogo a diverse derivazioni che producono ambiguità

$$A \overset{+}{\Rightarrow} A\ldots \quad A \overset{+}{\Rightarrow} \ldots A$$

ESEMPIO1: la grammatica $G_1$ genera *i* + *i* + *i* in due modi diversi (con due diverse derivazioni sinistre).

$$G_1: \quad E \rightarrow E + E \mid i$$

Ma $L(G_1)=i(+i)^*$ (un linguaggio regolare!): possibili grammatiche piu` semplici

Gr. non ambigua ricorsiva a dx: $E \rightarrow i + E \mid i$

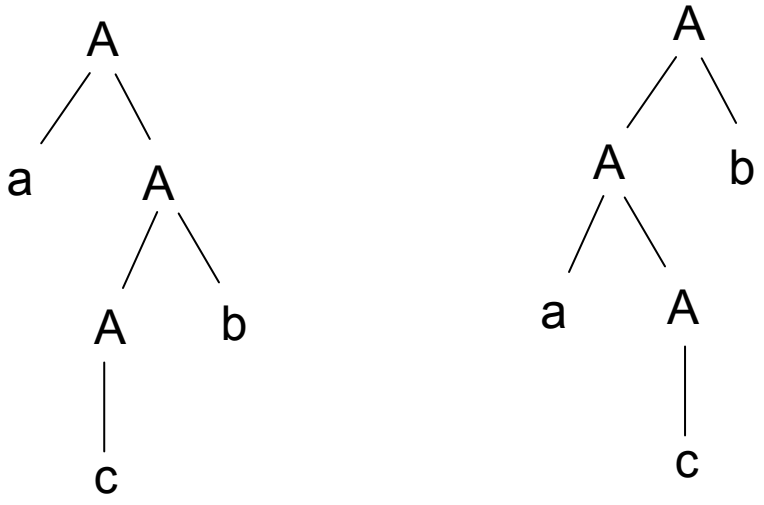
Gr. non ambigua ricorsiva a sx: $E \rightarrow E + i \mid i$

## ESEMPIO 2: Ricorsione sinistra e destra in regole diverse

Grammatica $G_2$ permette derivazioni con generazione di a o di b in una data frase in un ordine qualsiasi

$$G_2: \quad A \rightarrow aA \mid Ab \mid c$$

Esempio più semplice: due alberi sintattici per la stringa *acb*



ESEMPIO2 (segue): Ricorsione sinistra e destra in regole diverse

Primo modo di eliminare ambiguità: generare le due liste con regole separate

$$L(G_2) = a^* cb^*$$
$$S \to AcB$$
$$A \to aA \mid \varepsilon$$
$$B \to bB \mid \varepsilon$$

Secondo modo di eliminare ambiguità: imporre un ordine nelle derivazioni: es., generando le *a* prima delle *b*

$$L(G_2) = a^* cb^*$$
$$S \to aS \mid X$$
$$X \to Xb \mid c$$

ESEMPIO3: La grammatica che genera le espressioni polacche prefisse con somma e moltiplicazione, ha ricorsioni destre ma non sinistre e non è ambigua

$$S \to +SS \mid \times SS \mid i$$

## 2) AMBIGUITÀ NELL'UNIONE

Se due linguaggi $L_1(G_1)$ e $L_2(G_2)$ condividono alcune frasi (la loro intersezione non è vuota). La grammatica G del linguaggio unione dei due (slide n 9), risulta ambigua.

[Si suppone che le due grammatiche abbiano non terminali distinti, altrimenti l'unione delle regole potrebbe generare frasi non appartenenti all'unione dei due linguaggi (slide n 10)]

Una frase x appartenente all'unione di $L_1$ e $L_2$ che ammette due derivazioni distinte, una con le regole di $G_1$ e l'altra con le regole di $G_2$, risulta ambigua per la grammatica G che contiene tutte le regole. Non ambigue risultano le frasi appartenenti a $L_1 \setminus L_2$ oppure a $L_2 \setminus L_1$

Difficile rimediare: occorre rendere disgiunti i costrutti che generano ambiguita` o fonderli; spesso necessario cambiare la sintassi del linguaggio

## 3) AMBIGUITÀ INERENTE AL LINGUAGGIO

Un linguaggio si dice INERENTEMENTE AMBIGUO se tutte le grammatiche Equivalenti che lo generano sono ambigue.

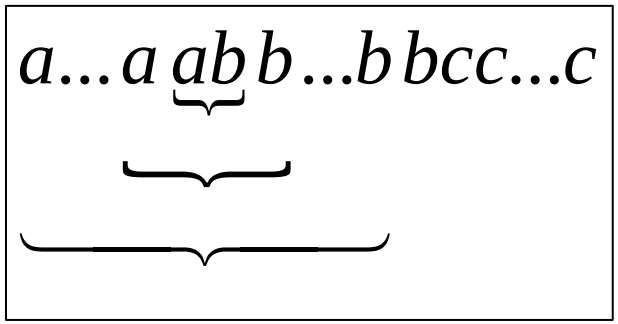
ESEMPIO: un linguaggio sempre ambiguo

$$L = \left\{ a^i b^j c^k \mid (i, j, k \geq 0) \wedge ((i = j) \vee (j = k)) \right\}$$

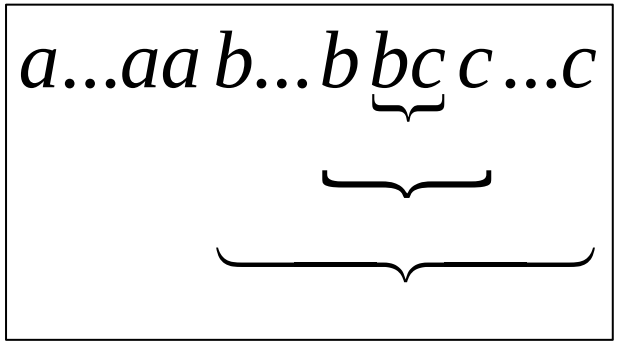
$$L = \left\{ a^i b^i c^* \mid i \geq 0 \right\} \bigcup \left\{ a^* b^i c^i \mid i \geq 0 \right\} = L_1 \bigcup L_2$$

La grammatica è ambigua per le frasi $\varepsilon$, $abc$, $a^2b^2c^2$, ...

Esse infatti sono generate da una $G_1$ con regole che verificano se $|x|_a = |x|_b$




E sono generate anche da una diversa $G_2$, con regole che verificano se $|x|_b=|x|_c$




Fortunatamente ambiguità inerente e` rara ed evitabile nei linguagi tecnici

## 4) AMBIGUITÀ DEL CONCATENAMENTO DI LINGUAGGI

Il concatenamento di due linguaggi può causare ambiguità quando esiste un suffisso di una frase del primo linguaggio che è anche un prefisso di una frase del secondo linguaggio.

$$G = (\Sigma_1 \cup \Sigma_2, \{S\} \cup V_{N_1} \cup V_{N_2}, \{S \to S_1 S_2\} \cup P_1 \cup P_2, S)$$

Supponendo $G_1$ e $G_2$ non ambigue, G è ambigua se se esistono due frasi rispettive $x' \in L_1$ e $x'' \in L_2$ tali che esiste una stringa non vuota *v* per cui

$$x' = u'v \wedge u' \in L_1 \quad x'' = vz'' \wedge z'' \in L_2$$

$$u'vz'' \in L_1.L_2 \quad \text{ed è ambigua:}$$

$$S \Rightarrow S_1 S_2 \overset{+}{\Rightarrow} u' S_2 \overset{+}{\Rightarrow} u'vz''$$

$$S \Rightarrow S_1 S_2 \overset{+}{\Rightarrow} u'vS_2 \overset{+}{\Rightarrow} u'vz''$$

ESEMPIO1 – Ambiguità nel concatenamento di linguaggi di Dyck

$$\Sigma_1 = \{a, a', b, b'\} \quad \Sigma_2 = \{b, b', c, c'\}$$

$$aa'bb'cc' \in L = L_1 L_2$$

$$G(L): \quad S \rightarrow S_1 S_2$$

$$S_1 \rightarrow aS_1a'S_1 \mid bS_1b'S_1 \mid \varepsilon$$

$$S_2 \rightarrow bS_2b'S_2 \mid cS_2c'S_2 \mid \varepsilon$$

$$\overbrace{aa'bb'}^{S_1}\overbrace{cc'}^{S_2} \qquad \overbrace{aa'}^{S_1}\overbrace{bb'cc'}^{S_2}$$

Per impedire l'ambiguità bisogna impedire lo spostamento di una stringa dal suffisso del primo linguaggio al prefisso del secondo: è possibile semplicemente interporre tra i due linguaggi un carattere terminale che faccia da separatore. Il carattere non deve appartenere agli alfabeti terminali. $L_1 \# L_2$ generato dalla grammatica avente regola iniziale $S \rightarrow S_1 \# S_2$

ESEMPIO2 – Codici univoci e non – ambiguità di concatenamento e codici nella teoria dell'informazione.

Un messaggio è una sequenza di simboli di un insieme finito Γ = {A, B,…, Z} che vengono poi codificati come stringhe di un alfabeto terminale Σ, tipicamente binario

$$\Gamma = \left\{ \overbrace{A}^{01}, \overbrace{C}^{10}, \overbrace{E}^{11}, \overbrace{R}^{001}, \right\}$$

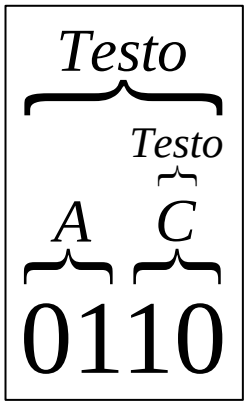
NB: nessuna codifica e` prefisso di un'altra

$$ARRECA : 01\ 001\ 001\ 11\ 10\ 01$$

$$Testo \rightarrow ATesto \mid CTesto \mid ETesto \mid RTesto \mid A \mid C \mid E \mid R$$

$$A \rightarrow 01 \quad C \rightarrow 10 \quad \mathrm{E} \rightarrow 11 \quad \mathrm{R} \rightarrow 001$$

La grammatica non è ambigua: ogni messaggio codificato, ha un solo albero sintattico e ammette quindi una sola decodifica



Una cattiva scelta della funzione di codifica rende ambigua la grammatica:

NB: la codifica di C e` prefisso di quella di R

$$\Gamma = \left\{ \overbrace{A}^{00}, \overbrace{C}^{01}, \overbrace{E}^{10}, \overbrace{R}^{010}, \right\}$$

$$Testo \rightarrow ATesto \mid CTesto \mid ETesto \mid RTesto \mid A \mid C \mid E \mid R$$

$$A \rightarrow 00 \quad C \rightarrow 01 \quad \mathrm{E} \rightarrow 10 \quad \mathrm{R} \rightarrow 010$$

$$ARRECA = 00\ 010\ 010\ 10\ 01\ 00$$

$$ACAEECA = 00\ 01\ 00\ 10\ 10\ 01\ 00$$

Il problema è legato a due aspetti:

01 00 10 = 010 010

01 è prefisso di 010

La *teoria dei codici* studia queste e altre condizioni atte a garantire che un insieme di codici sia univocamente decifrabile.

5) ALTRE SITUAZIONI AMBIGUE – nascono da espressioni regolari ambigue

ESEMPIO AMBIGUITA'

Ogni frase contenente due o più *c* è ambigua.

$$S \to DcD \quad D \to bD \mid cD \mid \varepsilon$$

$$\{b,c\}^* c \{b,c\}^*$$

RIMEDIO: individuare il *c* più a sinistra nella frase

$$S \to BcD \quad D \to bD \mid cD \mid \varepsilon \quad B \to bB \mid \varepsilon$$

ESEMPIO AMBIGUITA' Ogni frase con due o più *b* è ambigua.

$$S \to bSc \mid bbSc \mid \varepsilon$$

$$S \Rightarrow bbSc \Rightarrow bbbScc \Rightarrow bbbcc$$

$$S \Rightarrow bSc \Rightarrow bbbScc \Rightarrow bbbcc$$

RIMEDIO: Imposizione di ordine alle regole – La regola che genera due *b* può essere applicata prima o dopo la regola che genera un solo *b*.

$$S \to bSc \mid D \quad D \to bbDc \mid \varepsilon$$

6) AMBIGUITÀ DELLE FRASI CONDIZIONALI: il famoso ***dangling else***

$$S \rightarrow if\ \ b\ then\ S\ else\ S \mid if\ \ b\ then\ S \mid a$$

$$\overbrace{if\ \ b\ then\ \overbrace{if\ \ b\ then\ a\ else\ a}}$$

$$\overbrace{if\ \ b\ then\ \overbrace{if\ \ b\ then\ a}\ else\ a}$$

$$S \rightarrow S_E \mid S_T \quad S_E \rightarrow if\ \ b\ then\ S_E\ else\ S_E \mid a$$
$$S_T \rightarrow if\ \ b\ then\ S_E\ else\ S_T \mid if\ \ b\ then\ S$$

RIMEDIO 1 Escludiamo interpretazione; due n.t.: $S_E$ ha sempre l'else, $S_T$ ***può non*** avere l'else. Solo $S_E$ può precedere else

$$S \rightarrow if\ \ b\ then\ S\ else\ S\ end\_if \mid$$
$$\mid if\ \ b\ then\ S\ end\_if \mid a$$

RIMEDIO 2 Modifica il linguaggio, introduce marca di chiusura per delimitare il costrutto