Syntax Methods

- PL $\rightarrow$ rules defining syntax structure $\rightarrow$ **context-free grammar** (BNF)

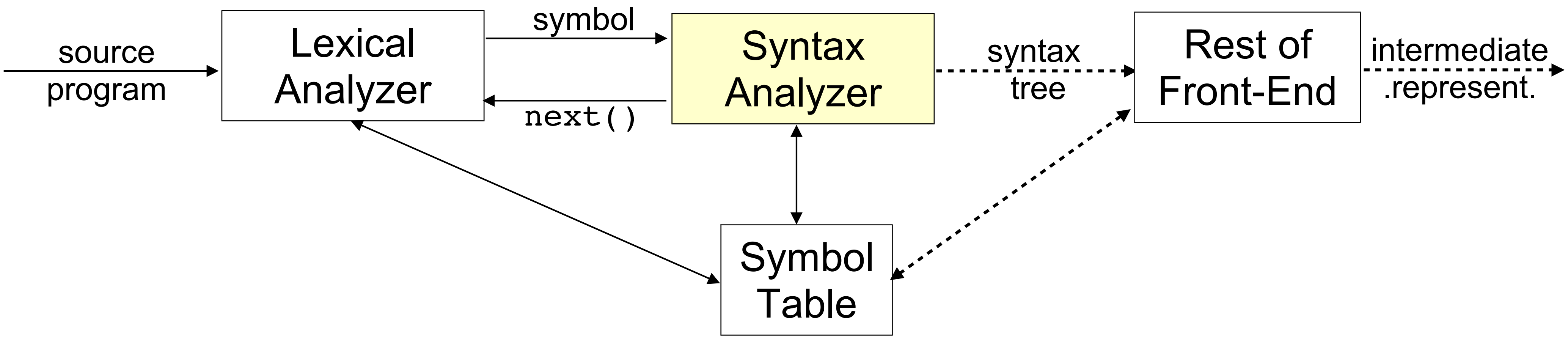
Type	Name	Productions
0	Recursively enumerable	$\alpha \rightarrow \beta$
1	Context-sensitive	$\alpha\ A\ \beta \rightarrow \alpha\ \gamma\ \beta$
2	Context-free	$A \rightarrow \gamma$
3	Regular	$A \rightarrow$ **a** $A \rightarrow$ **a** B

- Advantages in using G for designers of $\left\langle \begin{array}{l} \text{L} \\ \text{compiler of L} \end{array} \right.$

1. Formal, unambiguous, simple (yet powerful) notation
2. Possible automatic generation of parser of L(G)
3. Support for semantic analysis
4. Support for (syntax-directed) translation
5. Support for evolution of L: easier if implementation of L based on G (incrementality)

Syntax Methods (ii)

- Role of syntax analyzer:



- Taxonomy of syntax analyzers for context-free G:

  1. Universal ◄---------- inefficient
  2. Top-down
  3. Bottom-up

Context-Free Grammars

$$G = (T, N, P, S) \text{ where } \begin{cases} T = \{\text{ terminals }\} \\ N = \{\text{ nonterminals }\} \\ P = \{\text{ productions }\} \\ S = \text{axiom} \end{cases}$$

$$T \cup N \equiv \{\text{ grammar symbols }\}$$

$$\text{recursive} \begin{cases} expr \rightarrow expr \;\; op \;\; expr \\ expr \rightarrow \mathbf{(}\; expr \;\mathbf{)} \\ expr \rightarrow \mathbf{-}\; expr \end{cases}$$

$$expr \rightarrow \mathbf{id}$$

$$op \rightarrow \mathbf{+}$$

$$op \rightarrow \mathbf{-}$$

$$op \rightarrow \mathbf{*}$$

$$op \rightarrow \mathbf{/}$$

$$op \rightarrow \mathbf{\wedge}$$

$$\begin{cases} T = \{\, \mathbf{(}, \mathbf{)}, \mathbf{+}, \mathbf{-}, \mathbf{*}, \mathbf{/}, \mathbf{\wedge}, \mathbf{id} \,\} \\ N = \{\, expr, op \,\} \\ P = \{ \dots \} \\ S = expr \end{cases}$$

$$\Updownarrow$$

$$expr \rightarrow expr \;\; op \;\; expr \mid \mathbf{(}\; expr \;\mathbf{)} \mid \mathbf{-}\; expr \mid \mathbf{id}$$

$$op \rightarrow \mathbf{+} \mid \mathbf{-} \mid \mathbf{*} \mid \mathbf{/} \mid \mathbf{\wedge}$$

Derivations

- Process by which G defines L operationally: **derivation** / **syntax tree** (generative tool)

- **Derivation**: Precise idea of top-down construction of a syntax tree

  Production viewed as a **rewriting rule**: $A \to \alpha$: A replaced by α

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

$$\begin{cases} E \Rightarrow -E & \leftarrow E \text{ derives } -E \\ E * E \Rightarrow (E) * E \\ E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(\mathbf{id}) \end{cases}$$

$\Downarrow$

Derivation of **–(id)** from E = proof that **–(id)** is an instance of E

(phrase, if derived from aximom)

- More abstractly: we can say that $\alpha A \beta \Rightarrow \alpha\gamma\beta$ if $\begin{cases} A \to \gamma & = \text{ production} \\ \alpha, \beta & = \text{ strings of grammar symbols} \end{cases}$

Derivations (ii)

- Generalization: $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$ ← - - - - - - α_1 **derives** α_n (in several steps)

- Notation
  - $\Rightarrow$ derives in one step
  - $\overset{*}{\Rightarrow}$ derives in zero or more steps
  - $\overset{+}{\Rightarrow}$ derives in one or more steps

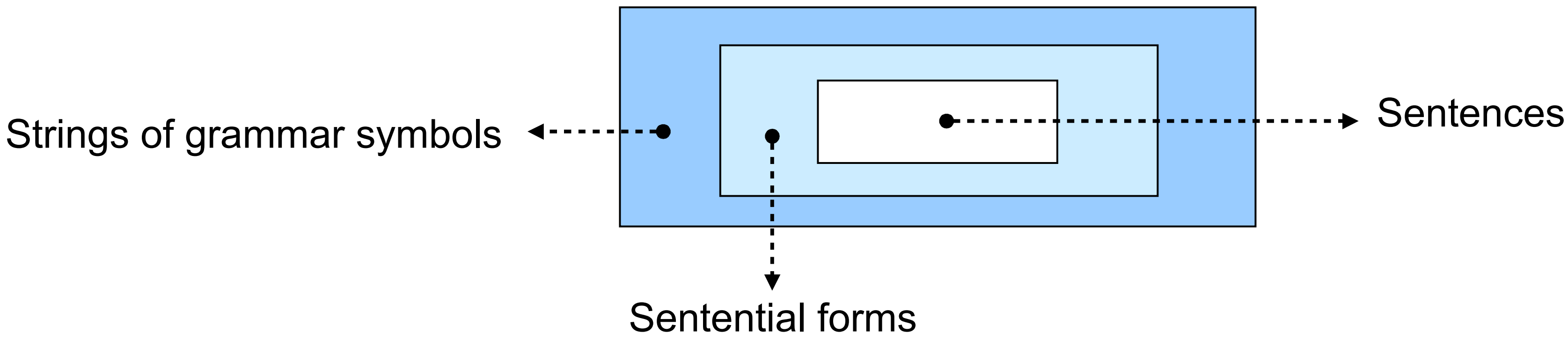
- Properties:
  1. $\forall$ string α ($\alpha \overset{*}{\Rightarrow} \alpha$)
  2. If $\alpha \overset{*}{\Rightarrow} \beta$ and $\beta \overset{*}{\Rightarrow} \gamma$, then $\alpha \overset{*}{\Rightarrow} \gamma$ (transitivity)

- Definition of L(G) by
  - S = axiom
  - $\overset{+}{\Rightarrow}$

  L(G) = { w | w = string of terminals, $S \overset{+}{\Rightarrow} w$ }

Derivations (iii)

- **Context-free L**: when can be generated by a context-free G
- If $L(G_1) = L(G_2)$ then G_1 and G_2 are **equivalent**
- $S \overset{*}{\Rightarrow} \alpha$, where α may contain nonterminals, $\alpha \equiv$ **sentential form** of G
  Sentence (phrase) of G = special sentential form of G composed of terminals only



$E \rightarrow E \mathbf{+} E \mid E * E \mid \mathbf{(}E\mathbf{)} \mid \mathbf{-}E \mid \mathbf{id}$ $\Longrightarrow$ $\mathbf{-(id + id)}$ = sentence of G

In fact: $E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow -(\mathbf{id}+\mathbf{id})$

We can express: $E \overset{*}{\Rightarrow} -(\mathbf{id}+\mathbf{id})$

Derivations (iv)

- Possible deriving a phrase in <u>different</u> modes

- **Canonical derivations** — **left** ($\alpha \underset{l}{\Rightarrow} \beta$), **right** ($\alpha \underset{r}{\Rightarrow} \beta$): at each step, substitution of leftmost / rightmost nonterminal

- Description of each step
  - leftmost: $wA\gamma \underset{l}{\Rightarrow} w\delta\gamma$
  - rightmost: $\gamma A w \underset{r}{\Rightarrow} \gamma\delta w$

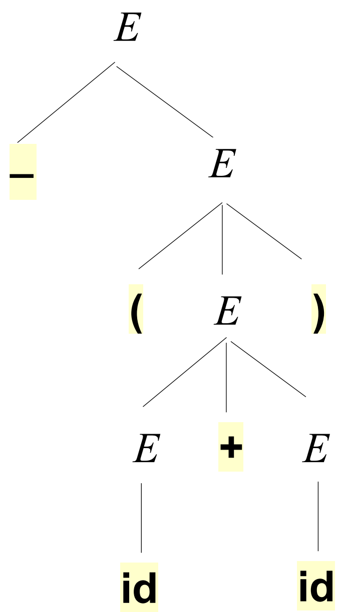
  where
  - w = string of terminals
  - $A \to \delta$ = production
  - γ = string of grammar symbols

- $S \overset{*}{\underset{l}{\Rightarrow}} \alpha \quad \Rightarrow \quad \alpha \equiv$ **left sentential form**

Relation between Syntax Trees and Derivations

- **Syntax tree** = graphical representation of a derivation irrespective of the order chosen in the substitutions (more direct representation of derivation)

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



- Variations in the order in which productions are applied can be eliminated considering canonical derivations

- Every syntax tree is associated with <u>one</u> canonical derivation (leftmost, rightmost)

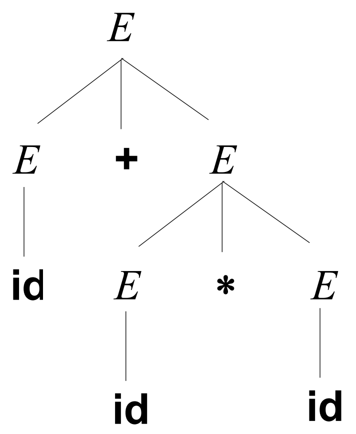
- G ambiguous: when $\exists$ sentence associated with <u>several</u> syntax trees
  $\rightarrow$ associated with <u>several</u> canonical derivations

Relation between Syntax Trees and Derivations (ii)

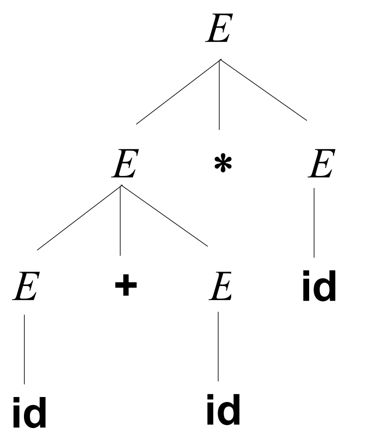
$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$ ⇨ Possible deriving $\mathbf{id + id * id}$ <u>canonically</u> in <u>different</u> modes

1. $E \underset{l}{\Rightarrow} E + E \underset{l}{\Rightarrow} \mathbf{id} + E \underset{l}{\Rightarrow} \mathbf{id} + E * E \underset{l}{\Rightarrow} \mathbf{id} + \mathbf{id} * E \underset{l}{\Rightarrow} \mathbf{id} + \mathbf{id} * \mathbf{id}$

2. $E \underset{l}{\Rightarrow} E * E \underset{l}{\Rightarrow} E + E * E \underset{l}{\Rightarrow} \mathbf{id} + E * E \underset{l}{\Rightarrow} \mathbf{id} + \mathbf{id} * E \underset{l}{\Rightarrow} \mathbf{id} + \mathbf{id} * \mathbf{id}$



correct association (1)



association <u>not</u>
reflected in the tree (2)

Ambiguity

- G **ambiguous**: When G produces <u>several</u> syntax trees for a sentence
  (≈ when G produces <u>several</u> canonical derivations for a sentence)

- Possible keeping G ambiguous → **Disambiguating rules**: leave out alternative syntax trees

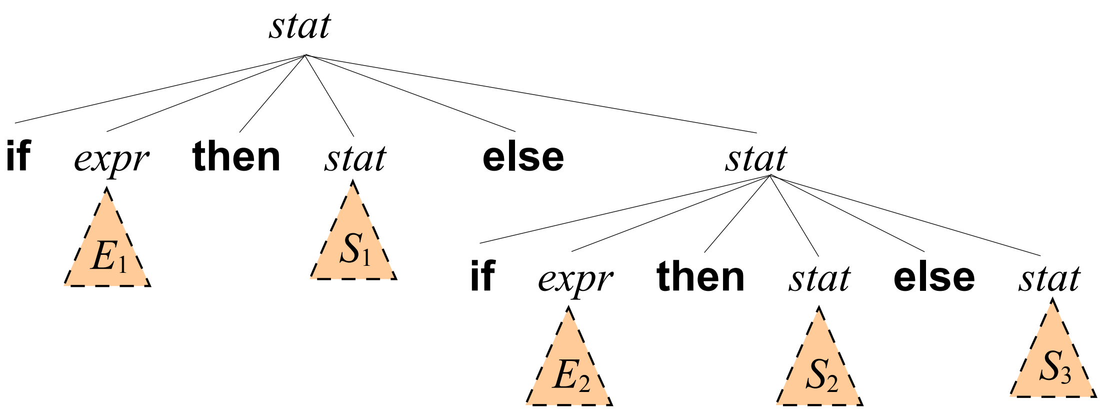
- Elimination of ambiguity in G:

G_1

stat → **if** *expr* **then** *stat* |
if *expr* **then** *stat* **else** *stat* |
other

if E_1 **then** S_1 **else if** E_2 **then** S_2 **else** S_3

(unambiguous)



Ambiguity (ii)

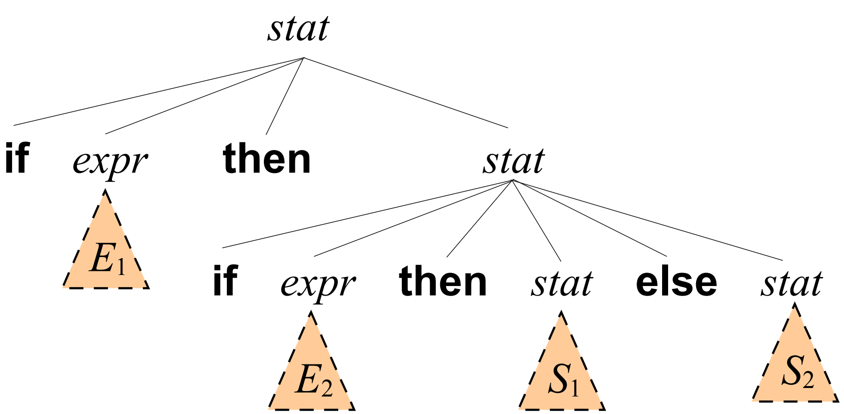
G₁

stat → **if** *expr* **then** *stat* |
if *expr* **then** *stat* **else** *stat* |
other

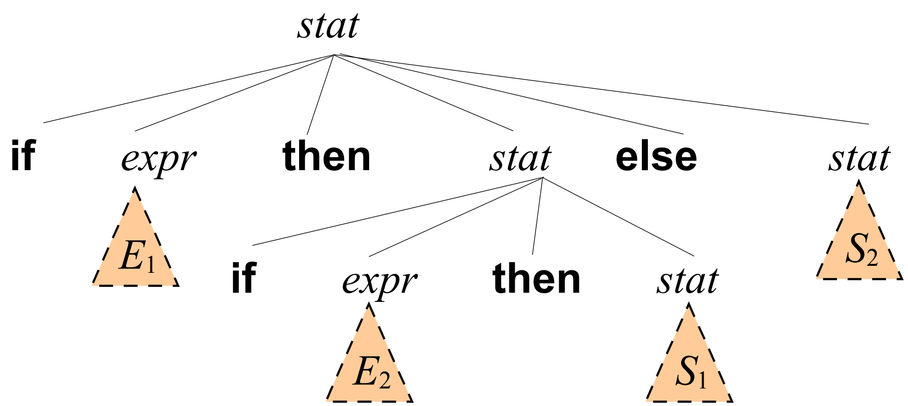
?

if E_1 **then** **if** E_2 **then** S_1 **else** S_2

(ambiguous)



(preferred in PL)



Disambiguating rule: Every **else** balances the nearest <u>unbalanced</u> **then**

⇨ statement between **then** and **else**: must be balanced (otherwise, rule violated)

Ambiguity (iii)

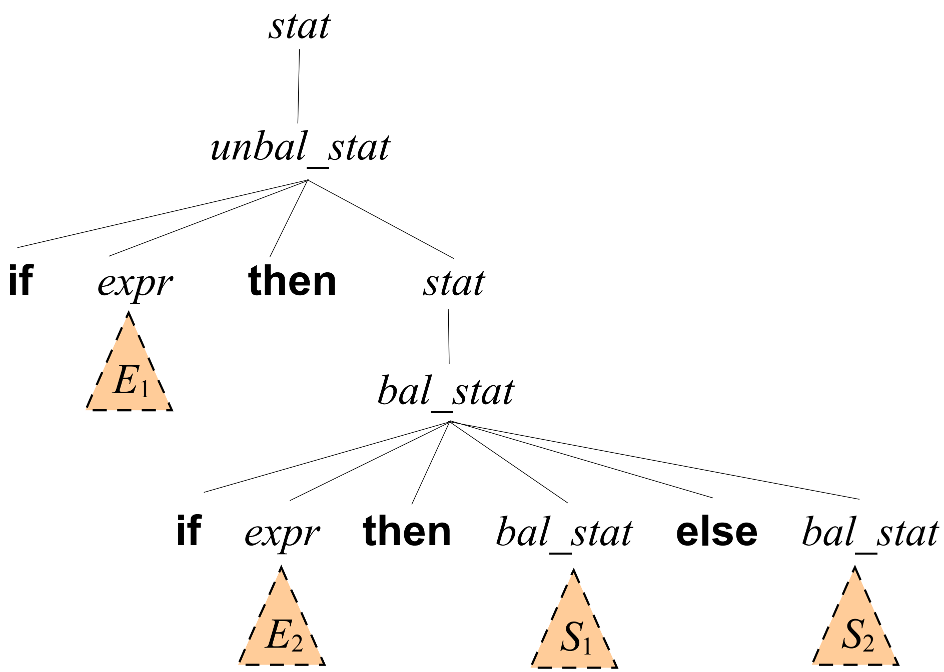
- Alternative: unambiguous grammar (based on disambiguating rule)

G_2

stat → *bal-stat* | *unbal_stat*
bal-stat → **if** *expr* **then** *bal-stat* **else** *bal-stat* | **other**
unbal_stat → **if** *expr* **then** *stat* |
  if *expr* **then** *bal-stat* **else** *unbal_stat*

$\Rightarrow$ $G_1 \approx G_2$

if E_1 **then if** E_2 **then** S_1 **else** S_2



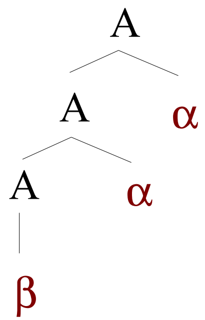
Left Recursion

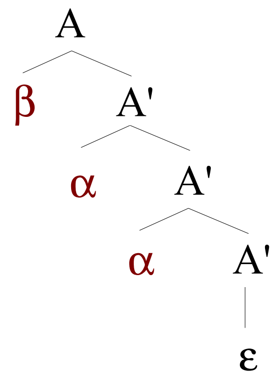
G **left recursive**: if $\exists$ A (A $\overset{+}{\Rightarrow}$ Aα) ← - - - - - - <u>intractable</u> with <u>top-down</u> parsing

1. **Simple direct recursion**:

$A \rightarrow A\alpha \mid \beta \quad \Rightarrow \quad A \rightarrow \beta A' \qquad A' \rightarrow \alpha A' \mid \varepsilon$

$E \rightarrow E + T \mid T \quad \Rightarrow \quad E \rightarrow TE' \qquad E' \rightarrow +TE' \mid \varepsilon$





2. **General direct recursion**:

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m \quad \Rightarrow \quad A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A' \qquad A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$

$E \rightarrow E + T \mid E - T \mid T \quad \Rightarrow \quad E \rightarrow TE' \qquad E' \rightarrow +TE' \mid -TE' \mid \varepsilon$

3. **Indirect recursion**:

$S \rightarrow A\mathbf{a} \mid \mathbf{b}$
$A \rightarrow A\mathbf{c} \mid S\mathbf{d} \mid \varepsilon$

$S \Rightarrow A\mathbf{a} \Rightarrow S\mathbf{da}$

$S \rightarrow A\mathbf{a} \mid \mathbf{b}$
$A \rightarrow A\mathbf{c} \mid B\mathbf{d} \mid \mathbf{e}$
$B \rightarrow B\mathbf{f} \mid S\mathbf{g} \mid \mathbf{h}$

$S \Rightarrow A\mathbf{a} \Rightarrow B\mathbf{da} \Rightarrow S\mathbf{gda}$

Left Recursion (ii)

General algorithm for elimination of left-recursion:

Hp: G without cycles: $A \overset{+}{\Rightarrow} A$, $A \rightarrow \varepsilon$ — sufficient condition

Order nonterminals: $A_1, A_2, \ldots, A_n$;

for i=1 **to** *n* **do**

  for j=1 **to** i-1 **do**

    Replace each production $A_i \rightarrow A_j\gamma$ with productions $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$,
    where $A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ are all current productions of A_j

  end-for;

  Eliminate possible direct left-recursions within productions of A_i

end-for.

A_1
A_2
.
.
.
A_j
.
.
.
A_i
.
.
.
A_n

Left Recursion (iii)

$S \to A\mathbf{a} \mid \mathbf{b}$
$A \to A\mathbf{c} \mid S\mathbf{d} \mid \varepsilon$

Note: Works even if not fulfilled sufficient condition ($A \to \varepsilon$)

- Ordering: $A_1 = S$, $A_2 = A$
- i = 1: Elimination of direct recursions of $A_1 = S$ $\Rightarrow$ $\nexists$
- i = 2: Substitution of $A_2 \to A_1\gamma = A \to S\mathbf{d}$ with $A \to A\mathbf{c} \mid A\mathbf{ad} \mid \mathbf{bd} \mid \varepsilon$

Elimination of direct recursions of A $\begin{cases} A \to \mathbf{bd}A' \mid A' \\ A' \to \mathbf{c}A' \mid \mathbf{ad}A' \mid \varepsilon \end{cases}$

$\Downarrow$

$S \to A\mathbf{a} \mid \mathbf{b}$
$A \to \mathbf{bd}A' \mid A'$
$A' \to \mathbf{c}A' \mid \mathbf{ad}A' \mid \varepsilon$

Left Recursion (iv)

$S \rightarrow A\mathbf{a} \mid \mathbf{b}$
$A \rightarrow A\mathbf{c} \mid B\mathbf{d} \mid \mathbf{e}$
$B \rightarrow B\mathbf{f} \mid S\mathbf{g} \mid \mathbf{h}$

- Order: $A_1 = S$, $A_2 = A$, $A_3 = B$
- i = 1: Elimination of direct recursions of S $\Rightarrow \nexists$
- i = 2: Substitution of $A_2 \rightarrow A_1\gamma = A \rightarrow S\gamma$ $\Rightarrow \nexists$

Elimination of direct recursions of A $\left\{ \begin{array}{l} A \rightarrow B\mathbf{d}A' \mid \mathbf{e}A' \\ A' \rightarrow \mathbf{c}A' \mid \varepsilon \end{array} \right.$ $\Rightarrow$

$S \rightarrow A\mathbf{a} \mid \mathbf{b}$
$A \rightarrow B\mathbf{d}A' \mid \mathbf{e}A'$
$A' \rightarrow \mathbf{c}A' \mid \varepsilon$
$B \rightarrow B\mathbf{f} \mid S\mathbf{g} \mid \mathbf{h}$

- i = 3: j = 1: Substitution of $A_3 \rightarrow A_1\gamma = B \rightarrow S\mathbf{g}$ with $B \rightarrow A\mathbf{ag} \mid \mathbf{bg}$ $\Rightarrow$

$S \rightarrow A\mathbf{a} \mid \mathbf{b}$
$A \rightarrow B\mathbf{d}A' \mid \mathbf{e}A'$
$A' \rightarrow \mathbf{c}A' \mid \varepsilon$
$B \rightarrow B\mathbf{f} \mid A\mathbf{ag} \mid \mathbf{bg} \mid \mathbf{h}$

j = 2: Substitution of $A_3 \rightarrow A_2\gamma = B \rightarrow A\mathbf{ag}$ with $B \rightarrow B\mathbf{f} \mid B\mathrm{d}A'\mathbf{ag} \mid \mathbf{e}A'\mathbf{ag} \mid \mathbf{bg} \mid \mathbf{h}$

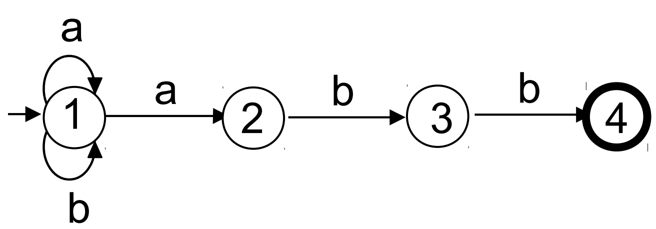
Elimination of direct recursions of B $\Rightarrow$

$S \rightarrow A\mathbf{a} \mid \mathbf{b}$
$A \rightarrow B\mathbf{d}A' \mid \mathbf{e}A'$
$A' \rightarrow \mathbf{c}A' \mid \varepsilon$
$B \rightarrow \mathbf{e}A'\mathbf{ag}B' \mid \mathbf{bg}B' \mid \mathbf{h}B'$
$B' \rightarrow \mathbf{f}B' \mid \mathbf{d}A'\mathbf{ag}B' \mid \varepsilon$

Relation between Grammars and Regular Expressions

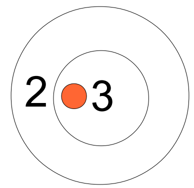
- $\forall$ regexpr $\Rightarrow$ $\exists$ equivalent G

(a | b)*abb



$\Rightarrow$

$A_1 \rightarrow \mathbf{a}A_1 \,|\, \mathbf{b}A_1 \,|\, \mathbf{a}A_2$
$A_2 \rightarrow \mathbf{b}A_3$
$A_3 \rightarrow \mathbf{b}A_4$
$A_4 \rightarrow \varepsilon$



- Algorithm for generating G starting from NFA = (Σ, S, T, s_0, F)

$\forall$ state $i \in$ S, create one nonterminal A_i;
$\forall$ transition $i \xrightarrow{a} j \in$ T, create one production $A_i \rightarrow \mathbf{a}A_j$;
$\forall$ final state $i \in$ F, create one production $A_i \rightarrow \varepsilon$;
Axiom of G = nonterminal corresponding to s_0.

EBNF

Same expressive power → increase in ⟨ writability, readability

Optionality: [ ]

if-stat → **if (** *expr* **)** *stat* [**else** *stat*]

Disjunction:

for-stat → **for** *var* := *expr* (**to** | **downto**) *expr* **do** *stat*

Repetition: { }

id-list → **id** {**, id** }

Non-empty repetition: $\{\ \}^+$

comp-stat → **begin** { *stat* }$^+$ **end**

Table of Operators and Grammar of Expressions

- Rules of precedence / associativity: establish the operands of each operator

- G for expressions: specifiable based on table of operators: associations, precedences

Association	*Operators*	*Nonterminal*
left	+ –	*expr*
left	∗ /	*term*
		factor

increasing precedence

⇒ ∀ level of precedence → nonterminal
Further nonterminal *factor* for base units

$$expr \rightarrow expr \ \mathbf{+} \ term \mid expr \ \mathbf{-} \ term \mid term$$
$$term \rightarrow term \ \mathbf{*} \ factor \mid term \ \mathbf{/} \ factor \mid factor$$
$$factor \rightarrow \mathbf{digit} \mid \mathbf{(}expr\mathbf{)}$$