

# Compilers

Third course  
Spring term

Alfonso Ortega: [alfonso.ortega@uam.es](mailto:alfonso.ortega@uam.es)  
Enrique Alfonseca: [enrique.alfonseca@uam.es](mailto:enrique.alfonseca@uam.es)



## Top-down analysis



## Top-down analysis

### Overview

- **Top-down analysis**
  - Blind search, depth-first search and width-first search.
  - Slow backtracking
- **Top-down analysis with LL(1) grammars**
  - **Procedures for modifying grammars**
    - Elimination of left-recursion
    - Elimination of lambda-rules
  - **Greibach Normal Form**
    - Definition
    - Procedure for obtaining the GNF of a grammar
    - Examples
  - **LL(1) grammars**
    - Initial examples
    - Definition of LL(1) grammars
    - Syntactic analysers based on LL(1) grammars

## Top-down analysis

### Introduction

- We need to find a derivation from the axiom to the program analysed.
- However, depending on the grammar, the number of possible derivations is extremely large.
- There are general strategies to solve any problem with blind search:
  - Breadth-first search and depth-first search.
  - In **breadth-first search**, it is guaranteed that we shall find a solution.
  - In **depth-first search**, there might be branches with infinite lengths. If that is not the case, sometimes it is possible to find a solution with less effort than in breadth-first search.

## Top-down analysis

### Top-down with backtracking: concepts

- We can consider “top-down analysis with a **backtracking parser**” as a particular case of depth-first blind search.
- The criterion to decide whether a rule is applicable in a given position is the coincidence with the non-terminal that is in the left-hand side of the rule.
- We shall not continue along an analysis path when we have terminals that do not coincide with the program.
- We can finish the process in two situations:
  - If we have been able to generate all the terminals in the program sequence. In this case, we have been able to construct a syntactic tree for the program, which is correct.
  - If it was not possible to finish with the derivation tree, but we have tried all the rules in each position and there is no other option available to try. In this case, the program will be syntactically incorrect.
- This technique will be illustrated with the following examples:

5

## Top-down analysis

### Previous examples

- Syntactic analysis of the word **aaabbb**

```
G=<{S}, {a,b}
{S → aSb
  |ab},
S>
```

S

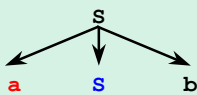
aaabbb

6

## Top-down analysis

### Previous examples

```
G=<{S}, {a,b}
{S → aSb
  |ab},
S>
```

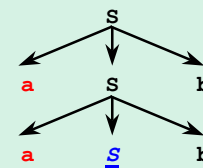


7

## Top-down analysis

### Previous examples

```
G=<{S}, {a,b}
{S → aSb
  |ab},
S>
```



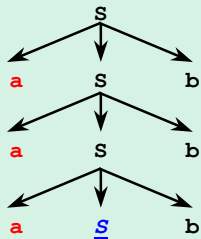
8

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

aaabb



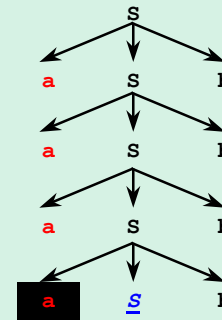
9

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

aaabb



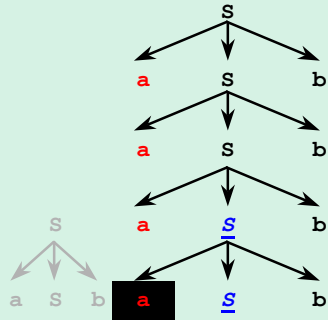
10

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

aaabb



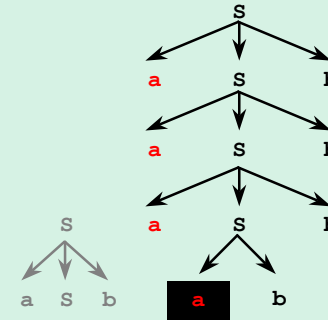
11

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

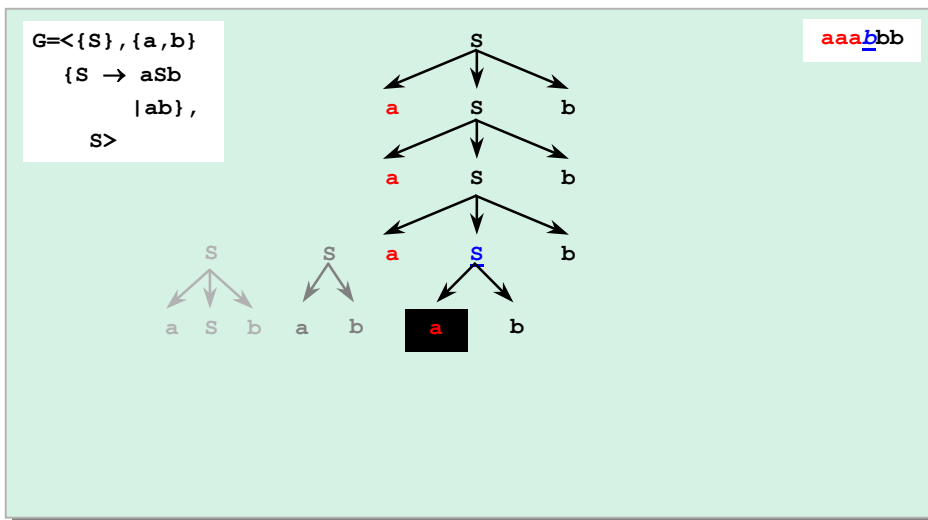
aaabb



12

## Top-down analysis

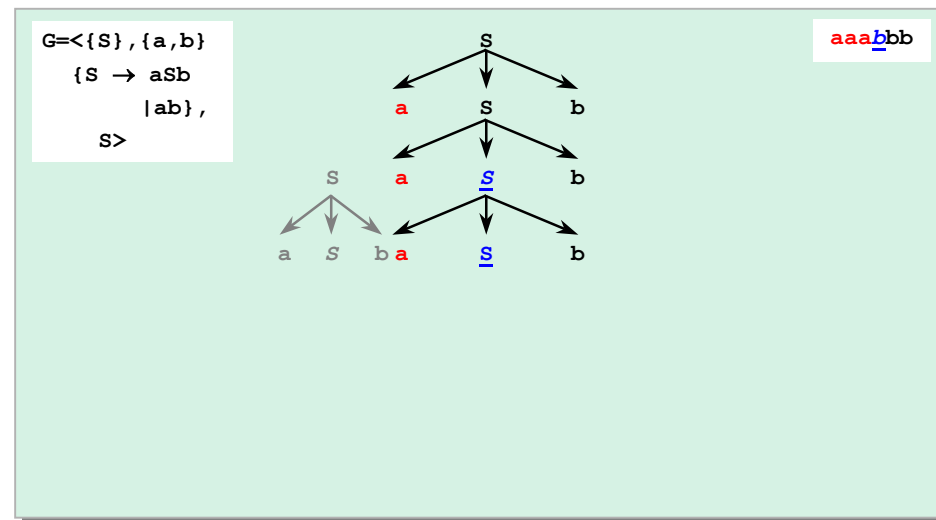
Previous examples



13

## Top-down analysis

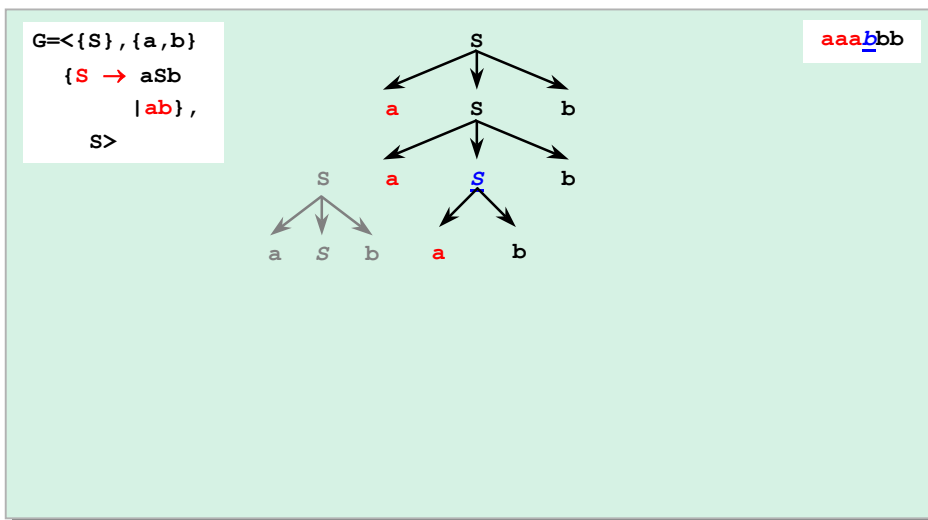
Previous examples



14

## Top-down analysis

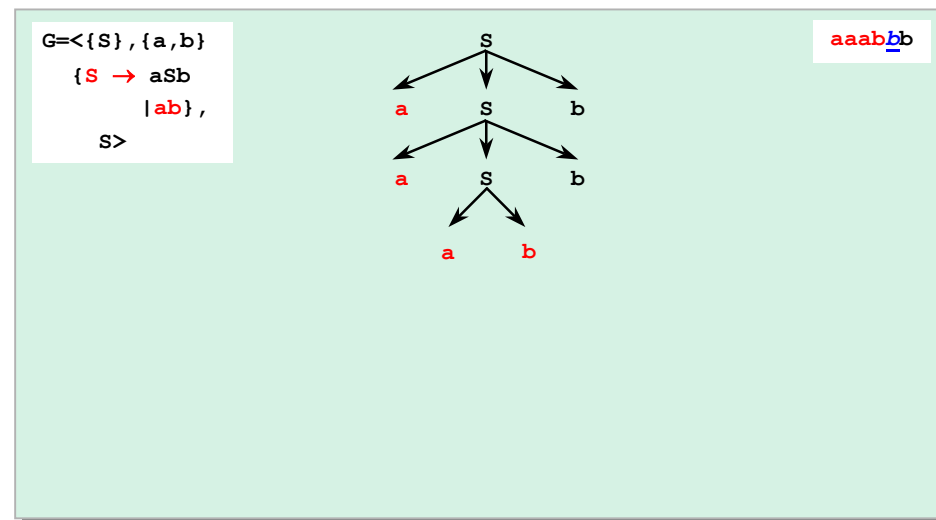
Previous examples



15

## Top-down analysis

Previous examples



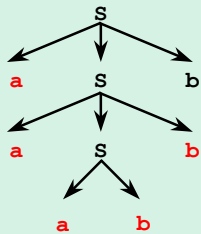
16

# Top-down analysis

Previous examples

- La next palabra

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

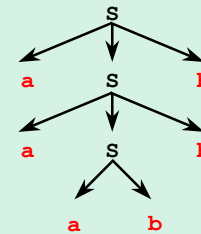


aaabbb

# Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$



aaabbb

ACCEPTED

# Top-down analysis

Previous examples

- Syntactic analysis of the word abbb

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

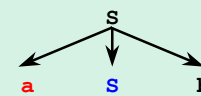
S

abbb

# Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

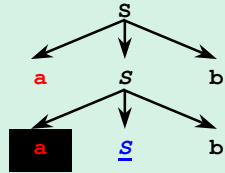


abbb

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

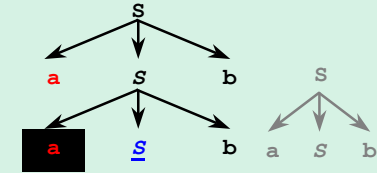


21

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

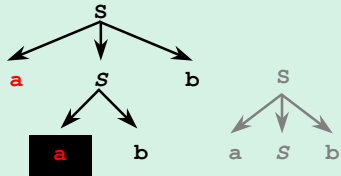


22

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

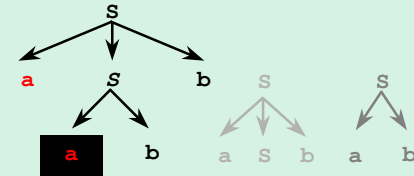


23

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$

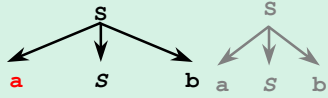


24

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$



abbb

25

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$



abbb

26

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$



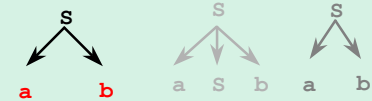
abbb

27

## Top-down analysis

Previous examples

$G = \langle \{S\}, \{a, b\}$   
 $\{S \rightarrow aSb$   
 $\quad | ab\},$   
 $S \rangle$



abbb

REJECTED

28

## Top-down analysis

Previous examples

- Analysis of the word  $i+--i$

```
G=<{E},{-,+ ,i}
{E→-E
 |i
 |E+E},
E>
```

E

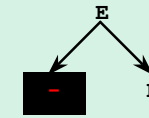
i+--i

29

## Top-down analysis

Previous examples

```
G=<{E},{-,+ ,i}
{E→-E
 |i
 |E+E},
E>
```



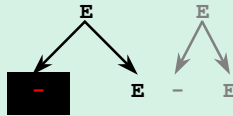
i+--i

30

## Top-down analysis

Previous examples

```
G=<{E},{-,+ ,i}
{E→-E
 |i
 |E+E},
E>
```



i+--i

31

## Top-down analysis

Previous examples

```
G=<{E},{-,+ ,i}
{E→-E
 |i
 |E+E},
E>
```



i+--i

32

## Top-down analysis

Previous examples

$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$



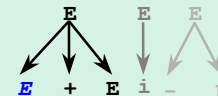
$i+--i$

33

## Top-down analysis

Previous examples

$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$



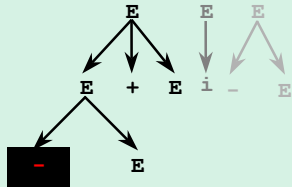
$i+--i$

34

## Top-down analysis

Previous examples

$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$



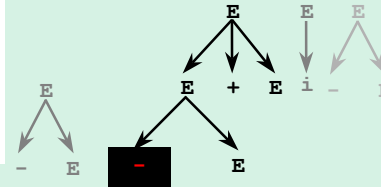
$i+--i$

35

## Top-down analysis

Previous examples

$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$



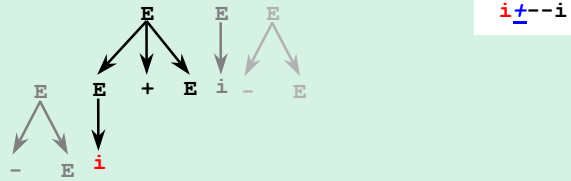
$i+--i$

36

## Top-down analysis

Previous examples

$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$

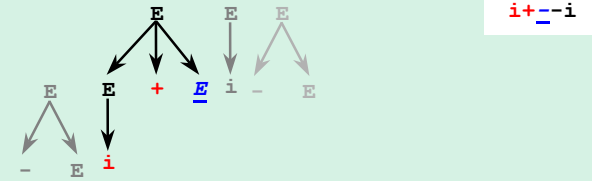


37

## Top-down analysis

Previous examples

$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$

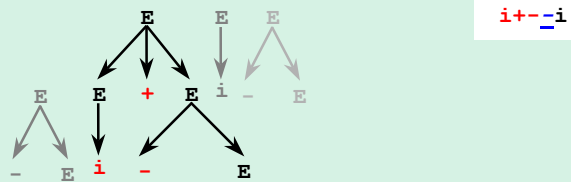


38

## Top-down analysis

Previous examples

$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$

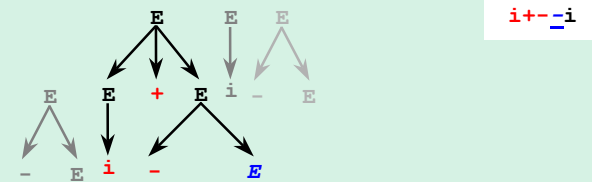


39

## Top-down analysis

Previous examples

$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$

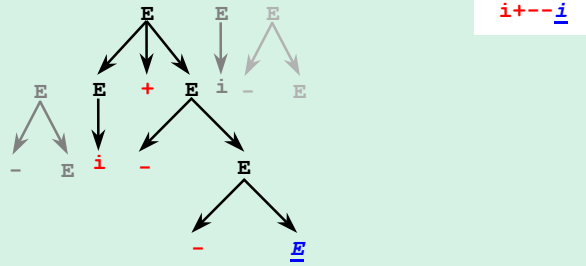


40

## Top-down analysis

Previous examples

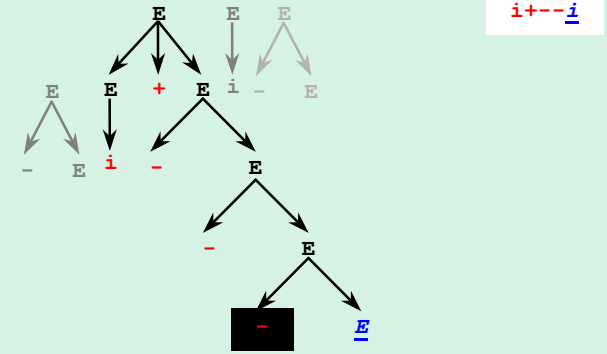
$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$



## Top-down analysis

Previous examples

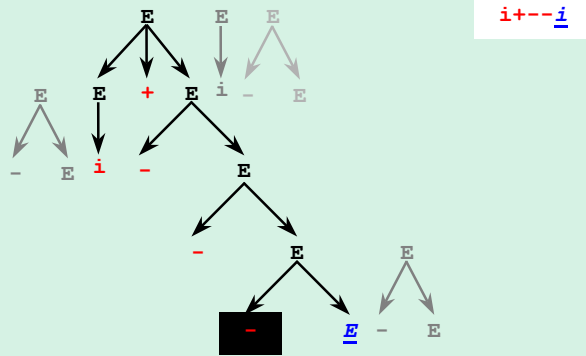
$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$



## Top-down analysis

Previous examples

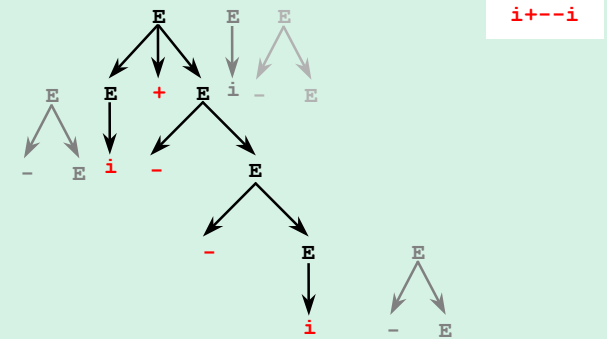
$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$



## Top-down analysis

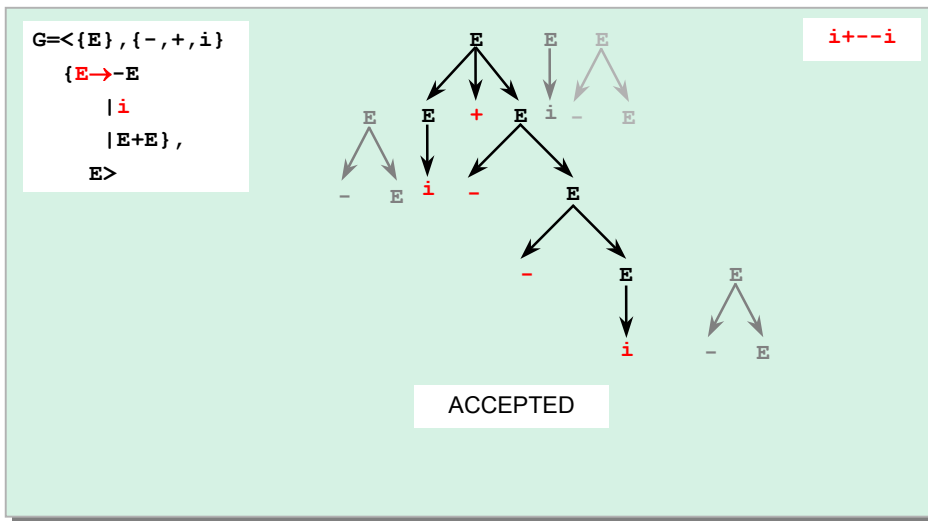
Previous examples

$G = \langle \{E\}, \{-, +, i\}$   
 $\{E \rightarrow -E$   
 $\quad | i$   
 $\quad | E+E\},$   
 $E \rangle$



## Top-down analysis

Previous examples

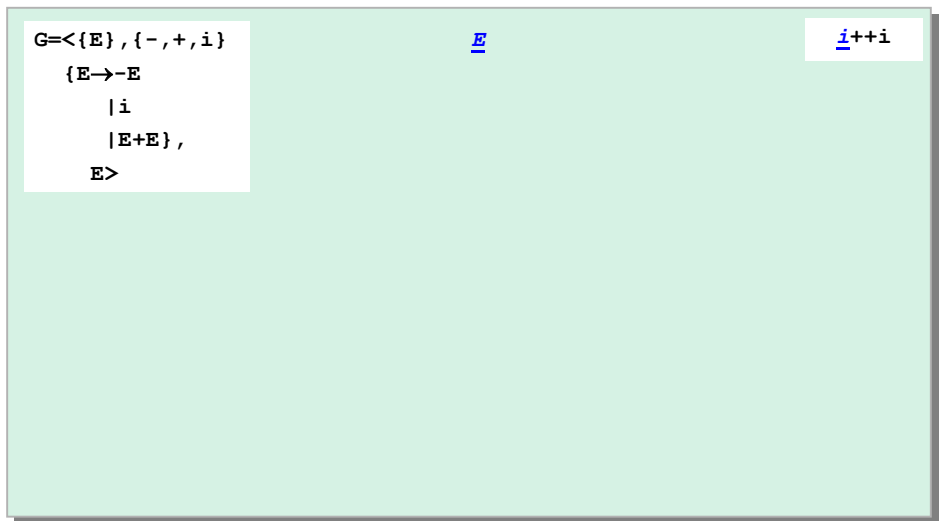


45

## Top-down analysis

Previous examples

- Consider the analysis of  $i++i$



46

## Top-down analysis

Conclusions

- If there are left-recursive rules, the analysis of the word may provoke that the algorithm enters an infinite loop.
- In the following slides, we shall study the possibilities for avoiding that fact, and we shall see other properties of grammars which are interesting for building efficient top-down syntactic analysers.

47

## Top-down analysis

Overview

- Top-down analysis**
  - Blind search, depth-first search and width-first search.
  - Slow backtracking
- Top-down analysis with LL(1) grammars**
  - Procedures for modifying grammars**
    - Elimination of left-recursion
    - Elimination of lambda-rules
  - Greibach Normal Form**
    - Definition
    - Procedure for obtaining the GNF of a grammar
    - Examples
  - LL(1) grammars**
    - Initial examples
    - Definition of LL(1) grammars
    - Syntactic analysers based on LL(1) grammars

48

## Properties of grammars for top-down analysis

### Properties

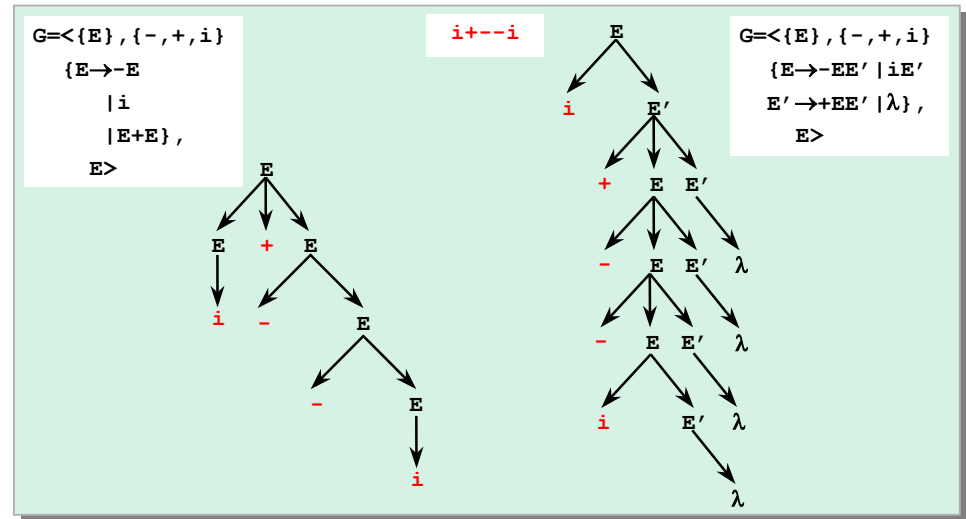
- It is easy to build LL(1) analysers, in which the input is read from left to right, and the derivations in the tree are also made from left to right. The number "1" means that it needs one look-ahead symbol from the input string.
- In order to build an LL(1) analyser, the grammar has to be expressed in LL(1) form.
- Possible steps to transform a grammar into LL(1) form are:
  - Removing all the left-recursive rules.
  - Removing inaccessible symbols.
  - Removing rules that generate the empty word:  $A \rightarrow \lambda$
  - Expressing the grammar in Greibach Normal Form.
  - Left factorisation (which is a necessary condition for LL(1) grammars)

49

## Greibach Normal Form

### Removing left-recursive rules

- Let us analyse the following example:



50

## Greibach Normal Form

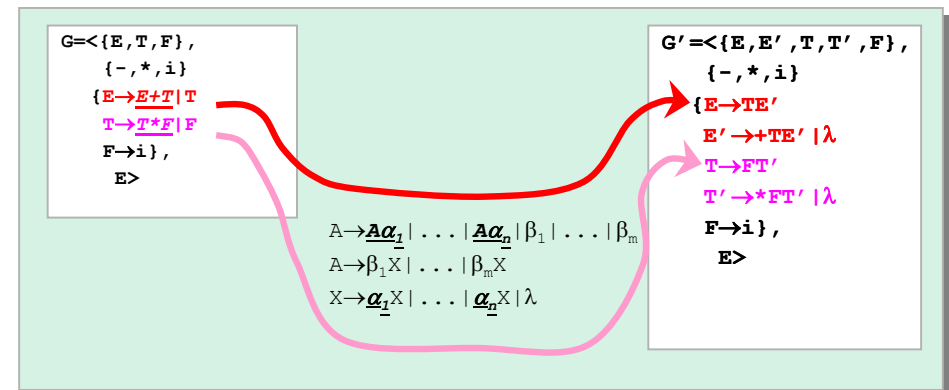
### Removing left-recursive rules

- It is difficult to think of a general procedure from the previous example.
- However, the technique can be generalised.

51

## Greibach Normal Form

### Removing left-recursive rules: first example



52

## Greibach Normal Form

### Lemma: removing left-recursive rules

Every context-independent grammar can be transformed into an equivalent grammar without left-recursive rules.

- A left-recursive rule is a rule of the following form:
 
$$A \rightarrow Ax, \quad A \in \Sigma_N \wedge x \in (\Sigma_T \cup \Sigma_N)^*$$
- The lemma is shown in a constructive way doing, for each recursive rule, the following treatment:
- Let  $\langle \Sigma_T, \Sigma_N, S, P \rangle$  be a grammar with left-recursive rules:
  - $A \rightarrow A\alpha_1 | \dots | A\alpha_n | \beta_1 | \dots | \beta_m$
  - Where  $\{\beta_i\}_{i=1}^m$  represent all the non-left-recursive rules
- We can substitute the previous rules by the following set of rules
  - $A \rightarrow \beta_1 X | \dots | \beta_m X$
  - $X \rightarrow \alpha_1 X | \dots | \alpha_n X | \lambda$

53

## Greibach Normal Form

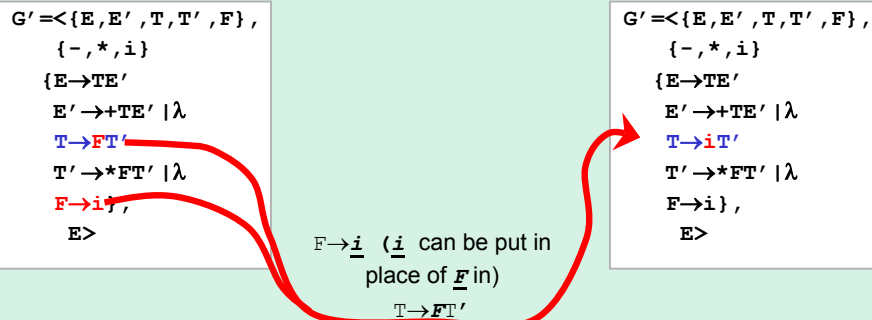
### Removing lambda-rules

We can modify a rule by substituting a non-terminal in its right-hand side by all the right-hand sides of the rules for that non-terminal. The grammar obtained in this way generates the same language as the original one.

54

## Greibach Normal Form

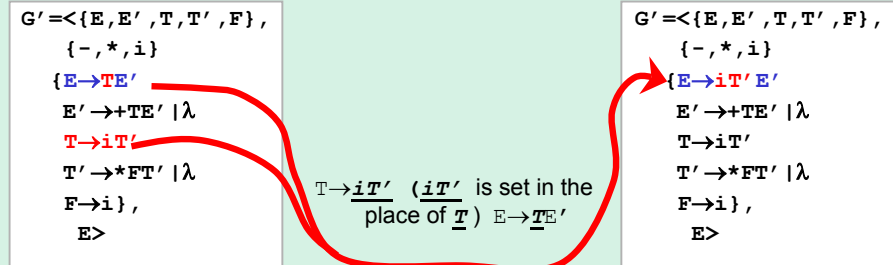
### Removing lambda-rules



55

## Greibach Normal Form

### Removing lambda-rules



56

## Greibach Normal Form

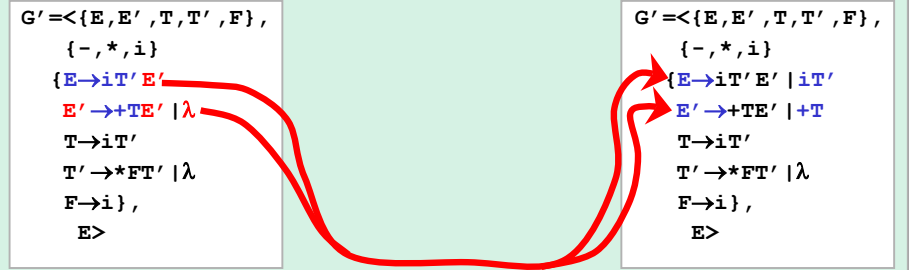
### Removing lambda-rules

We can delete a rule for a non-terminal symbol, provided that we also add new rules by substituting all appearances of the non-terminal by all the right-hand sides of the rules eliminated.

- A case with special interest is the rules that generate the empty word,  $\lambda$
- A lambda-rule is a rule of the following form:
 
$$A \rightarrow \lambda, A \in \Sigma_N$$
- It is used to remove a non-terminal symbol from a word.

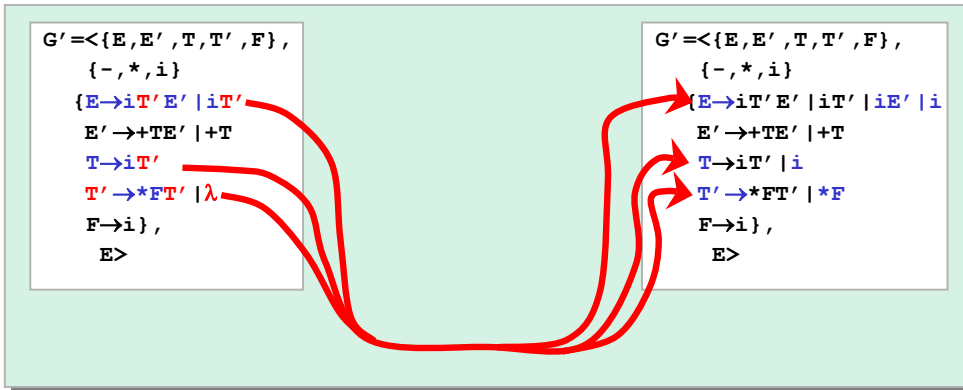
## Greibach Normal Form

### Removing lambda-rules



## Greibach Normal Form

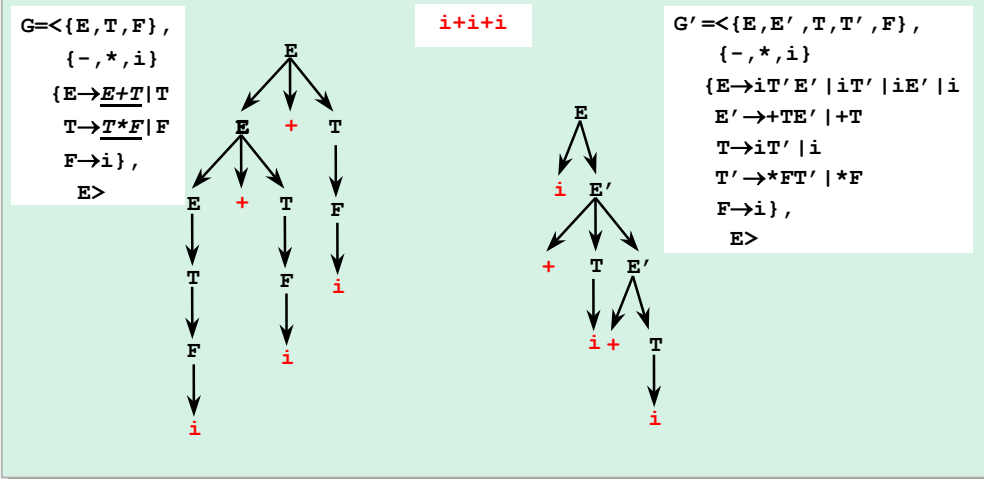
### Removing lambda-rules



## Greibach Normal Form

### Removing lambda-rules

- Let us test some derivations:



## Greibach Normal Form

### Removing lambda-rules

- Let us test some derivations:

$G = \langle \{E, T, F\}, \{-, *, i\}, \{E \rightarrow \underline{E+T} | T, T \rightarrow \underline{T*F} | F, F \rightarrow i\}, E \rangle$

i\*i\*i

$G' = \langle \{E, E', T, T', F\}, \{-, *, i\}, \{E \rightarrow iT'E' | iT' | iE' | i, E' \rightarrow +TE' | +T, T \rightarrow iT' | i, T' \rightarrow *FT' | *F, F \rightarrow i\}, E \rangle$

## Greibach Normal Form

### Removing lambda-rules

- Let us test some derivations:

$G = \langle \{E, T, F\}, \{-, *, i\}, \{E \rightarrow \underline{E+T} | T, T \rightarrow \underline{T*F} | F, F \rightarrow i\}, E \rangle$

i\*i+i

$G' = \langle \{E, E', T, T', F\}, \{-, *, i\}, \{E \rightarrow iT'E' | iT' | iE' | i, E' \rightarrow +TE' | +T, T \rightarrow iT' | i, T' \rightarrow *FT' | *F, F \rightarrow i\}, E \rangle$

## Top-down analysis

### Overview

- Top-down analysis**
  - Blind search, depth-first search and width-first search.
  - Slow backtracking
- Top-down analysis with LL(1) grammars**
  - Procedures for modifying grammars**
    - Elimination of left-recursion
    - Elimination of lambda-rules
  - Greibach Normal Form**
    - Definition
    - Procedure for obtaining the GNF of a grammar
    - Examples
  - LL(1) grammars**
    - Initial examples
    - Definition of LL(1) grammars
    - Syntactic analysers based on LL(1) grammars

## Greibach Normal Form

### Definition

- Informally,
 

A context-independent grammar is in Greibach Normal Form if and only if the right-hand-side of all the rules starts with a terminal symbol, followed, optionally, by non-terminals.
- Formally,
 

$\langle \Sigma_T, \Sigma_N, S, P \rangle$ , context-independent, is in Greibach Normal Form is  $\Leftrightarrow$  (def)
 
$$\forall r \in P \ r = A \rightarrow ax \ \ , \ \ a \in \Sigma_T \ \wedge \ x \in \Sigma_N^*$$

## Greibach Normal Form

### Theorem

Every context-independent grammar which does not generate the empty word can be expressed in Greibach Normal form.

65

## Greibach Normal Form

### Proof of the theorem

- It will be a constructive proof, showing how the grammar can be transformed.
- The transformation is performed in the following way:

1. If the language contains the empty word, add the following rule (S is the axiom)

$$S \rightarrow \lambda$$

2. Remove all the left-recursive rules applying the lemma seen before.
3. The following partial ordering will be established between the non-terminal symbols, deduced from the production rules:

$$A_i < A_j \Leftrightarrow (\exists \alpha \in \Sigma_T^* \mid A_i \rightarrow A_j \alpha \in P) \wedge (\neg \exists \beta \in \Sigma_T^* \mid A_j \rightarrow A_i \beta \in P)$$

If this ordering produces a loop, i.e., if

$$\exists \alpha, \beta \in \Sigma_T^* \mid A_i \rightarrow A_j \alpha \in P \wedge A_j \rightarrow A_i \beta \in P$$

we can choose any of the two following options:

- $A_i < A_j$
- $A_j < A_i$

66

## Greibach Normal Form

### Example of step 3

$G = \langle \{E, T, F\},$   
 $\{-, *, i\}$   
 $\{E \rightarrow E+T \mid T, T \rightarrow T*F \mid F, F \rightarrow i\},$   
 $E \rangle$

- The following partial ordering can be deduced from the rules in the grammar:
  - $E \rightarrow E+T$  does not provide any ordering.
  - From  $E \rightarrow T$  we can deduce  $E < T$
  - From  $T \rightarrow T*F$  we do not get any ordering.
  - From  $T \rightarrow F$  we can deduce  $T < F$
  - From  $F \rightarrow i$  we cannot deduce anything.
- In conclusion,

$$E < T < F$$

67

## Greibach Normal Form

### Proof of the theorem

4. We can classify the rules in the following three groups:

- **Type 1:** Rules of the form

$$A \rightarrow ax, \quad a \in \Sigma_T, \quad x \in \Sigma^*$$

- **Type 2:** Rules of the form

$$A \rightarrow Bx, \quad A, B \in \Sigma_N, \quad x \in \Sigma^* \wedge A < B$$

- **Type 3:** Rules of the form

$$A \rightarrow Bx, \quad A, B \in \Sigma_N, \quad x \in \Sigma^* \wedge B < A$$

68

## Greibach Normal Form

### Example of step 4

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \rangle$$

$$\{E \rightarrow TE', E' \rightarrow +TE' \mid \lambda, T \rightarrow FT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\},$$

$$E \rangle$$

- Type 1 rules:
  - $E' \rightarrow +TE'$
  - $T' \rightarrow *FT'$
  - $F \rightarrow i$
- Type 2 rules:
  - $E \rightarrow TE'$
  - $T \rightarrow FT'$
- There are no type 3 rules.
- The lambda-rules will be seen later.

69

## Greibach Normal Form

### Proof of the theorem

5. All the rules of type 3 will be deleted.
- **Type 3:** Rules of the form
 
$$A \rightarrow Bx, \quad A, B \in \Sigma_N, \quad x \in \Sigma^* \wedge B < A$$
  - To do this,
    1. We replace B with all the right-hand sides for B.
    2. This process is repeated for all the type-3 rules.
    3. If, during this process, there appear new left-recursive rules, we transform them using the procedure seen before.
    4. If there appear inaccessible symbols, we eliminate them.

70

## Greibach Normal Form

### Example of step 5

$$G' = \langle \{A, B, C\}, \{a, b\} \rangle$$

$$\{A \rightarrow BC, B \rightarrow CA \mid a, C \rightarrow AB \mid b\},$$

$$A \rangle$$

- Type 3 rules:
  - $C \rightarrow AB$
  - We substitute A with its right-hand sides (BC):  $C \rightarrow BCB$

$$G' = \langle \{A, B, C\}, \{a, b\} \rangle$$

$$\{A \rightarrow BC, B \rightarrow CA \mid a, C \rightarrow BCB \mid b\},$$

$$A \rangle$$

71

## Greibach Normal Form

### Example of step 5

- Now, there is a new type-3 rule:
  - $C \rightarrow BCB$
  - We substitute B with its left-hand sides (CA | a):  $C \rightarrow CACB \mid aCB$

$$G' = \langle \{A, B, C\}, \{a, b\} \rangle$$

$$\{A \rightarrow BC, B \rightarrow CA \mid a, C \rightarrow CACB \mid aCB \mid b\},$$

$$A \rangle$$

72

## Greibach Normal Form

### Proof of the theorem

6. At this point, there should not be more type-3 rules. The next step will be the removal of 2-type rules, starting with the non-terminals which are at the end of the partial ordering.

- **Type 2:** Rules of the following form:

$$A \rightarrow Bx, \quad A, B \in \Sigma_N, \quad x \in \Sigma^* \wedge A < B$$

- To do this,
  1. B will be replaced by all the right-hand sides of the rules for B.
  2. This is repeated until we do not have any more type-2 or type-3 rules.
  3. If there appear new left-recursive rules, they will also be removed.
  4. Inaccessible symbols will also be removed.

73

## Greibach Normal Form

### Example of step 6

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \\ \{E \rightarrow TE', \quad E' \rightarrow +TE' \mid \lambda, \quad T \rightarrow FT', \quad T' \rightarrow *FT' \mid \lambda, \quad F \rightarrow i\}, \\ E \rangle$$

- There are two type-2 rules, with the ordering  $E < T < F$ 
  - $E \rightarrow TE'$
  - $T \rightarrow FT'$
- Firstly, we substitute  $F$  in  $T \rightarrow FT'$  with its right-hand sides ( $i$ ), and obtain:

$$T \rightarrow iT'$$

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \\ \{E \rightarrow TE', \quad E' \rightarrow +TE' \mid \lambda, \quad T \rightarrow iT', \quad T' \rightarrow *FT' \mid \lambda, \quad F \rightarrow i\}, \\ E \rangle$$

74

## Greibach Normal Form

### Example of step 6

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \\ \{E \rightarrow TE', \quad E' \rightarrow +TE' \mid \lambda, \quad T \rightarrow iT', \quad T' \rightarrow *FT' \mid \lambda, \quad F \rightarrow i\}, \\ E \rangle$$

- Next, we substitute  $T$  in  $E \rightarrow TE'$  with its right-hand sides, ( $iT'$ ). We obtain the rule

$$E \rightarrow iT'E'$$

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \\ \{E \rightarrow iT'E', \quad E' \rightarrow +TE' \mid \lambda, \quad T \rightarrow iT', \quad T' \rightarrow *FT' \mid \lambda, \quad F \rightarrow i\}, \\ E \rangle$$

75

## Greibach Normal Form

### Proof of the theorem

7. Now, all the rules belong to **type 1**: they all have the following form:

$$A \rightarrow ax, \quad a \in \Sigma_T, \quad x \in \Sigma^*$$

- The only different with respect to Greibach Normal Form may be due to rules having more than one terminal symbol in the right-hand side.
- This can be solved with a trivial substitution, adding a new non-terminal symbol, as in the following example.

$$A \rightarrow abc$$

We can replace that rule with:

$$A \rightarrow aBC, \quad B \rightarrow b$$

- Where B is a new non-terminal symbol

76

## Greibach Normal Form

### Example of step 7

$$G' = \langle \{A, X\}, \{a, b\} \rangle$$

$$\{A \rightarrow \underline{b}aX \mid aX, X \rightarrow \underline{b}aX \mid \lambda\},$$

$$A \rangle$$

- There are two rules in which the terminal symbol a has to be replaced by the new non-terminal symbol  $A'$  :
  - We add the new rule  $A' \rightarrow a$
  - We substitute  $A \rightarrow \underline{b}aX$  by  $A \rightarrow \underline{b}A'X$
  - We substitute  $X \rightarrow \underline{b}aX$  by  $X \rightarrow \underline{b}A'X$

$$G' = \langle \{A, X, A'\}, \{a, b\} \rangle$$

$$\{A \rightarrow \underline{b}A'X \mid aX, X \rightarrow \underline{b}A'X \mid \lambda, A' \rightarrow a\},$$

$$A \rangle$$

77

## Greibach Normal Form

### Proof of the theorem

8. The last thing to do is the treatment of lambda-rules.

- In Greibach Normal Form, these rules are forbidden. The only exception is when the language contains the empty word, in which there has to be, necessarily, a lambda rule for the axiom of the grammar.
- They should be removed using the procedure previously studied.

**NOTE:** For the purpose of building LL(1) compilers, the grammar needs not be exactly in Greibach Normal form:

- Some of the lambda rules will not be wrong for an LL(1) grammar.
- Otherwise, they will be removed using the procedure already seen.
- When these rules are removed, sometimes it is difficult to comply with all the conditions for LL(1) grammars.
  - In this case, it may be necessary to alter the grammar manually to obtain an equivalent one which can be restated as an LL(1) grammar.

78

## Greibach Normal Form

### Example 1

1. As the language does not contain the empty word, there is nothing to do.
2. Remove all the left-recursive rules:

$$G = \langle \{E, T, F\}, \{-, *, i\} \rangle$$

$$\{E \rightarrow E+T \mid T$$

$$T \rightarrow T*F \mid F$$

$$F \rightarrow i\},$$

$$E \rangle$$

$$A \rightarrow \underline{\alpha_1} \mid \dots \mid \underline{\alpha_n} \mid \beta_1 \mid \dots \mid \beta_m$$

$$A \rightarrow \beta_1 X \mid \dots \mid \beta_m X$$

$$X \rightarrow \underline{\alpha_1} X \mid \dots \mid \underline{\alpha_n} X \mid \lambda$$

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \rangle$$

$$\{E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \lambda$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \lambda$$

$$F \rightarrow i\},$$

$$E \rangle$$

79

## Greibach Normal Form

### Example 1

3. We establish the partial ordering for the non-terminal symbols. In general, we can establish the ordering using the non-terminal symbols in the original grammar, and the ones added in step 2 may be added to the ordering if necessary.

$$G = \langle \{E, T, F\}, \{-, *, i\} \rangle$$

$$\{E \rightarrow E+T \mid T, T \rightarrow T*F \mid F, F \rightarrow i\},$$

$$E \rangle$$

- The following ordering will be deduced from the previous rules:
  - From  $E \rightarrow E+T$  we do not deduce anything
  - From  $E \rightarrow T$  we deduce  $E < T$
  - From  $T \rightarrow T*F$  we deduce nothing
  - From  $T \rightarrow F$  we deduce  $T < F$
  - From  $F \rightarrow i$  we can't deduce anything.
- In conclusion,

$$E < T < F$$

80

## Greibach Normal Form

### Example 1

4. The rules will be classified in three groups.
5. The grammar does not contain type-3 rules, so there is nothing to be done

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \\ \{E \rightarrow TE', E' \rightarrow +TE' \mid \lambda, T \rightarrow FT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\}, \\ E \rangle$$

- Type-1 rules
  - $E' \rightarrow +TE'$
  - $T' \rightarrow *FT'$
  - $F \rightarrow i$
- Type-2 rules
  - $E \rightarrow TE'$
  - $T \rightarrow FT'$
- No type-3 rules.
- Lambda rules will be treated at the end.

81

## Greibach Normal Form

### Example 1

6. Type-2 rules removal:

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \\ \{E \rightarrow TE', E' \rightarrow +TE' \mid \lambda, T \rightarrow FT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\}, \\ E \rangle$$

- There are two type-2 rules, with the ordering  $E < T < F$ 
  - $E \rightarrow TE'$
  - $T \rightarrow FT'$
- We first solve  $F$  in  $T \rightarrow FT'$  and replace it by its right-hand sides ( $i$ ), and obtain

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \\ \{E \rightarrow TE', E' \rightarrow +TE' \mid \lambda, T \rightarrow iT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\}, \\ E \rangle$$

82

## Greibach Normal Form

### Example 1

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \\ \{E \rightarrow TE', E' \rightarrow +TE' \mid \lambda, T \rightarrow iT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\}, \\ E \rangle$$

- Next, we delete  $T$  in  $E \rightarrow TE'$  and replace it with its right-hand sides ( $iT'$ ), to obtain

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\} \\ \{E \rightarrow iT'E', E' \rightarrow +TE' \mid \lambda, T \rightarrow iT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\}, \\ E \rangle$$

83

## Greibach Normal Form

### Example 1

7. As there are no terminal symbols incorrectly set in the right-hand sides of the rules, there is nothing to do at this step.

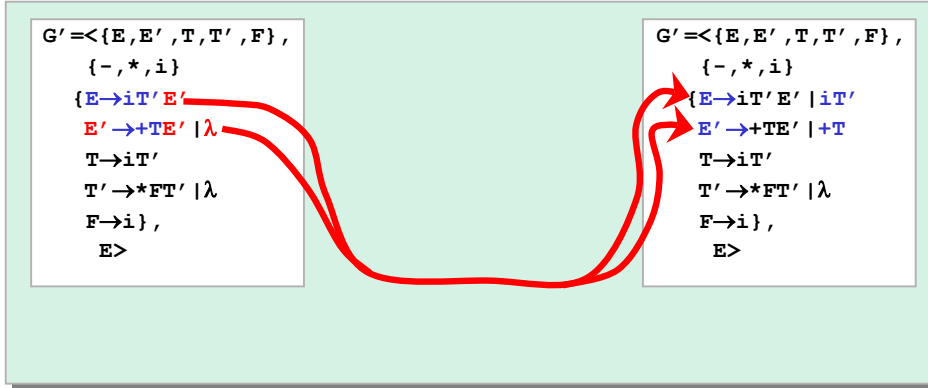
84

## Greibach Normal Form

### Example 1

8. In the case that we want to obtain the grammar in Greibach Normal form, the last step would be to remove all the lambda-rules.

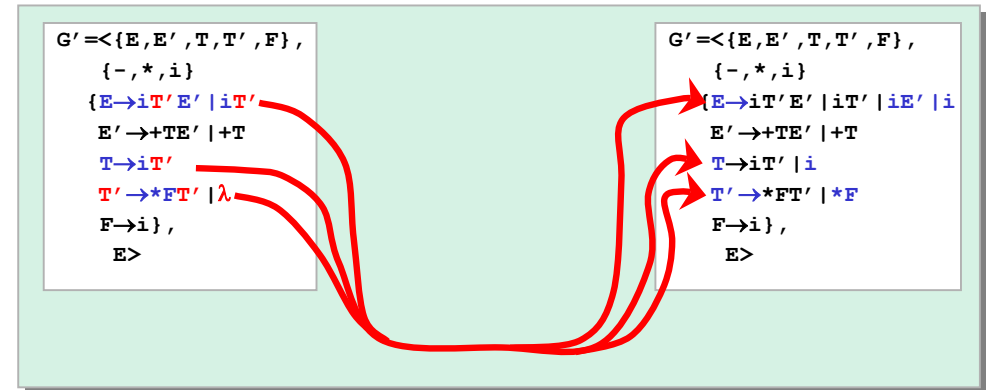
REMEMBER: this step is not strictly necessary for LL(1) parsers



85

## Greibach Normal Form

### Example 1



When there are no more lambda-rules, we have the grammar, finally, in GNF.

86

## Top-down analysis

### Overview

- **Top-down analysis**
  - Blind search, depth-first search and width-first search.
  - Slow backtracking
- **Top-down analysis with LL(1) grammars**
  - **Procedures for modifying grammars**
    - Elimination of left-recursion
    - Elimination of lambda-rules
  - **Greibach Normal Form**
    - Definition
    - Procedure for obtaining the GNF of a grammar
    - Examples
  - **LL(1) grammars**
    - Initial examples
    - Definition of LL(1) grammars
    - Syntactic analysers based on LL(1) grammars

87

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Changes in step 8

- Some lambda rules are not incorrect in LL(1) grammars.
- Therefore, we are only going to remove those that are not right.
- The following procedure will be followed:
  - When a non-terminal has a lambda rule, e.g.
 
$$X \rightarrow \lambda$$
  - The following situation may arise: that the same non-terminal has other rules with a different right-hand side. Because we are in step 8, we know that all rules are now type-1 rules, starting with a terminal symbol.

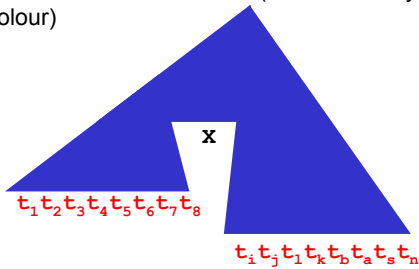
$$X \rightarrow a\alpha, \quad a \in \Sigma_T \wedge \alpha \in \Sigma^*$$

88

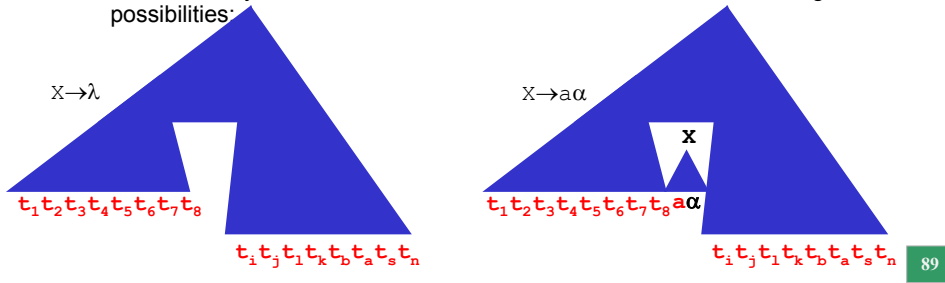
## Adjusting a grammar in Greibach Normal Form to LL(1)

### Changes in step 8

- Imagine a syntactic derivation tree for a word (the terminal symbols are represented in red colour)



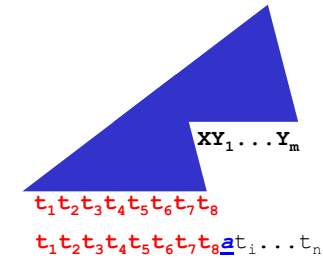
- If X is the only non-terminal that remains, there would be the following two possibilities:



## Adjusting a grammar in Greibach Normal Form to LL(1)

### Changes in step 8

- Imagine that we are doing the top-down parsing of that word, and we are just before the symbol **a**



- Intuitively, the efficiency of LL(1), which is better than simple “top-down parsing with slow backtracking”, is due to the indexation of the right-hand sides, for each non-terminal, using the next terminal to be analysed.
- This will be possible only if each non-terminal has just one right-hand side starting with each terminal.
- If we do not have lambda rules ( $X \rightarrow \lambda$ ), the top-down analysis can be done completely without any branching in the analysis.

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Changes in step 8

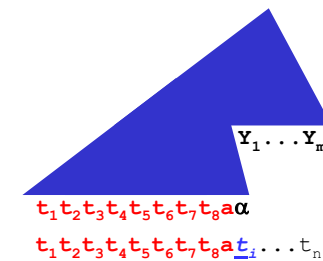
- If we had the following rules for a non-terminal X,

$$\begin{aligned}
 X &\rightarrow t_1 \beta_1 \\
 &\dots \\
 X &\rightarrow t_{i-1} \beta_{i-1} \\
 X &\rightarrow \mathbf{a} \alpha \\
 X &\rightarrow t_{i+1} \beta_{i+1} \\
 &\dots \\
 X &\rightarrow t_k \beta_k \\
 \text{,, } \{t_1, \dots, t_{i-1}, a, t_{i+1}, \dots, t_k\} &\subseteq \Sigma_T \\
 \wedge t_p \neq t_q \neq a \quad \forall p, q \in \{1, \dots, i-1, i+1, \dots, k\} \\
 \wedge \{\beta_1, \dots, \beta_{i-1}, a, \beta_{i+1}, \dots, \beta_k\} &\subseteq \Sigma_N^*
 \end{aligned}$$

- There would be just one possibility to choose if the next symbol in the input string were “a”.

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Changes in step 8

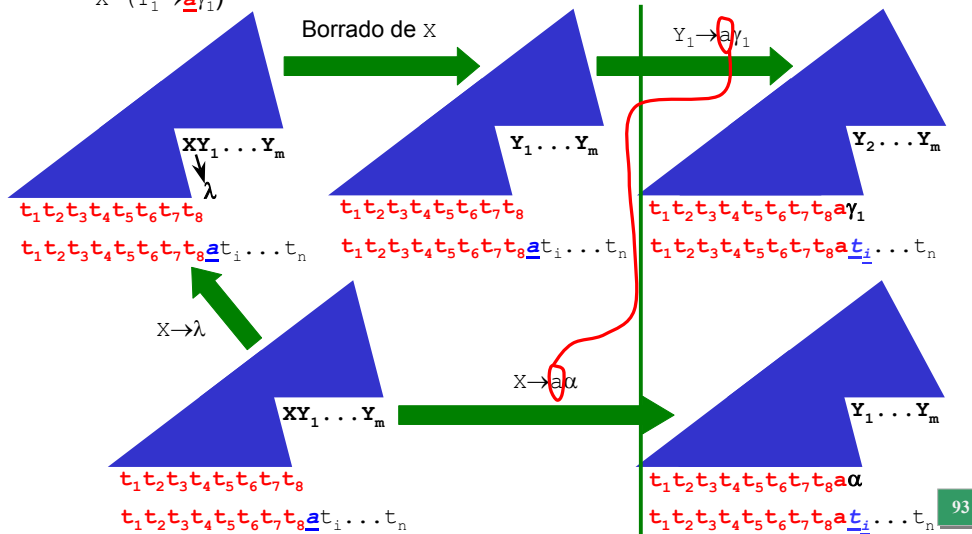


- Lambda-rules will be problematic whenever they produce several possibilities of choosing the next rule to apply.

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Changes in step 8

- Let us imagine that we have the following three rules:  $X \rightarrow a\alpha$ ,  $X \rightarrow \lambda$  y  $Y_1 \rightarrow a\gamma_1$
- A terminal "a" can be derived directly from X ( $X \rightarrow \underline{a}\alpha$ ), and from  $Y_1$ , which follows X ( $Y_1 \rightarrow \underline{a}\gamma_1$ )

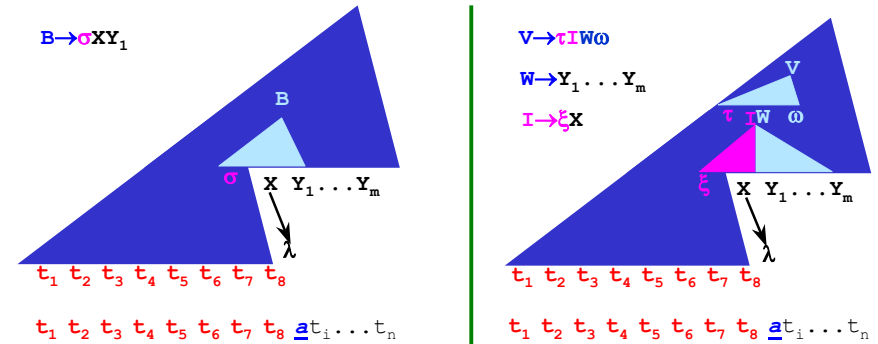


93

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Changes in step 8

- In order to identify these situations, we need to know:
  - Which are the first terminals derived by X (in our example, a)
  - Which is the first terminal that can be derived by what is after X, i.e. next(X)
- The following figures describe the two possible cases:



94

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Step 8 for example 1

8. If we want to generate a top-down syntactic analyser with the LL(1) technique, we have to study, in step 8, which lambda-rules produce ambiguities.

$G' = \langle \{E, E', T, T', F\}, \{-, *, i\}, \{E \rightarrow iT'E', E' \rightarrow +TE' | \lambda, T \rightarrow iT', T' \rightarrow *FT' | \lambda, F \rightarrow i\}, E \rangle$

Who can follow  $E'$ ? next( $E'$ )

We study all the right-hand sides that contain  $E'$

- Let's start with the lambda-rule  $E' \rightarrow \lambda$ .

95

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Step 8 for example 1

8. (cont.)

$G' = \langle \{E, E', T, T', F\}, \{-, *, i\}, \{E \rightarrow iT'E', E' \rightarrow +TE' | \lambda, T \rightarrow iT', T' \rightarrow *FT' | \lambda, F \rightarrow i\}, E \rangle$

Who can follow  $E'$ ? next( $E'$ )

We study all the right-hand sides that contain  $E'$

In the two rules,  $E'$  appears as the last symbol in the right-hand side.

$E'$  can be followed by anything that follows the left-hand side of those rules:

- $E'$
- $E$

96

## Adjusting a grammar in Greibach Normal Form to LL(1)

Step 8 for example 1

8. (cont.)

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\}, \{E \rightarrow iT'E', E' \rightarrow +TE' \mid \lambda, T \rightarrow iT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\}, E \rangle$$

Therefore:

- next(E') is included in next(E') – obvious
- next(E) is included in next(E'). As E is the axiom, and it does not appear in any other right-hand side. next(E) is the end-of-program symbol, \$.

So next(E') is {\$}.

On the other hand, first(E') = {+}

- We can conclude that the first lambda rule is correct for LL(1), and can be left like that.

97

## Adjusting a grammar in Greibach Normal Form to LL(1)

Step 8 for example 1

8. (cont.)

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\}, \{E \rightarrow iT'E', E' \rightarrow +TE' \mid \lambda, T \rightarrow iT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\}, E \rangle$$

Who can follow T' ? next(T')

We study all the right-hand sides that contain T'

- Let us continue with the second lambda rule.

98

## Adjusting a grammar in Greibach Normal Form to LL(1)

Step 8 for example 1

8. (cont.)

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\}, \{E \rightarrow iT'E', E' \rightarrow +TE' \mid \lambda, T \rightarrow iT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\}, E \rangle$$

Who can follow T' ? next(T')

We study all the right-hand sides that contain T'

There is one rule in which T' is followed by E'. In two other rules, it appears as the last symbol of the right-hand side.

Therefore, next(T') will be:

- first(E')
- next(T)
- next(T') -- obvious

99

## Adjusting a grammar in Greibach Normal Form to LL(1)

Step 8 for example 1

8. (cont.)

$$G' = \langle \{E, E', T, T', F\}, \{-, *, i\}, \{E \rightarrow iT'E', E' \rightarrow +TE' \mid \lambda, T \rightarrow iT', T' \rightarrow *FT' \mid \lambda, F \rightarrow i\}, E \rangle$$

We can focus now on next(T)

Let us see all the rules in which T appears in the right-hand side.

100

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Step 8 for example 1

8. (cont.)

$$G' = \langle \{E, E', T, T', F\}, \{ -, *, i \}, \{ E \rightarrow iT'E' \}, \{ E' \rightarrow +TE' \mid \lambda \}, \{ T \rightarrow iT' \}, \{ T' \rightarrow *FT' \mid \lambda \}, \{ F \rightarrow i \}, E \rangle$$

We can focus now on **next(T)**

Let us see all the rules in which **T** appears in the right-hand side.

It only appears followed by **E'**.  
Therefore,  $\text{next}(T) = \text{first}(E')$

In conclusion,  $\text{next}(T')$  will be:

- $\text{first}(E')$
- $\text{next}(T) = \text{first}(E')$
- $\text{next}(T')$  -- obvious

- Therefore,  $\text{next}(T') = \{+\}$

101

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Step 8 for example 1

8. (cont.)

$$G' = \langle \{E, E', T, T', F\}, \{ -, *, i \}, \{ E \rightarrow iT'E' \}, \{ E' \rightarrow +TE' \mid \lambda \}, \{ T \rightarrow iT' \}, \{ T' \rightarrow *FT' \mid \lambda \}, \{ F \rightarrow i \}, E \rangle$$

Finally, we need to check whether the lambda rule produces any ambiguity during the analysis.

For instance,

- The first terminal derived by **T'** is **\***.
- The terminals in  $\text{next}(T')$  are just **+**.

- Because the terminals are different (**\*** y **+**) we can conclude that the lambda rule is appropriate for an LL(1) grammar.

102

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2

- Obtain, if possible, the LL(1) equivalent grammar from the following one:

$$G_2 = \langle \{A, B\}, \{a, b\}, \{A \rightarrow Ba \mid a\}, \{B \rightarrow Ab \mid b\}, A \rangle$$

103

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2

1. The language does not contain the empty word, so there is nothing to do.
2. There are no left-recursive rules, so there is again nothing to do here.
3. We shall establish the partial ordering between the non-terminal symbols.

$$G_2 = \langle \{A, B\}, \{a, b\}, \{A \rightarrow Ba \mid a\}, \{B \rightarrow Ab \mid b\}, A \rangle$$

- From the previous rules, we can deduce two different partial orderings:
  - From  $A \rightarrow Ba$  we can deduce  $A < B$
  - From  $B \rightarrow Ab$  we can deduce  $B < A$
- In summary, there are two options:
  - Option 1:  $B < A$
  - Option 2:  $A < B$
- We shall study the two possibilities separately, to see how the choice taken affects the result.

104

## Adjusting a grammar in Greibach Normal Form to LL(1)

Example 2, option 1 B<A

4. The rules will be classified in three groups

$$G_2 = \langle \{A, B\}, \{a, b\}, \{A \rightarrow Ba \mid a, B \rightarrow Ab \mid b\}, A \rangle$$

- Type 1 rules
  - $A \rightarrow a$
  - $B \rightarrow b$
- Type 2 rules
  - $B \rightarrow Ab$
- Type 3 rules
  - $A \rightarrow Ba$

105

## Adjusting a grammar in Greibach Normal Form to LL(1)

Example 2, option 1 B<A

5. The type-3 rule will be eliminated by substituting B

$$G_2 = \langle \{A, B\}, \{a, b\}, \{A \rightarrow Ba \mid a, B \rightarrow Ab \mid b\}, A \rangle$$

- Type-3 rules:
  - $A \rightarrow Ba$
- The B will be replaced by its right-hand sides (Ab and b), and we obtain:
  - $A \rightarrow Aba \mid ba$

$$G_2 = \langle \{A, B\}, \{a, b\}, \{A \rightarrow Aba \mid ba \mid a, B \rightarrow Ab \mid b\}, A \rangle$$

106

## Adjusting a grammar in Greibach Normal Form to LL(1)

Example 2, option 1 B<A

- In all the steps, we need to check whether there is any inaccessible symbol.

$$G_2 = \langle \{A, B\}, \{a, b\}, \{A \rightarrow Aba \mid ba \mid a, B \rightarrow Ab \mid b\}, A \rangle$$

- It can be easily seen that, in this grammar, B cannot be generated from the axiom.
- It is inaccessible, so we can eliminate it and all the rules where it is at the left-hand side:
  - $B \rightarrow Ab \mid b$

$$G_2 = \langle \{A\}, \{a, b\}, \{A \rightarrow Aba \mid ba \mid a\}, A \rangle$$

107

## Adjusting a grammar in Greibach Normal Form to LL(1)

Example 2, option 1 B<A

- We also need to eliminate all the left-recursive rules that have turned up:

$$G_2 = \langle \{A\}, \{a, b\}, \{A \rightarrow Aba \mid ba \mid a\}, A \rangle$$

- The rule  $A \rightarrow Aba$  is left-recursive, so we apply the lemma which said:
  - If we have a grammar  $\langle \Sigma_T, \Sigma_N, S, P \rangle$ , the left-recursive rules
    - $A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \dots \mid \beta_m$
  - will be substituted by
    - $A \rightarrow \beta_1 X \mid \dots \mid \beta_m X$
    - $X \rightarrow \alpha_1 X \mid \dots \mid \alpha_n X \mid \lambda$
- In our example,
  - $A \rightarrow baX \mid aX$
  - $X \rightarrow baX \mid \lambda$

$$G_2 = \langle \{A, X\}, \{a, b\}, \{A \rightarrow baX \mid aX, X \rightarrow baX \mid \lambda\}, A \rangle$$

108

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 1 B<A

6. There no type-2 rules to eliminate
7. We eliminate all the terminal symbols that are not in the first position of the rules:

$$G_2 = \langle \{A, X\}, \{a, b\} \rangle$$

$$\{A \rightarrow b \underline{a} X \mid aX$$

$$X \rightarrow b \underline{a} X \mid \lambda\},$$

$$A \rangle$$

- We define a new non-terminal  $Z$  to derive the 'a' ( $Z \rightarrow a$ )
- And we substitute the 'a' in all the right-hand sides by 'Z': ( $A \rightarrow b \underline{Z} X$  y  $X \rightarrow b \underline{Z} X$ )

$$G_2 = \langle \{A, X\}, \{a, b\} \rangle$$

$$\{A \rightarrow b \underline{Z} X \mid aX$$

$$X \rightarrow b \underline{Z} X \mid \lambda$$

$$Z \rightarrow a\},$$

$$A \rangle$$

109

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 1 B<A

8. Study of the  $\lambda$ -rules

$$G_2 = \langle \{A, X\}, \{a, b\} \rangle$$

$$\{A \rightarrow bZ X \mid aX$$

$$X \rightarrow bZ X \mid \lambda$$

$$Z \rightarrow a\},$$

$$A \rangle$$

There is just one lambda-rule, for X.

Let us study the set  $\text{next}(X)$ , to see whether it is different from  $\text{first}(X)$ .

110

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 1 B<A

8. (cont.)

$$G_2 = \langle \{A, X\}, \{a, b\} \rangle$$

$$\{A \rightarrow bZ X \mid aX$$

$$X \rightarrow bZ X \mid \lambda$$

$$Z \rightarrow a\},$$

$$A \rangle$$

Concerning  $\text{next}(X)$ , it appears in three rules. From that, we can conclude that  $\text{next}(X)$  contains:

- $\text{next}(A)$
- $\text{next}(X)$  - obvious

Because A is the axiom, and it does not appear in any right-hand side, we can conclude that  $\text{next}(A)$  will be the end-of-program symbol  $\{\$, \}$ , and hence  $\text{next}(X)$  will also be  $\{\$, \}$ .

On the other hand,  $\text{first}(X) = \{b\}$

- This lambda-rule is not wrong for an LL(1) grammar.

111

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 A<B

4. Let us see now

$$G_2 = \langle \{A, B\}, \{a, b\} \rangle$$

$$\{A \rightarrow Ba \mid a$$

$$B \rightarrow Ab \mid b\},$$

$$A \rangle$$

- Type-1 rules:
  - $A \rightarrow a$
  - $B \rightarrow b$
- Type-2 rules:
  - $A \rightarrow Ba$
- Type-2 rules:
  - $B \rightarrow Ab$

112

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 A<B

5. We start by eliminating the type-3 rules from the grammar:

$$G_2 = \langle \{A, B\}, \{a, b\}, \{A \rightarrow Ba \mid a, B \rightarrow Ab \mid b\}, A \rangle$$

- Type-3 rules:
  - $B \rightarrow Ab$
- We replace  $A$  with its right-hand sides ( $Ba$  and  $a$ ) to get:
  - $B \rightarrow Bab \mid ab \mid b$

$$G_2 = \langle \{A, B\}, \{a, b\}, \{A \rightarrow Ba \mid a, B \rightarrow Bab \mid ab \mid b\}, A \rangle$$

113

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 A<B

- Now, we need to eliminate left-recursive rules.

$$G_2 = \langle \{A, B\}, \{a, b\}, \{A \rightarrow Ba \mid a, B \rightarrow Bab \mid ab \mid b\}, A \rangle$$

- The rule  $B \rightarrow Bab$  is left-recursive, so we apply the lemma.
- In the example, we get:
  - $B \rightarrow abX \mid bX$
  - $X \rightarrow abX \mid \lambda$

$$G_2 = \langle \{A, B, X\}, \{a, b\}, \{A \rightarrow Ba \mid a, B \rightarrow abX \mid bX, X \rightarrow abX \mid \lambda\}, A \rangle$$

114

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 A<B

6. Remove type-2 rules:

$$G_2 = \langle \{A, B, X\}, \{a, b\}, \{A \rightarrow Ba \mid a, B \rightarrow abX \mid bX, X \rightarrow abX \mid \lambda\}, A \rangle$$

- There is one type-2 rule:
  - $A \rightarrow Ba$
- We substitute  $B$  with all its right-hand sides ( $abX$  y  $bX$ ) to get:

$$G_2 = \langle \{A, B, X\}, \{a, b\}, \{A \rightarrow abXa \mid bXa \mid a, B \rightarrow abX \mid bX, X \rightarrow abX \mid \lambda\}, A \rangle$$

115

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 A<B

- As before, we need to check whether there is any inaccessible symbol:

$$G_2 = \langle \{A, B, X\}, \{a, b\}, \{A \rightarrow abXa \mid bXa \mid a, B \rightarrow abX \mid bX, X \rightarrow abX \mid \lambda\}, A \rangle$$

- Now, it is not possible to arrive to  $B$  from the axiom, so we can delete it and all the rules that have it in the left-hand side:
  - $B \rightarrow abX \mid bX$

$$G_2 = \langle \{A, X\}, \{a, b\}, \{A \rightarrow abXa \mid bXa \mid a, X \rightarrow abX \mid \lambda\}, A \rangle$$

116

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 A<B

#### 7. Removal of all the terminals that are not in the first position.

$$G_2 = \langle \{A, X\}, \{a, b\} \rangle$$

$$\{A \rightarrow a \underline{b} X a \mid b X \underline{a} \mid a$$

$$X \rightarrow a \underline{b} X \mid \lambda\},$$

$$A \rangle$$

- First, we can eliminate the symbol  $\underline{b}$ : we define a new non-terminal  $B$  to derive it ( $B \rightarrow b$ ), and we put the non-terminal in the right-hand sides ( $A \rightarrow a \underline{B} X a$  y  $X \rightarrow a \underline{B} X$ )

$$G_2 = \langle \{A, B, X\}, \{a, b\} \rangle$$

$$\{A \rightarrow a \underline{B} X a \mid b X \underline{a} \mid a$$

$$X \rightarrow a \underline{B} X \mid \lambda$$

$$B \rightarrow b\},$$

$$A \rangle$$

117

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 A<B

#### 7. (cont.)

$$G_2 = \langle \{A, B, X\}, \{a, b\} \rangle$$

$$\{A \rightarrow a B X \underline{a} \mid b X \underline{a} \mid a$$

$$X \rightarrow a B X \mid \lambda$$

$$B \rightarrow b\},$$

$$A \rangle$$

- We also have a misplaced terminal  $\underline{a}$ : we define a new non-terminal  $Z$  to derive it ( $Z \rightarrow a$ ), and we substitute it in the problematic right-hand sides ( $A \rightarrow a B X \underline{Z} \mid b X \underline{Z}$ )

$$G_2 = \langle \{A, B, X, Z\}, \{a, b\} \rangle$$

$$\{A \rightarrow a B X \underline{Z} \mid b X \underline{Z} \mid a$$

$$X \rightarrow a B X \mid \lambda$$

$$B \rightarrow b$$

$$Z \rightarrow a\},$$

$$A \rangle$$

118

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 A<B

#### 8. Study of the $\lambda$ -rules.

$$G_2 = \langle \{A, B, X, Z\}, \{a, b\} \rangle$$

$$\{A \rightarrow a B X Z \mid b X Z \mid a$$

$$X \rightarrow a B X \mid \lambda$$

$$B \rightarrow b$$

$$Z \rightarrow a\},$$

$$A \rangle$$

We need to check whether  $\text{next}(X)$  and  $\text{first}(X)$  have any terminal symbol in common.

- There is just one lambda-rule.

119

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 A<B

#### 8. (cont.)

$$G_2 = \langle \{A, B, X, Z\}, \{a, b\} \rangle$$

$$\{A \rightarrow a B X Z \mid b X Z \mid a$$

$$X \rightarrow a B X \mid \lambda$$

$$B \rightarrow b$$

$$Z \rightarrow a\},$$

$$A \rangle$$

Let us calculate  $\text{next}(X)$ :

- $X$  appears followed by  $\underline{Z}$ .
- $X$  also appears at the end of a rule.

Therefore,  $\text{next}(X)$  will contain:

- $\text{first}(Z) = a$
- $\text{next}(X)$  - obvious

120

## Adjusting a grammar in Greibach Normal Form to LL(1)

### Example 2, option 2 $A < B$

8. (cont.)

$$G_2 = \langle \{A, B, X, Z\}, \{a, b\} \rangle$$

$$\{A \rightarrow aBXZ \mid bXZ \mid a$$

$$X \rightarrow aBX \mid \lambda$$

$$B \rightarrow b$$

$$Z \rightarrow a\},$$

$$A \rangle$$

Let us calculate next(X):

- X appears followed by z.
- X also appears at the end of a rule.

Therefore, next(X) will contain:

- first(Z) = a
- next(X) – obvious

But it is the case that:

- first(X) is the symbol a
- next(X) = first(Z) = a

- So we can conclude that this lambda-rule produces ambiguities during the parsing.

121

## LL(1)

### Concept

- LL(1) languages are

Those whose grammars appear in **Greibach Normal Form**, and there are not two rules for the same non-terminal, starting with the same terminal symbol in their right-hand side.

122

## LL(1)

### Converting from GNF to LL(1)

- A grammar in Greibach Normal Form may not be LL(1)
- For instance,

$$G_2 = \langle \{U, V, W, X, Y, Z, T\}, \{a, b, c, d, e\} \rangle$$

$$\{ \dots$$

$$U \rightarrow aV \mid aW$$

$$V \rightarrow bX \mid cY$$

$$W \rightarrow dZ \mid eT$$

$$\dots \},$$

$$U \rangle$$

- Two of the right-hand sides for  $U$  start with the same terminal symbol, a.

123

## LL(1)

### Converting from GNF to LL(1)

- Sometimes, this can be solved with a "common factor".
- Steps
  - Obtain the longest common prefix of the two right-hand sides (in the example the symbol a).
  - Leave a single right-hand side which starts with that common prefix, and which ends with a new non-terminal, e.g.  $K$

$$G_2 = \langle \{U, V, W, X, Y, Z, T\}, \{a, b, c, d, e\} \rangle$$

$$\{ \dots$$

$$U \rightarrow aK$$

$$\dots$$

$$V \rightarrow bX \mid cY$$

$$W \rightarrow dZ \mid eT$$

$$\dots \}, \quad U \rangle$$

124

## LL(1)

### Converting from GNF to LL(1)

- The new non-terminal will generate the remaining of the right-hand sides,  $K \rightarrow \underline{v} | w$ .

```
G2 = <{U, V, W, X, Y, Z, T},  
      {a, b, c, d, e}  
      { ...  
      U → aK  
      K → V | W  
      V → bX | cY  
      W → dZ | eT  
      ... }, U >
```

125

## LL(1)

### Converting from GNF to LL(1)

- We have to take care to leave the rules again in GNF...
  - We can derive the initial non-terminals in the rules for K, so they start with a non-terminal.
  - In this case, we can apply the rules for V and W:

```
G2 = <{U, V, W, X, Y, Z, T},  
      {a, b, c, d, e}  
      { ...  
      U → aK  
      K → bX | cY | dZ | eT  
      V → bX | cY  
      W → dZ | eT  
      ... }, U >
```

126

## LL(1)

### Converting from GNF to LL(1)

- ...and we have to remove the inaccessible symbols (v y w)

```
G2 = <{U, V, W, X, Y, Z, T},  
      {a, b, c, d, e}  
      { ...  
      U → aK  
      K → bX | cY | dZ | eT  
      ...  
      },  
      U >
```

127

## Formalisation of LL(1) grammars

### Concept

- In the remaining part of this lesson, we are going to describe formally LL(1) analysers, which we have already introduced informally with examples.

128

## Formalisation of LL(1) grammars

### Example

- Consider the following grammar, and the first and next sets for each non-terminal:

```
G=< {E,E',T,T',F},
    {+,*,(,),id}
    {
      E  → TE'
      E' → +TE' | λ
      T  → FT'
      T' → *FT' | λ
      F  → (E) | id
    },
E>
```

**first(E)=first(T)=first(F)={ (, id}**  
**first(E')={+, λ}**  
**first(T')={\*, λ}**  
**next(E)=next(E')={, \$}**  
**next(T)=next(T')={+, , \$}**  
**next(F)={+, \*, , \$}**

129

## Constructing LL(1) analysers

### Algorithm

- The following algorithm calculates the analysis table  $T \in M_{|\Sigma_N| \times |\Sigma_T|+1}$
- In this matrix, there will be a row for each non-terminal, and a column for each terminal, including the end-of-program symbol \$ :

- $\forall A \rightarrow \alpha \in P$  repeat:
  - $\forall a \in \text{first}(\alpha) \cap \Sigma_T \quad A \rightarrow \alpha \in T[A, a]$ .
  - If  $\lambda \in \text{first}(\alpha)$ 
    - then,  $\forall b \in \text{next}(A) \quad A \rightarrow \alpha \in T[A, b]$  (note that  $b$  can also be \$)

130

## Constructing LL(1) analysers

### Examples

- In the grammar for the previous example

$\Sigma_N$	$\Sigma_T \cup \{\$\}$					
	id	+	*	(	)	\$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → λ	E' → λ
T	E → FT'			E → FT'		
T'		T' → λ	T' → *FT'		T' → λ	T' → λ
F	F → id			F → (E)		

131

## Constructing LL(1) analysers

### Examples

- Given the following grammar,

```
G=< {P,P',E},
    {i,t,a,e,b}
    {
      P  → iEtPP' | a
      P' → eP   | λ
      E  → b
    }
P>
```

132

## Constructing LL(1) analysers

### Examples

- We can obtain the next table:

$\Sigma_N$	$\Sigma_T \cup \{\$ \}$					
	a	b	e	i	t	\$
P	$P \rightarrow a$			$P \rightarrow iEtPP'$		
P'			$P' \rightarrow \lambda$ $P' \rightarrow eP$			$P' \rightarrow \lambda$
E		$E \rightarrow b$				

133

## LL(1), first and next sets

### Definition

- We can define **LL(1) grammars** as those that comply with the following condition:

The analysis table constructed with the previous procedure is deterministic, i.e., all the rows contain at most one rule.

### Examples

- The grammar in the first example is LL(1)
- The grammar in the second example is not LL(1)

134

## Selective top-down analysers

### Concept

- LL(1) analysers are also called “with no backtracking”, because they are deterministic and it will never be necessary to backtrack during the analysis of a program.
- They are also called “recursive-descent” parsers, because of the kind of analysis programs that are produced from LL(1) grammars.
- The reason of the efficiency of these parsers is because the right-hand sides of the rules for each non-terminal symbol can be considered to be indexed by the next terminal.

135

## Selective top-down analysers

### Automatically building a parser for an LL(1) grammar

- According to the procedure described, in an LL(1) grammar we are going to have two kinds of rules:

- Rules for non-terminals that have a  $\lambda$ -rule:**

$$U \rightarrow xX_1X_2 \dots X_n \mid yY_1Y_2 \dots Y_m \mid \dots \mid zZ_1 \dots Z_p \mid \lambda$$

- Rules for non-terminals that do not have a  $\lambda$ -rule:**

$$U \rightarrow xX_1X_2 \dots X_n \mid yY_1Y_2 \dots Y_m \mid \dots \mid zZ_1 \dots Z_p$$

136

## Selective top-down analysers

Automatically building a parser for an LL(1) grammar:  $\lambda$  rules

- The next C function will be generated:

```
int U(char * string, int i)
{
  if (i < 0) return i;
  /* this propagates errors */
  switch (string[i]) {
    case x:
      i++;
      i=X1(string, i);
      i=X2(string, i);
      ...
      i=Xn(string, i);
      break;
    case y:
      i++;
      i=Y1(string, i);
      i=Y2(string, i);
      ...
      i=Ym(string, i);
      break;
    ...
  }
```

```
case z:
  i++;
  i=Z1(string, i);
  i=Z2(string, i);
  ...
  i=Zp(string, i);
  break;
}
return i;
}
```

137

## Selective top-down analysers

Automatically building a parser for an LL(1) grammar: rules without  $\lambda$

- The next C function will be generated:

```
int U(char * string, int i)
{
  if (i < 0) return i;
  /* This propagates errors */
  switch (string[i]) {
    case x:
      i++;
      i=X1(string, i);
      i=X2(string, i);
      ...
      i=Xn(string, i);
      break;
    case y:
      i++;
      i=Y1(string, i);
      i=Y2(string, i);
      ...
      i=Ym(string, i);
      break;
    ...
  }
```

```
case z:
  i++;
  i=Z1(string, i);
  i=Z2(string, i);
  ...
  i=Zp(string, i);
  break;
  /* End-of-string goes to the
  default case */
  default:
    return -n;
  /* The error will be
  different for each rule */
}
return i;
}
```

138

## Selective top-down analysers

Complete example

- Let us start with the following context-independent grammar

```
G3 = <{E, T, F}, {i, +, -, *, /, (, )}
{E → T+E | T-E | T
  T → F*T | F/T | F
  F → i | (E)
},
E>
```

139

## Selective top-down analysers

Complete example

- The following is the grammar in Greibach Normal Form

```
G3 = <{E, T, M, S, P, D, C},
      {i, +, -, *, /, (, )}
{E → iPTME | (ECPTME | iDTME | (ECDTME | iME | (ECME |
      IPTSE | (ECPTSE | iDTSE | (ECDTSE | iSE | (ECSE |
      IPT | (ECPT | iDT | (ECDT | i | (EC
T → IPT | (ECPT | iDT | (ECDT | i | (EC
M → +
S → -
P → *
D → /
C → )}, E>
```

140

## Selective top-down analysers

### Complete example

- The following is a possible LL(1) grammar, obtained from the GNF grammar by taking "common factor" in the right-hand sides of the rules.

```
G3 = <{E, T, M, S, P, D, C},
      {i, +, -, *, /, (, )}
      {E → iV | (ECV
        V → *TX | /TX | +E | -E | λ
        X → +E | -E | λ
        T → iU | (ECU
        U → *T | /T | λ
        C → )}, E>
```

141

## Selective top-down analysers

### Complete example

- The following would be the LL(1) analyser (continued)

```
int E(char * string, int i)
{
  if (i < 0) return i;
  /* This propagates previous
  errors */
  switch (string[i]) {
    case 'i':
      i++;
      i=V(string, i);
      break;
    case '\(':
      i++;
      i=E(string, i);
      i=C(string, i);
      i=V(string, i);
      break;
    default: return -1; /*no λ*/
  }
  return i;
}
```

```
int V(char * string, int i)
{
  if (i < 0) return i;
  /* This propagates previous
  errors */
  switch (string[i]) {
    case '*':
      case '/':
        i++;
        i=T(string, i);
        i=X(string, i);
        break;
    case '+':
      case '-':
        i++;
        i=E(string, i);
        break; /* λ */
  }
  return i;
}
```

142

## Selective top-down analysers

### Complete example

- The following would be the LL(1) analyser (continued)

```
int X(char * string, int i)
{
  if (i < 0) return i;
  /* This propagates previous
  errors */
  switch (string[i]) {
    case '+':
      case '-':
        i++;
        i=E(string, i);
        break; /* λ */
  }
  return i;
}
```

```
int T(char * string, int i)
{
  if (i < 0) return i;
  /* This propagates previous
  errors */
  switch (string[i]) {
    case 'i':
      i++;
      i=U(string, i);
      break;
    case '\(':
      i++;
      i=E(string, i);
      i=C(string, i);
      i=U(string, i);
      break;
    default: return -2; /*no λ*/
  }
  return i;
}
```

143

## Selective top-down analysers

### Complete example

- The following would be the LL(1) analyser (continued)

```
int U(char * string, int i)
{
  if (i < 0) return i;
  /* This propagates previous
  errors */
  switch (string[i]) {
    case '*':
      case '/':
        i++;
        i=T(string, i);
        break; /* λ */
  }
  return i;
}
```

```
int C(char * string, int i)
{
  if (i < 0) return i;
  /* Propagate previous errors
  */
  switch (string[i]) {
    case ')':
      i++;
      break;
    default: return -4; /*no λ*/
  }
  return i;
}
```

144

## Selective top-down analysers

### Complete example

- A string  $x$  would be analysed with the following call:

```
axiom( x, 0);
```

- If the value returned equals the length of the string, it was correct. If it is a negative number, it was incorrect.
- The following execution example illustrates the LL(1) analysis:

```
E("i + i * i", 0)
0[1]: E( "I + I * I", 0)
  1[1]: V( "I + I * I", 1)
    2[1]: E( "I + I * I", 2)
      3[1]: V( "I + I * I", 3)
        4[1]: T( "I + I * I", 4)
          5[1]: U( "I + I * I", 5)
            5[1]: returned 5
          4[1]: returned 5
        4[1]: X( "I + I * I", 5)
        4[1]: returned 5
      3[1]: returned 5
    2[1]: returned 5
  1[1]: returned 5
0[1]: returned 5
5
```

145

## Selective top-down analysers

### Complete example

- The following string will not be recognised

```
E("i + i *", 0)
0[1]: E( "I + I *", 0)
  1[1]: V( "I + I *", 1)
    2[1]: E( "I + I *", 2)
      3[1]: V( "I + I *", 3)
        4[1]: T( "I + I *", 4)
          4[1]: returned -2
        4[1]: X( "I + I *", -2)
        4[1]: returned -2
      3[1]: returned -2
    2[1]: returned -2
  1[1]: returned -2
0[1]: returned -2
-2
```

146

## Syntactic analysis

### Bibliography

- [Alf] "Teoría de Autómatas y lenguajes formales" M. Alfonseca y otros
- [Hop] "Introducción a la teoría de autómatas, lenguajes y computación" Hopcroft, J.; Motwani, R.; Ullman, J.
- [Aho] "Compiladores. Principios, técnicas y herramientas" A. V. Aho; R. Sefthi; J. D. Ullman

147