

Compiler Construction

Lecture 8: Syntactic Analysis IV (Practical Issues in LL Parsing)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University
`noll@cs.rwth-aachen.de`

`http://www-i2.informatik.rwth-aachen.de/i2/cc08/`

Summer semester 2008

- 1 Repetition: $LL(1)$ Grammars
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Error Handling in LL Parsing

Definition (Lookahead set)

Given $\pi = A \rightarrow \beta \in P$,

$$\text{la}(\pi) := \text{fi}(\beta \cdot \text{fo}(A)) \subseteq \Sigma_\varepsilon$$

is called the **lookahead set** of π (where $\text{fi}(\Gamma) := \bigcup_{\gamma \in \Gamma} \text{fi}(\gamma)$).

Corollary

❶ For all $a \in \Sigma$,

$$a \in \text{la}(A \rightarrow \beta) \text{ iff } a \in \text{fi}(\beta) \text{ or } (\beta \Rightarrow^* \varepsilon \text{ and } a \in \text{fo}(A))$$

❷ $\varepsilon \in \text{la}(A \rightarrow \beta)$ iff $\beta \Rightarrow^* \varepsilon$ and $\varepsilon \in \text{fo}(A)$

Characterization of $LL(1)$

Theorem (Characterization of $LL(1)$)

$G \in LL(1)$ iff for all pairs of rules $A \rightarrow \beta \mid \gamma \in P$ (where $\beta \neq \gamma$):

$$\text{la}(A \rightarrow \beta) \cap \text{la}(A \rightarrow \gamma) = \emptyset.$$

Proof.

on the board



Remark: the above theorem generally does not hold if $k > 1$
(cf. exercises)

Deterministic Top-Down Parsing

Approach: given $G \in CFG_{\Sigma}$,

- ① Verify that $G \in LL(1)$ by computing the lookahead sets and checking alternatives for disjointness
 - ② Start with nondeterministic top-down parsing automaton $NTA(G)$
 - ③ Use **1-symbol lookahead** to control the choice of expanding productions:
 - $(aw, A\alpha, z) \vdash (aw, \beta\alpha, zi)$
if $\pi(i) = A \rightarrow \beta$ and $a \in la(\pi(i))$
 - $(\varepsilon, A\alpha, z) \vdash (\varepsilon, \beta\alpha, zi)$
if $\pi(i) = A \rightarrow \beta$ and $\varepsilon \in la(\pi(i))$
 - [as before: $(aw, a\alpha, z) \vdash (w, \alpha, z)$]
- $\implies$ **deterministic top-down parsing automaton $DTA(G)$**

Remarks:

- $DTA(G)$ is actually not a pushdown automaton (a is read but not consumed). But: can be simulated using the finite control.
- Advantage of using lookahead is twofold:
 - Removal of nondeterminism
 - Earlier detection of syntax errors
(in configurations $(aw, A\alpha, z)$ where $a \notin \bigcup_{A \rightarrow \beta \in P} la(A \rightarrow \beta)$)

- 1 Repetition: $LL(1)$ Grammars
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Error Handling in LL Parsing

Transformation to $LL(1)$

Assume that $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma} \setminus LL(1)$

(i.e., there exist $A \rightarrow \beta \mid \gamma \in P$ such that $la(A \rightarrow \beta) \cap la(A \rightarrow \gamma) \neq \emptyset$)

Transformation to $LL(1)$

Assume that $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma} \setminus LL(1)$

(i.e., there exist $A \rightarrow \beta \mid \gamma \in P$ such that $la(A \rightarrow \beta) \cap la(A \rightarrow \gamma) \neq \emptyset$)

Two **heuristics** for transforming G into $G' \in LL(1)$:

- 1 Removal of left recursion
- 2 Left factorization

(used in parser-generating systems such as ANTLR)

Transformation to $LL(1)$

Assume that $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma} \setminus LL(1)$

(i.e., there exist $A \rightarrow \beta \mid \gamma \in P$ such that $la(A \rightarrow \beta) \cap la(A \rightarrow \gamma) \neq \emptyset$)

Two **heuristics** for transforming G into $G' \in LL(1)$:

- 1 Removal of left recursion
- 2 Left factorization

(used in parser-generating systems such as ANTLR)

Remarks:

- Transformations generally **preserve the semantics** (= generated language) of CFGs but **not the syntactic structure** of words (different syntax trees).
- Transformations **cannot always yield an $LL(1)$ grammar** (since not every context-free language is generated by an LL grammar; details later).

Definition 8.1 (Left recursion)

A grammar $G = \langle N, \Sigma, P, S \rangle \in CFG_\Sigma$ is called **left recursive** if there exist $A \in N$ and $\alpha \in X^*$ such that $A \Rightarrow^+ A\alpha$.

Definition 8.1 (Left recursion)

A grammar $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma}$ is called **left recursive** if there exist $A \in N$ and $\alpha \in X^*$ such that $A \Rightarrow^+ A\alpha$.

Corollary 8.2

If $G \in CFG_{\Sigma}$ is left recursive with $A \Rightarrow^+ A\alpha$, then there exists $\beta \in X^$ such that $A \Rightarrow_l^+ A\beta$.*

Definition 8.1 (Left recursion)

A grammar $G = \langle N, \Sigma, P, S \rangle \in CFG_\Sigma$ is called **left recursive** if there exist $A \in N$ and $\alpha \in X^*$ such that $A \Rightarrow^+ A\alpha$.

Corollary 8.2

If $G \in CFG_\Sigma$ is left recursive with $A \Rightarrow^+ A\alpha$, then there exists $\beta \in X^$ such that $A \Rightarrow_l^+ A\beta$.*

Example 8.3

The grammar (cf. Example 5.11)

$$\begin{aligned} G_{AE} : \quad E &\rightarrow E+T \mid T \\ T &\rightarrow T*F \mid F \\ F &\rightarrow (E) \mid a \mid b \end{aligned}$$

is left recursive, and in Example 7.8 it was shown that $G_{AE} \notin LL(1)$

Lemma 8.4

If $G \in CFG_\Sigma$ is left recursive, then $G \notin \bigcup_{k \in \mathbb{N}} LL(k)$.

Lemma 8.4

If $G \in CFG_\Sigma$ is left recursive, then $G \notin \bigcup_{k \in \mathbb{N}} LL(k)$.

Proof.

(for $k = 1$) Assume that $G \in LL(1)$ is left recursive with $A \Rightarrow_l^+ A\beta$. Together with the reducedness of G this implies that $S \Rightarrow_l^* vA\alpha \Rightarrow_l^+ vA\beta\alpha \Rightarrow_l^+ vw$ for some $v, w \in \Sigma^*$ and $\alpha \in X^*$.

Lemma 8.4

If $G \in CFG_\Sigma$ is left recursive, then $G \notin \bigcup_{k \in \mathbb{N}} LL(k)$.

Proof.

(for $k = 1$) Assume that $G \in LL(1)$ is left recursive with $A \Rightarrow_l^+ A\beta$. Together with the reducedness of G this implies that $S \Rightarrow_l^* vA\alpha \Rightarrow_l^+ vA\beta\alpha \Rightarrow_l^+ vw$ for some $v, w \in \Sigma^*$ and $\alpha \in X^*$. The corresponding computation of $DTA(G)$ (Def. 7.9) starts with $(vw, S, \varepsilon) \vdash^* (w, A\alpha, \dots) \vdash^+ (w, A\beta\alpha, \dots)$.

Lemma 8.4

If $G \in CFG_\Sigma$ is left recursive, then $G \notin \bigcup_{k \in \mathbb{N}} LL(k)$.

Proof.

(for $k = 1$) Assume that $G \in LL(1)$ is left recursive with $A \Rightarrow_l^+ A\beta$.

Together with the reducedness of G this implies that

$S \Rightarrow_l^* vA\alpha \Rightarrow_l^+ vA\beta\alpha \Rightarrow_l^+ vw$ for some $v, w \in \Sigma^*$ and $\alpha \in X^*$.

The corresponding computation of $DTA(G)$ (Def. 7.9) starts with $(vw, S, \varepsilon) \vdash^* (w, A\alpha, \dots) \vdash^+ (w, A\beta\alpha, \dots)$.

But in the last state the behaviour of $DTA(G)$ is determined by the same input ($\text{fi}(w)$) and stack symbol (A). Thus it enters a loop of the form $(w, A\alpha, \dots) \vdash^+ (w, A\beta\alpha, \dots) \vdash^+ (w, A\beta\beta\alpha, \dots) \vdash^+ \dots$ and will never recognize w . Contradiction □

Removing Direct Left Recursion

Direct left recursion occurs in productions of the form

$$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \dots \mid \beta_n \quad \text{where } \alpha_i \neq \varepsilon \text{ and } \beta_j \neq A\dots$$

Removing Direct Left Recursion

Direct left recursion occurs in productions of the form

$$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \dots \mid \beta_n \quad \text{where } \alpha_i \neq \varepsilon \text{ and } \beta_j \neq A\dots$$

Transformation: replacement by **right recursion**

$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \dots \mid \beta_n A' \\ A' &\rightarrow \alpha_1 A' \mid \dots \mid \alpha_m A' \mid \varepsilon \end{aligned}$$

(with a new $A' \in N$) which preserves $L(G)$.

Removing Direct Left Recursion

Direct left recursion occurs in productions of the form

$$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \dots \mid \beta_n \quad \text{where } \alpha_i \neq \varepsilon \text{ and } \beta_j \neq A\dots$$

Transformation: replacement by **right recursion**

$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \dots \mid \beta_n A' \\ A' &\rightarrow \alpha_1 A' \mid \dots \mid \alpha_m A' \mid \varepsilon \end{aligned}$$

(with a new $A' \in N$) which preserves $L(G)$.

Example 8.5

$$\begin{aligned} G_{AE} : \quad & E \rightarrow E+T \mid T \\ & T \rightarrow T*F \mid F \\ & F \rightarrow (E) \mid a \mid b \end{aligned} \quad \text{is transformed into}$$

$$\begin{aligned} G'_{AE} : \quad & E \rightarrow TE' \\ & E' \rightarrow +TE' \mid \varepsilon \\ & T \rightarrow FT' \\ & T' \rightarrow *FT' \mid \varepsilon \\ & F \rightarrow (E) \mid a \mid b \end{aligned} \quad \text{with } G'_{AE} \in LL(1) \text{ (see Example 7.8).}$$

Removing Indirect Left Recursion

Indirect left recursion occurs in productions of the form ($n \geq 1$)

$$\begin{array}{lcl} A & \rightarrow & A_1\alpha_1 \mid \dots \\ A_1 & \rightarrow & A_2\alpha_2 \mid \dots \\ & \vdots & \\ A_{n-1} & \rightarrow & A_n\alpha_n \mid \dots \\ A_n & \rightarrow & A\beta \mid \dots \end{array}$$

Removing Indirect Left Recursion

Indirect left recursion occurs in productions of the form ($n \geq 1$)

$$\begin{aligned} A &\rightarrow A_1\alpha_1 \mid \dots \\ A_1 &\rightarrow A_2\alpha_2 \mid \dots \\ &\vdots \\ A_{n-1} &\rightarrow A_n\alpha_n \mid \dots \\ A_n &\rightarrow A\beta \mid \dots \end{aligned}$$

Transformation: into **Greibach Normal Form** with productions of the form

$$\begin{aligned} A &\rightarrow aB_1 \dots B_n \quad (\text{where } B_i \neq S) \text{ or} \\ S &\rightarrow \varepsilon \end{aligned}$$

(cf. *Automata Theory and Formal Languages*)

Left Factorization

Applies to productions of the form

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

which are problematic if α “longer than” lookahead.

Left Factorization

Applies to productions of the form

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

which are problematic if α “longer than” lookahead.

Transformation: delaying the decision by **left factorization**

$$\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta \mid \gamma \end{aligned}$$

(with a new $A' \in N$) which preserves $L(G)$.

Left Factorization

Applies to productions of the form

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

which are problematic if α “longer than” lookahead.

Transformation: delaying the decision by **left factorization**

$$\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta \mid \gamma \end{aligned}$$

(with a new $A' \in N$) which preserves $L(G)$.

Example 8.6

Statement $\rightarrow$ if *Condition* then *Statement* else *Statement* fi
 | if *Condition* then *Statement* fi

is transformed into

Statement $\rightarrow$ if *Condition* then *Statement* S'
 $S' \rightarrow$ else *Statement* fi | fi

- 1 Repetition: $LL(1)$ Grammars
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Error Handling in LL Parsing

The Complexity of $LL(1)$ Parsing I

- $LL(1)$ parsing has time (and hence space) **complexity** $\mathcal{O}(|w|)$ (where $w \in \Sigma^*$ is the input word)

The Complexity of $LL(1)$ Parsing I

- $LL(1)$ parsing has time (and hence space) **complexity** $\mathcal{O}(|w|)$ (where $w \in \Sigma^*$ is the input word)
- Here: proof for **ε -free grammars** (i.e., $A \rightarrow \alpha \in P \implies \alpha \neq \varepsilon$)

The Complexity of $LL(1)$ Parsing I

- $LL(1)$ parsing has time (and hence space) **complexity** $\mathcal{O}(|w|)$ (where $w \in \Sigma^*$ is the input word)
- Here: proof for **ε -free grammars** (i.e., $A \rightarrow \alpha \in P \implies \alpha \neq \varepsilon$)
- General case: see O. Mayer: *Syntaxanalyse*, p. 211ff

The Complexity of $LL(1)$ Parsing I

- $LL(1)$ parsing has time (and hence space) **complexity** $\mathcal{O}(|w|)$ (where $w \in \Sigma^*$ is the input word)
- Here: proof for **ε -free grammars** (i.e., $A \rightarrow \alpha \in P \implies \alpha \neq \varepsilon$)
- General case: see O. Mayer: *Syntaxanalyse*, p. 211ff

Lemma 8.7

Let $G = \langle N, \Sigma, P, S \rangle \in LL(1)$ be ε -free. If

$$(w, S, \varepsilon) \vdash^n (\varepsilon, \varepsilon, z)$$

in $DTA(G)$, then

$$n \leq |w| \cdot |N| + 1.$$

The Complexity of $LL(1)$ Parsing II

Proof.

Let $(w, S, \varepsilon) \vdash^n (\varepsilon, \varepsilon, z)$ in $DTA(G)$. To show: $n \leq |w| \cdot |N| + 1$

- ① Clear: the computation involves $|w|$ matching steps and one accept-step.



The Complexity of $LL(1)$ Parsing II

Proof.

Let $(w, S, \varepsilon) \vdash^n (\varepsilon, \varepsilon, z)$ in $DTA(G)$. To show: $n \leq |w| \cdot |N| + 1$

- 1 Clear: the computation involves $|w|$ matching steps and one accept-step.
- 2 Since G is ε -free, every matching step is preceded by k expansion steps of the form

$$\begin{aligned}(av, A_1\alpha_1, \dots) &\vdash (av, A_2\alpha_2\alpha_1, \dots) \\ &\vdots \\ &\vdash (av, A_k\alpha_k \dots \alpha_1, \dots) \\ &\vdash (av, a\alpha_{k+1} \dots \alpha_1, \dots)\end{aligned}$$

where $A_i \rightarrow A_{i+1}\alpha_{i+1}$ for each $i \in [k-1]$ and $A_k \rightarrow a\alpha_{k+1}$.



The Complexity of $LL(1)$ Parsing II

Proof.

Let $(w, S, \varepsilon) \vdash^n (\varepsilon, \varepsilon, z)$ in $DTA(G)$. To show: $n \leq |w| \cdot |N| + 1$

- ① Clear: the computation involves $|w|$ matching steps and one accept-step.
- ② Since G is ε -free, every matching step is preceded by k expansion steps of the form

$$\begin{aligned}(av, A_1\alpha_1, \dots) &\vdash (av, A_2\alpha_2\alpha_1, \dots) \\ &\vdots \\ &\vdash (av, A_k\alpha_k \dots \alpha_1, \dots) \\ &\vdash (av, a\alpha_{k+1} \dots \alpha_1, \dots)\end{aligned}$$

where $A_i \rightarrow A_{i+1}\alpha_{i+1}$ for each $i \in [k-1]$ and $A_k \rightarrow a\alpha_{k+1}$.

- ③ This implies that $A_i \neq A_j$ for $i \neq j$ (by Lemma 8.4, G is not left recursive), and hence $k \leq |N|$.



The Complexity of $LL(1)$ Parsing II

Proof.

Let $(w, S, \varepsilon) \vdash^n (\varepsilon, \varepsilon, z)$ in $DTA(G)$. To show: $n \leq |w| \cdot |N| + 1$

- ① Clear: the computation involves $|w|$ matching steps and one accept-step.
- ② Since G is ε -free, every matching step is preceded by k expansion steps of the form

$$\begin{aligned}(av, A_1\alpha_1, \dots) &\vdash (av, A_2\alpha_2\alpha_1, \dots) \\ &\vdots \\ &\vdash (av, A_k\alpha_k \dots \alpha_1, \dots) \\ &\vdash (av, a\alpha_{k+1} \dots \alpha_1, \dots)\end{aligned}$$

where $A_i \rightarrow A_{i+1}\alpha_{i+1}$ for each $i \in [k-1]$ and $A_k \rightarrow a\alpha_{k+1}$.

- ③ This implies that $A_i \neq A_j$ for $i \neq j$ (by Lemma 8.4, G is not left recursive), and hence $k \leq |N|$.
- ④ Altogether: $n \leq |w| \cdot |N| + 1$.



- 1 Repetition: $LL(1)$ Grammars
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Error Handling in LL Parsing

Recursive-Descent Parsing I

Idea: avoid explicit use of pushdown store (as in $DTA(G)$) by employing **recursive procedures** (with implicit runtime stack)

Recursive-Descent Parsing I

Idea: avoid explicit use of pushdown store (as in $DTA(G)$) by employing **recursive procedures** (with implicit runtime stack)

Advantage: simple implementation

Recursive-Descent Parsing I

Idea: avoid explicit use of pushdown store (as in $\text{DTA}(G)$) by employing **recursive procedures** (with implicit runtime stack)

Advantage: simple implementation

Ingredients:

- variable **token** for current token
- function **next()** for invoking the scanner
- procedure **print(i)** for displaying the leftmost analysis (or errors)

Recursive-Descent Parsing I

Idea: avoid explicit use of pushdown store (as in $DTA(G)$) by employing **recursive procedures** (with implicit runtime stack)

Advantage: simple implementation

Ingredients:

- variable **token** for current token
- function **next()** for invoking the scanner
- procedure **print(i)** for displaying the leftmost analysis (or errors)

Method: to every $A \in N$ we assign a procedure **A()** which

- tests **token** with regard to the lookahead sets of the A -productions,
- prints the corresponding rule number and
- evaluates the corresponding right-hand side as follows:
 - for $a \in \Sigma$: check **token**; call **next()**
 - for $A \in N$: call **A()**

Example 8.8 (Arithmetic expressions; cf. Example 8.5)

```
proc main();
  token := next(); E()
proc E();   (*  $E \rightarrow T E'$  *)
  if token in {'(', 'a', 'b'} then print(1); T(); E'()
  else print(error); stop fi
proc E'();  (*  $E' \rightarrow + T E' \mid \varepsilon$  *)
  if token = '+' then print(2); token := next(); T(); E'()
  elsif token in {EOF, ')'} then print(3)
  else print(error); stop fi
proc T();   (*  $T \rightarrow F T'$  *)
  if token in {'(', 'a', 'b'} then print(4); F(); T'()
  else print(error); stop fi
proc T'();  (*  $T' \rightarrow * F T' \mid \varepsilon$  *)
  if token = '*' then print(5); token := next(); F(); T'()
  elsif token in {'+', EOF, ')'} then print(6)
  else print(error); stop fi
proc F();   (*  $F \rightarrow ( E ) \mid a \mid b$  *)
  if token = '(' then print(7); token := next(); E();
    if token = ')' then token := next() else print(error); stop fi
  elsif token = 'a' then print(8); token := next()
  elsif token = 'b' then print(9); token := next()
  else print(error); stop fi
```

- 1 Repetition: $LL(1)$ Grammars
- 2 Transformation to $LL(1)$
- 3 The Complexity of $LL(1)$ Parsing
- 4 Recursive-Descent Parsing
- 5 Error Handling in LL Parsing

Error configurations of $\text{DTA}(G)$:

- $(aw, A\alpha, z)$ where $a \notin \bigcup_{A \rightarrow \beta \in P} \text{la}(A \rightarrow \beta)$ ($\implies \text{act}(a, A) = \text{error}$)
- $(\varepsilon, A\alpha, z)$ where $\varepsilon \notin \bigcup_{A \rightarrow \beta \in P} \text{la}(A \rightarrow \beta)$ ($\implies \text{act}(\varepsilon, A) = \text{error}$)
- $(aw, b\alpha, z)$ where $a \neq b$ ($\implies \text{act}(a, b) = \text{error}$)
- $(\varepsilon, b\alpha, z)$ ($\implies \text{act}(\varepsilon, b) = \text{error}$)
- (aw, ε, z) ($\implies \text{act}(a, \varepsilon) = \text{error}$)

Error configurations of $\text{DTA}(G)$:

- $(aw, A\alpha, z)$ where $a \notin \bigcup_{A \rightarrow \beta \in P} \text{la}(A \rightarrow \beta)$ ($\implies \text{act}(a, A) = \text{error}$)
- $(\varepsilon, A\alpha, z)$ where $\varepsilon \notin \bigcup_{A \rightarrow \beta \in P} \text{la}(A \rightarrow \beta)$ ($\implies \text{act}(\varepsilon, A) = \text{error}$)
- $(aw, b\alpha, z)$ where $a \neq b$ ($\implies \text{act}(a, b) = \text{error}$)
- $(\varepsilon, b\alpha, z)$ ($\implies \text{act}(\varepsilon, b) = \text{error}$)
- (aw, ε, z) ($\implies \text{act}(a, \varepsilon) = \text{error}$)

Observation: correct prefix property of LL parsing, i.e., syntactic errors are detected at the earliest possible position (every input prefix which does not produce an error can be extended to a word $w \in L(G)$)

Does **not** mean: error is recognized at the position where it is caused!

Error configurations of $\text{DTA}(G)$:

- $(aw, A\alpha, z)$ where $a \notin \bigcup_{A \rightarrow \beta \in P} \text{la}(A \rightarrow \beta)$ ($\implies \text{act}(a, A) = \text{error}$)
- $(\varepsilon, A\alpha, z)$ where $\varepsilon \notin \bigcup_{A \rightarrow \beta \in P} \text{la}(A \rightarrow \beta)$ ($\implies \text{act}(\varepsilon, A) = \text{error}$)
- $(aw, b\alpha, z)$ where $a \neq b$ ($\implies \text{act}(a, b) = \text{error}$)
- $(\varepsilon, b\alpha, z)$ ($\implies \text{act}(\varepsilon, b) = \text{error}$)
- (aw, ε, z) ($\implies \text{act}(a, \varepsilon) = \text{error}$)

Observation: correct prefix property of LL parsing, i.e., syntactic errors are detected at the earliest possible position (every input prefix which does not produce an error can be extended to a word $w \in L(G)$)

Does **not** mean: error is recognized at the position where it is caused!

Example: assignment `a := b * c - (d + e)`;

Possible corrections:

- remove closing bracket: `a := b * c - (d + e);`
- insert opening bracket: `a := b * (c - (d + e));`

The General Problem

- Let $w = xy \in \Sigma^*$ be the input word such that x is the longest prefix of a word in $L(G)$ (i.e., the error is **detected** at the first symbol of y) and $w \notin L(G)$.

The General Problem

- Let $w = xy \in \Sigma^*$ be the input word such that x is the longest prefix of a word in $L(G)$ (i.e., the error is **detected** at the first symbol of y) and $w \notin L(G)$.
- Parser makes assumption about error type and **corrects** w accordingly:
 - Assumes prefix x' of x to be correct
 - Correct prefix property
 $\implies$ there exists $z \in \Sigma^*$ such that $x'z \in L(G)$
 - Parser chooses prefix z' of z and suffix y' of y
 - Parsing resumed with input $w' := x'z'y'$ (at first symbol of z')
(**error recovery**)

The General Problem

- Let $w = xy \in \Sigma^*$ be the input word such that x is the longest prefix of a word in $L(G)$ (i.e., the error is **detected** at the first symbol of y) and $w \notin L(G)$.
- Parser makes assumption about error type and **corrects** w accordingly:
 - Assumes prefix x' of x to be correct
 - Correct prefix property
 $\implies$ there exists $z \in \Sigma^*$ such that $x'z \in L(G)$
 - Parser chooses prefix z' of z and suffix y' of y
 - Parsing resumed with input $w' := x'z'y'$ (at first symbol of z')
(**error recovery**)
- Desirable properties of correction:
 - At least one symbol of y' can be processed before next error occurs (if $y' \neq \varepsilon$)
 - Preserve as many symbols of w as possible (i.e., x' and y' “long” and z' “short”)

The General Problem

- Let $w = xy \in \Sigma^*$ be the input word such that x is the longest prefix of a word in $L(G)$ (i.e., the error is **detected** at the first symbol of y) and $w \notin L(G)$.
- Parser makes assumption about error type and **corrects** w accordingly:
 - Assumes prefix x' of x to be correct
 - Correct prefix property
 $\implies$ there exists $z \in \Sigma^*$ such that $x'z \in L(G)$
 - Parser chooses prefix z' of z and suffix y' of y
 - Parsing resumed with input $w' := x'z'y'$ (at first symbol of z')
(**error recovery**)
- Desirable properties of correction:
 - At least one symbol of y' can be processed before next error occurs (if $y' \neq \varepsilon$)
 - Preserve as many symbols of w as possible (i.e., x' and y' “long” and z' “short”)
- $x' \neq x$ hard to implement, therefore usually $x' := x$

Further criteria for “good” error handling:

- Continuation of parsing in any case, independent of severity of error
- Frequency of correct error diagnosis
- Suppression of subsequent errors
- Complexity of analyzing correct inputs not impaired

Further criteria for “good” error handling:

- Continuation of parsing in any case, independent of severity of error
- Frequency of correct error diagnosis
- Suppression of subsequent errors
- Complexity of analyzing correct inputs not impaired

Observation: no “best method” available

- correction not unique
- experience of programmer
- peculiarities of (programming) language

⇒ employ heuristics

Simplest form of error handling: **panic mode**

Upon occurrence of an error,

- skip input symbols
- until a token in a selected set of “separating” or “closing” tokens appears (**synchronizing tokens**)

Simplest form of error handling: **panic mode**

Upon occurrence of an error,

- skip input symbols
- until a token in a selected set of “separating” or “closing” tokens appears (**synchronizing tokens**)

Example: suitable synchronizing tokens in imperative languages for

- assignments: “;”
- declarations: “;” or “,”
- control structures: fi or od
- blocks: end

Panic Mode

Simplest form of error handling: **panic mode**

Upon occurrence of an error,

- skip input symbols
- until a token in a selected set of “separating” or “closing” tokens appears (**synchronizing tokens**)

Example: suitable synchronizing tokens in imperative languages for

- assignments: “;”
- declarations: “;” or “,”
- control structures: fi or od
- blocks: end

Challenge: choose set of synchronizing tokens such that

- parser recovers quickly from errors that are likely to occur and
- not too much input is overread

(see Aho/Lam/Sethi/Ullman: *Compilers: Principles, Techniques, and Tools*, 2nd ed., p. 228ff)

Error Handling Example I

Example 8.9 (cf. Example 7.10)

$$\begin{aligned} G'_{AE} : \quad & E \rightarrow TE' && (1) \\ & E' \rightarrow +TE' \mid \varepsilon && (2, 3) \\ & T \rightarrow FT' && (4) \\ & T' \rightarrow *FT' \mid \varepsilon && (5, 6) \\ & F \rightarrow (E) \mid \mathbf{a} \mid \mathbf{b} && (7, 8, 9) \end{aligned}$$

$A \in N$	$\text{fo}(A)$
E	$\{\varepsilon,)\}$
E'	$\{\varepsilon,)\}$
T	$\{+, \varepsilon,)\}$
T'	$\{+, \varepsilon,)\}$
F	$\{*, +, \varepsilon,)\}$

Error Handling Example I

Example 8.9 (cf. Example 7.10)

$$\begin{aligned}
 G'_{AE} : \quad & E \rightarrow TE' & (1) \\
 & E' \rightarrow +TE' \mid \varepsilon & (2, 3) \\
 & T \rightarrow FT' & (4) \\
 & T' \rightarrow *FT' \mid \varepsilon & (5, 6) \\
 & F \rightarrow (E) \mid a \mid b & (7, 8, 9)
 \end{aligned}$$

$A \in N$	$\text{fo}(A)$
E	$\{\varepsilon,)\}$
E'	$\{\varepsilon,)\}$
T	$\{+, \varepsilon,)\}$
T'	$\{+, \varepsilon,)\}$
F	$\{*, +, \varepsilon,)\}$

With **synchronizing tokens** from fo sets:

$\text{act} : \Sigma_\varepsilon \times X_\varepsilon \rightarrow \{(\alpha, i) \mid \pi(i) = A \rightarrow \alpha\} \cup \{\text{pop, accept, error, **sync**}\}$ (empty = error)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- $\text{act}(x, Y) = \text{sync}$
 $\implies$ pop Y and resume parsing
- $\text{act}(x, A) = \text{error}$
 $\implies$ skip x and resume parsing

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- $\text{act}(x, Y) = \text{sync}$
 $\implies$ pop Y and resume parsing
- $\text{act}(x, A) = \text{error}$
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$
 $\vdash (a*b, E, \varepsilon)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- $\text{act}(x, Y) = \text{sync}$
 $\implies$ pop Y and resume parsing
- $\text{act}(x, A) = \text{error}$
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$
 $\vdash (a*b, E, \varepsilon)$
 $\vdash (a*b, TE', 1)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		(a, 8)	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		(b, 9)	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- $\text{act}(x, Y) = \text{sync}$
 $\implies$ pop Y and resume parsing
- $\text{act}(x, A) = \text{error}$
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$
 $\vdash (a*b, E, \varepsilon)$
 $\vdash (a*b, TE', 1)$
 $\vdash (a*b, FT'E', 14)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- $\text{act}(x, Y) = \text{sync}$
 $\implies$ pop Y and resume parsing
- $\text{act}(x, A) = \text{error}$
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$
 $\vdash (a*b, E, \varepsilon)$
 $\vdash (a*b, TE', 1)$
 $\vdash (a*b, FT'E', 14)$
 $\vdash (a*b, aT'E', 148)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

$\vdash (*+b, T'E', 148)$

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*+b, E, \varepsilon)$
 $\vdash (a*+b, E, \varepsilon)$
 $\vdash (a*+b, TE', 1)$
 $\vdash (a*+b, FT'E', 14)$
 $\vdash (a*+b, aT'E', 148)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*+b, E, \varepsilon)$
 $\vdash (a*+b, E, \varepsilon)$
 $\vdash (a*+b, TE', 1)$
 $\vdash (a*+b, FT'E', 14)$
 $\vdash (a*+b, aT'E', 148)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*+b, E, \varepsilon)$
 $\vdash (a*+b, E, \varepsilon)$
 $\vdash (a*+b, TE', 1)$
 $\vdash (a*+b, FT'E', 14)$
 $\vdash (a*+b, aT'E', 148)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$
 $\vdash (+b, T'E', 1485)$

$(+a*+b, E, \varepsilon)$
 $\vdash (a*+b, E, \varepsilon)$
 $\vdash (a*+b, TE', 1)$
 $\vdash (a*+b, FT'E', 14)$
 $\vdash (a*+b, aT'E', 148)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$
 $\vdash (+b, T'E', 1485)$
 $\vdash (+b, E', 14856)$

$(+a*+b, E, \varepsilon)$
 $\vdash (a*+b, E, \varepsilon)$
 $\vdash (a*+b, TE', 1)$
 $\vdash (a*+b, FT'E', 14)$
 $\vdash (a*+b, aT'E', 148)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$
 $\vdash (+b, T'E', 1485)$
 $\vdash (+b, E', 14856)$
 $\vdash (+b, +TE', 148562)$

$(+a*+b, E, \varepsilon)$
 $\vdash (a*+b, E, \varepsilon)$
 $\vdash (a*+b, TE', 1)$
 $\vdash (a*+b, FT'E', 14)$
 $\vdash (a*+b, aT'E', 148)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*+b, E, \varepsilon)$
 $\vdash (a*+b, E, \varepsilon)$
 $\vdash (a*+b, TE', 1)$
 $\vdash (a*+b, FT'E', 14)$
 $\vdash (a*+b, aT'E', 148)$

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$
 $\vdash (+b, T'E', 1485)$
 $\vdash (+b, E', 14856)$
 $\vdash (+b, +TE', 148562)$
 $\vdash (b, TE', 148562)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$
 $\vdash (a*b, E, \varepsilon)$
 $\vdash (a*b, TE', 1)$
 $\vdash (a*b, FT'E', 14)$
 $\vdash (a*b, aT'E', 148)$

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$
 $\vdash (+b, T'E', 1485)$
 $\vdash (+b, E', 14856)$
 $\vdash (+b, +TE', 148562)$
 $\vdash (b, TE', 148562)$
 $\vdash (b, FT'E', 1485624)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$
 $\vdash (a*b, E, \varepsilon)$
 $\vdash (a*b, TE', 1)$
 $\vdash (a*b, FT'E', 14)$
 $\vdash (a*b, aT'E', 148)$

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$
 $\vdash (+b, T'E', 1485)$
 $\vdash (+b, E', 14856)$
 $\vdash (+b, +TE', 148562)$
 $\vdash (b, TE', 148562)$
 $\vdash (b, FT'E', 1485624)$
 $\vdash (b, bT'E', 14856249)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$
 $\vdash (a*b, E, \varepsilon)$
 $\vdash (a*b, TE', 1)$
 $\vdash (a*b, FT'E', 14)$
 $\vdash (a*b, aT'E', 148)$

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$
 $\vdash (+b, T'E', 1485)$
 $\vdash (+b, E', 14856)$
 $\vdash (+b, +TE', 148562)$
 $\vdash (b, TE', 148562)$
 $\vdash (b, FT'E', 1485624)$
 $\vdash (b, bT'E', 14856249)$
 $\vdash (\varepsilon, T'E', 14856249)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$
 $\vdash (a*b, E, \varepsilon)$
 $\vdash (a*b, TE', 1)$
 $\vdash (a*b, FT'E', 14)$
 $\vdash (a*b, aT'E', 148)$

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$
 $\vdash (+b, T'E', 1485)$
 $\vdash (+b, E', 14856)$
 $\vdash (+b, +TE', 148562)$
 $\vdash (b, TE', 148562)$
 $\vdash (b, FT'E', 1485624)$
 $\vdash (b, bT'E', 14856249)$
 $\vdash (\varepsilon, T'E', 14856249)$
 $\vdash (\varepsilon, E', 148562496)$

Error Handling Example II

Example 8.9 (continued)

act	E	E'	T	T'	F	a	b	(	)	*	+	ε
a	$(TE', 1)$		$(FT', 4)$		$(a, 8)$	pop	sync	sync	sync	sync	sync	
b	$(TE', 1)$		$(FT', 4)$		$(b, 9)$	sync	pop	sync	sync	sync	sync	
(	$(TE', 1)$		$(FT', 4)$		$((E), 7)$	sync	sync	pop	sync	sync	sync	
)	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	pop	sync	sync	
*				$(*FT', 5)$	sync	sync	sync	sync	sync	pop	sync	
+		$(+TE', 2)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	pop	
ε	sync	$(\varepsilon, 3)$	sync	$(\varepsilon, 6)$	sync	sync	sync	sync	sync	sync	sync	accept

Meaning of table entries:

- act(x, Y) = sync
 $\implies$ pop Y and resume parsing
- act(x, A) = error
 $\implies$ skip x and resume parsing

$(+a*b, E, \varepsilon)$
 $\vdash (a*b, E, \varepsilon)$
 $\vdash (a*b, TE', 1)$
 $\vdash (a*b, FT'E', 14)$
 $\vdash (a*b, aT'E', 148)$

$\vdash (*+b, T'E', 148)$
 $\vdash (*+b, *FT'E', 1485)$
 $\vdash (+b, FT'E', 1485)$
 $\vdash (+b, T'E', 1485)$
 $\vdash (+b, E', 14856)$
 $\vdash (+b, +TE', 148562)$
 $\vdash (b, TE', 148562)$
 $\vdash (b, FT'E', 1485624)$
 $\vdash (b, bT'E', 14856249)$
 $\vdash (\varepsilon, T'E', 14856249)$
 $\vdash (\varepsilon, E', 148562496)$
 $\vdash (\varepsilon, \varepsilon, 1485624963)$