



Compiler Construction

Lecture 11: Syntax Analysis VII (Practical Issues)

Summer Semester 2016

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-16/cc/>

Recap: $LR(1)$ and $LALR(1)$ Parsing

Outline of Lecture 11

Recap: $LR(1)$ and $LALR(1)$ Parsing

Bottom-Up Parsing of Ambiguous Grammars

Generating Parsers Using `yacc` and `bison`

Expressiveness of LL and LR Grammars

LL and LR Parsing in Practice

Recap: $LR(1)$ and $LALR(1)$ Parsing

$LR(1)$ Items and Sets

Observation: not every element of $fo(A)$ can follow every occurrence of A
 \implies refinement of $LR(0)$ items by adding possible lookahead symbols

Definition ($LR(1)$ items and sets)

Let $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma}$ be start separated by $S' \rightarrow S$.

- If $S' \Rightarrow_r^* \alpha A a w \Rightarrow_r \alpha \beta_1 \beta_2 a w$, then $[A \rightarrow \beta_1 \cdot \beta_2, a]$ is called an $LR(1)$ item for $\alpha \beta_1$.
- If $S' \Rightarrow_r^* \alpha A \Rightarrow_r \alpha \beta_1 \beta_2$, then $[A \rightarrow \beta_1 \cdot \beta_2, \varepsilon]$ is called an $LR(1)$ item for $\alpha \beta_1$.
- Given $\gamma \in X^*$, $LR(1)(\gamma)$ denotes the set of all $LR(1)$ items for γ , called the $LR(1)$ set (or: $LR(1)$ information) of γ .
- $LR(1)(G) := \{LR(1)(\gamma) \mid \gamma \in X^*\}$.

Recap: $LR(1)$ and $LALR(1)$ Parsing

$LR(1)$ Conflicts

Definition ($LR(1)$ conflicts)

Let $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma}$ and $I \in LR(1)(G)$.

- I has a **shift/reduce conflict** if there exist $A \rightarrow \alpha_1 a \alpha_2$, $B \rightarrow \beta \in P$ and $x \in \Sigma_{\epsilon}$ such that

$$[A \rightarrow \alpha_1 \cdot a \alpha_2, x], [B \rightarrow \beta \cdot, a] \in I.$$

- I has a **reduce/reduce conflict** if there exist $x \in \Sigma_{\epsilon}$ and $A \rightarrow \alpha$, $B \rightarrow \beta \in P$ with $A \neq B$ or $\alpha \neq \beta$ such that

$$[A \rightarrow \alpha \cdot, x], [B \rightarrow \beta \cdot, x] \in I.$$

Lemma

$G \in LR(1)$ iff no $I \in LR(1)(G)$ contains conflicting items.

Recap: $LR(1)$ and $LALR(1)$ Parsing

The $LR(1)$ Action Function

Definition ($LR(1)$ action function)

The $LR(1)$ action function

$$\text{act} : LR(1)(G) \times \Sigma_\varepsilon \rightarrow \{\text{red } i \mid i \in [p]\} \cup \{\text{shift, accept, error}\}$$

is defined by

$$\text{act}(I, x) := \begin{cases} \text{red } i & \text{if } i \neq 0, \pi_i = A \rightarrow \alpha \text{ and } [A \rightarrow \alpha \cdot, x] \in I \\ \text{shift} & \text{if } [A \rightarrow \alpha_1 \cdot x \alpha_2, y] \in I \text{ and } x \in \Sigma \\ \text{accept} & \text{if } [S' \rightarrow S \cdot, \varepsilon] \in I \text{ and } x = \varepsilon \\ \text{error} & \text{otherwise} \end{cases}$$

Corollary

For every $G \in CFG_\Sigma$, $G \in LR(1)$ iff its $LR(1)$ action function is well defined.

Recap: $LR(1)$ and $LALR(1)$ Parsing

$LR(0)$ Equivalence

Observation: potential **redundancy by containment** of $LR(0)$ sets in $LR(1)$ sets (cf. Corollary 10.3)

Definition ($LR(0)$ equivalence)

Let $lr_0 : LR(1)(G) \rightarrow LR(0)(G)$ be defined by

$$lr_0(I) := \{[A \rightarrow \beta_1 \cdot \beta_2] \mid [A \rightarrow \beta_1 \cdot \beta_2, x] \in I\}.$$

Two sets $I_1, I_2 \in LR(1)(G)$ are called **$LR(0)$ -equivalent** (notation: $I_1 \sim_0 I_2$) if $lr_0(I_1) = lr_0(I_2)$.

Recap: $LR(1)$ and $LALR(1)$ Parsing

$LALR(1)$ Sets I

Corollary

For every $G \in CFG_{\Sigma}$, $|LR(1)(G) / \sim_0| = |LR(0)(G)|$.

Idea: merge $LR(0)$ -equivalent $LR(1)$ sets

(maintaining the lookahead information, but possibly introducing conflicts)

Definition ($LALR(1)$ sets)

Let $G \in CFG_{\Sigma}$.

- An information $I \in LR(1)(G)$ determines the $LALR(1)$ set

$$\bigcup [I]_{\sim_0} = \bigcup \{I' \in LR(1)(G) \mid I' \sim_0 I\}.$$

- The set of all $LALR(1)$ sets of G is denoted by $LALR(1)(G)$.

Remark: by Corollary 10.16, $|LALR(1)(G)| = |LR(0)(G)|$
(but $LALR(1)$ sets provide additional lookahead information)

Bottom-Up Parsing of Ambiguous Grammars

Outline of Lecture 11

Recap: $LR(1)$ and $LALR(1)$ Parsing

Bottom-Up Parsing of Ambiguous Grammars

Generating Parsers Using `yacc` and `bison`

Expressiveness of LL and LR Grammars

LL and LR Parsing in Practice

Bottom-Up Parsing of Ambiguous Grammars

Ambiguous Grammars

Reminder (Definition 5.5): a context-free grammar $G \in CFG_{\Sigma}$ is called **unambiguous** if every word $w \in L(G)$ has exactly one syntax tree. Otherwise it is called **ambiguous**.

Bottom-Up Parsing of Ambiguous Grammars

Ambiguous Grammars

Reminder (Definition 5.5): a context-free grammar $G \in CFG_{\Sigma}$ is called **unambiguous** if every word $w \in L(G)$ has exactly one syntax tree. Otherwise it is called **ambiguous**.

Lemma 11.1

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

Bottom-Up Parsing of Ambiguous Grammars

Ambiguous Grammars

Reminder (Definition 5.5): a context-free grammar $G \in CFG_{\Sigma}$ is called **unambiguous** if every word $w \in L(G)$ has exactly one syntax tree. Otherwise it is called **ambiguous**.

Lemma 11.1

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

Proof.

Assume that there exist $k \in \mathbb{N}$ and $G \in LR(k)$ such that G is ambiguous.

Bottom-Up Parsing of Ambiguous Grammars

Ambiguous Grammars

Reminder (Definition 5.5): a context-free grammar $G \in CFG_{\Sigma}$ is called **unambiguous** if every word $w \in L(G)$ has exactly one syntax tree. Otherwise it is called **ambiguous**.

Lemma 11.1

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

Proof.

Assume that there exist $k \in \mathbb{N}$ and $G \in LR(k)$ such that G is ambiguous.

Hence there exists $w \in L(G)$ with different right derivations. Let αAv be the last common sentence of the two derivations (i.e., $\beta \neq \beta'$):

$$S \Rightarrow_r^* \alpha Av \begin{cases} \Rightarrow_r \alpha \beta v \Rightarrow_r^* w \\ \Rightarrow_r \alpha \beta' v \Rightarrow_r^* w \end{cases}$$

Bottom-Up Parsing of Ambiguous Grammars

Ambiguous Grammars

Reminder (Definition 5.5): a context-free grammar $G \in CFG_\Sigma$ is called **unambiguous** if every word $w \in L(G)$ has exactly one syntax tree. Otherwise it is called **ambiguous**.

Lemma 11.1

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

Proof.

Assume that there exist $k \in \mathbb{N}$ and $G \in LR(k)$ such that G is ambiguous.

Hence there exists $w \in L(G)$ with different right derivations. Let αAv be the last common sentence of the two derivations (i.e., $\beta \neq \beta'$):

$$S \Rightarrow_r^* \alpha Av \begin{cases} \Rightarrow_r \alpha \beta v \Rightarrow_r^* w \\ \Rightarrow_r \alpha \beta' v \Rightarrow_r^* w \end{cases}$$

But since $\text{first}_k(v) = \text{first}_k(v)$ for every $v \in \Sigma^*$, Definition 8.8 yields that $\beta = \beta'$. Contradiction \square

Bottom-Up Parsing of Ambiguous Grammars

Ambiguous Grammars

Reminder (Definition 5.5): a context-free grammar $G \in CFG_\Sigma$ is called **unambiguous** if every word $w \in L(G)$ has exactly one syntax tree. Otherwise it is called **ambiguous**.

Lemma 11.1

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

Proof.

Assume that there exist $k \in \mathbb{N}$ and $G \in LR(k)$ such that G is ambiguous.

Hence there exists $w \in L(G)$ with different right derivations. Let αAv be the last common sentence of the two derivations (i.e., $\beta \neq \beta'$):

$$S \Rightarrow_r^* \alpha Av \begin{cases} \Rightarrow_r \alpha \beta v \Rightarrow_r^* w \\ \Rightarrow_r \alpha \beta' v \Rightarrow_r^* w \end{cases}$$

But since $\text{first}_k(v) = \text{first}_k(v)$ for every $v \in \Sigma^*$, Definition 8.8 yields that $\beta = \beta'$. Contradiction \square

However ambiguity is a **natural specification method** which generally avoids involved syntactic constructs.

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars I

Example 11.2 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad (0) \quad E \rightarrow E+E \mid E * E \mid a \quad (1, 2, 3)$

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars I

Example 11.2 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad (0) \quad E \rightarrow E+E \mid E * E \mid a \quad (1, 2, 3)$

Precedence: $*$ $>$ $+$ **Associativity:** left (thus: $a+a*a+a := (a+(a*a))+a$)

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars I

Example 11.2 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad (0) \quad E \rightarrow E+E \mid E*E \mid a \quad (1, 2, 3)$

Precedence: $* > +$ **Associativity:** left (thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G) : I_0 := LR(0)(\varepsilon) : [E' \rightarrow \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$
 $I_1 := LR(0)(E) : [E' \rightarrow E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$
 $I_2 := LR(0)(a) : [E \rightarrow a \cdot]$
 $I_3 := LR(0)(E+) : [E \rightarrow E+ \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$
 $I_4 := LR(0)(E*) : [E \rightarrow E* \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$
 $I_5 := LR(0)(E+E) : [E \rightarrow E+E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$
 $I_6 := LR(0)(E*E) : [E \rightarrow E*E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars I

Example 11.2 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad (0) \quad E \rightarrow E+E \mid E*E \mid a \quad (1, 2, 3)$

Precedence: $* > +$ **Associativity:** left (thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G) : I_0 := LR(0)(\varepsilon) : \quad [E' \rightarrow \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_1 := LR(0)(E) : \quad [E' \rightarrow E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

$I_2 := LR(0)(a) : \quad [E \rightarrow a \cdot]$

$I_3 := LR(0)(E+) : \quad [E \rightarrow E+ \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_4 := LR(0)(E*) : \quad [E \rightarrow E* \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_5 := LR(0)(E+E) : \quad [E \rightarrow E+E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

$I_6 := LR(0)(E*E) : \quad [E \rightarrow E*E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

Conflicts: I_1 : $SLR(1)$ -solvable (reduce on ε , shift on $+/*$)

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars I

Example 11.2 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad (0) \quad E \rightarrow E+E \mid E*E \mid a \quad (1, 2, 3)$

Precedence: $* > +$ **Associativity:** left (thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G) : I_0 := LR(0)(\varepsilon) : [E' \rightarrow \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_1 := LR(0)(E) : [E' \rightarrow E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

$I_2 := LR(0)(a) : [E \rightarrow a \cdot]$

$I_3 := LR(0)(E+) : [E \rightarrow E+ \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_4 := LR(0)(E*) : [E \rightarrow E* \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_5 := LR(0)(E+E) : [E \rightarrow E+E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

$I_6 := LR(0)(E*E) : [E \rightarrow E*E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

Conflicts: I_1 : $SLR(1)$ -solvable (reduce on ε , shift on $+/*$)

I_5, I_6 : not $SLR(1)$ -solvable ($+, * \in fo(E)$)

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars I

Example 11.2 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad (0) \quad E \rightarrow E+E \mid E*E \mid a \quad (1, 2, 3)$

Precedence: $* > +$ **Associativity:** left (thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G) : I_0 := LR(0)(\varepsilon) : [E' \rightarrow \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_1 := LR(0)(E) : [E' \rightarrow E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

$I_2 := LR(0)(a) : [E \rightarrow a \cdot]$

$I_3 := LR(0)(E+) : [E \rightarrow E+ \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_4 := LR(0)(E*) : [E \rightarrow E* \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_5 := LR(0)(E+E) : [E \rightarrow E+E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

$I_6 := LR(0)(E*E) : [E \rightarrow E*E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

Conflicts: I_1 : $SLR(1)$ -solvable (reduce on ε , shift on $+/*$)

I_5, I_6 : not $SLR(1)$ -solvable ($+, * \in fo(E)$)

Solution: $I_5 : * > + \implies act(I_5, *) := \text{shift}, + \text{ left assoc.} \implies act(I_5, +) := \text{red 1}$

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars I

Example 11.2 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad (0) \quad E \rightarrow E+E \mid E*E \mid a \quad (1, 2, 3)$

Precedence: $* > +$ **Associativity:** left (thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G) : I_0 := LR(0)(\varepsilon) : [E' \rightarrow \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_1 := LR(0)(E) : [E' \rightarrow E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

$I_2 := LR(0)(a) : [E \rightarrow a \cdot]$

$I_3 := LR(0)(E+) : [E \rightarrow E+ \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_4 := LR(0)(E*) : [E \rightarrow E* \cdot E] \quad [E \rightarrow \cdot E+E] \quad [E \rightarrow \cdot E*E] \quad [E \rightarrow \cdot a]$

$I_5 := LR(0)(E+E) : [E \rightarrow E+E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

$I_6 := LR(0)(E*E) : [E \rightarrow E*E \cdot] \quad [E \rightarrow E \cdot +E] \quad [E \rightarrow E \cdot *E]$

Conflicts: I_1 : $SLR(1)$ -solvable (reduce on ε , shift on $+/*$)

I_5, I_6 : not $SLR(1)$ -solvable ($+, * \in fo(E)$)

Solution: $I_5 : * > + \implies act(I_5, *) := \text{shift}, + \text{ left assoc.} \implies act(I_5, +) := \text{red 1}$

$I_6 : * > + \implies act(I_6, +) := \text{red 2}, * \text{ left assoc.} \implies act(I_6, *) := \text{red 2}$

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars II

Example 11.3 (“Dangling else”)

$G : S' \rightarrow S \quad S \rightarrow iSeS \mid iS \mid a$

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars II

Example 11.3 (“Dangling else”)

$G : S' \rightarrow S \quad S \rightarrow iSeS \mid iS \mid a$

Ambiguity: $iaea := (1) i(iaea)$ (common) or $(2) i(ia)ea$

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars II

Example 11.3 (“Dangling else”)

$G : S' \rightarrow S \quad S \rightarrow iSeS \mid iS \mid a$

Ambiguity: $iaea := (1) i(iaea)$ (common) or $(2) i(ia)ea$

$LR(0)(G) : I_0 := LR(0)(\varepsilon) : [S' \rightarrow \cdot S] [S \rightarrow \cdot iSeS] [S \rightarrow \cdot iS] [S \rightarrow \cdot a]$
 $I_1 := LR(0)(S) : [S' \rightarrow S \cdot]$
 $I_2 := LR(0)(i) : [S \rightarrow i \cdot SeS] [S \rightarrow i \cdot S] [S \rightarrow \cdot iSeS]$
 $[S \rightarrow \cdot iS] [S \rightarrow \cdot a]$
 $I_3 := LR(0)(a) : [S \rightarrow a \cdot]$
 $I_4 := LR(0)(iS) : [S \rightarrow iS \cdot eS] [S \rightarrow iS \cdot]$
 $I_5 := LR(0)(iSe) : [S \rightarrow iSe \cdot S] [S \rightarrow \cdot iSeS] [S \rightarrow \cdot iS] [S \rightarrow \cdot a]$
 $I_6 := LR(0)(iSeS) : [S \rightarrow iSeS \cdot]$

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars II

Example 11.3 (“Dangling else”)

$G : S' \rightarrow S \quad S \rightarrow iSeS \mid iS \mid a$

Ambiguity: $iaea := (1) i(iaea)$ (common) or $(2) i(ia)ea$

$LR(0)(G) : I_0 := LR(0)(\varepsilon) : [S' \rightarrow \cdot S] [S \rightarrow \cdot iSeS] [S \rightarrow \cdot iS] [S \rightarrow \cdot a]$
 $I_1 := LR(0)(S) : [S' \rightarrow S \cdot]$
 $I_2 := LR(0)(i) : [S \rightarrow i \cdot SeS] [S \rightarrow i \cdot S] [S \rightarrow \cdot iSeS]$
 $[S \rightarrow \cdot iS] [S \rightarrow \cdot a]$
 $I_3 := LR(0)(a) : [S \rightarrow a \cdot]$
 $I_4 := LR(0)(iS) : [S \rightarrow iS \cdot eS] [S \rightarrow iS \cdot]$
 $I_5 := LR(0)(iSe) : [S \rightarrow iSe \cdot S] [S \rightarrow \cdot iSeS] [S \rightarrow \cdot iS] [S \rightarrow \cdot a]$
 $I_6 := LR(0)(iSeS) : [S \rightarrow iSeS \cdot]$

Conflict in $I_4 : e \in \text{fo}(S) \implies$ not $SLR(1)$ -solvable

Bottom-Up Parsing of Ambiguous Grammars

Bottom-Up Parsing of Ambiguous Grammars II

Example 11.3 (“Dangling else”)

$G : S' \rightarrow S \quad S \rightarrow iSeS \mid iS \mid a$

Ambiguity: $iaea := (1) i(iaea)$ (common) or $(2) i(ia)ea$

$LR(0)(G) : l_0 := LR(0)(\varepsilon) : [S' \rightarrow \cdot S] [S \rightarrow \cdot iSeS] [S \rightarrow \cdot iS] [S \rightarrow \cdot a]$
 $l_1 := LR(0)(S) : [S' \rightarrow S \cdot]$
 $l_2 := LR(0)(i) : [S \rightarrow i \cdot SeS] [S \rightarrow i \cdot S] [S \rightarrow \cdot iSeS]$
 $[S \rightarrow \cdot iS] [S \rightarrow \cdot a]$
 $l_3 := LR(0)(a) : [S \rightarrow a \cdot]$
 $l_4 := LR(0)(iS) : [S \rightarrow iS \cdot eS] [S \rightarrow iS \cdot]$
 $l_5 := LR(0)(iSe) : [S \rightarrow iSe \cdot S] [S \rightarrow \cdot iSeS] [S \rightarrow \cdot iS] [S \rightarrow \cdot a]$
 $l_6 := LR(0)(iSeS) : [S \rightarrow iSeS \cdot]$

Conflict in $l_4 : e \in \text{fo}(S) \implies$ not $SLR(1)$ -solvable

Solution (1): $\text{act}(l_4, e) := \text{shift}$

Generating Parsers Using yacc and bison

Outline of Lecture 11

Recap: $LR(1)$ and $LALR(1)$ Parsing

Bottom-Up Parsing of Ambiguous Grammars

Generating Parsers Using yacc and bison

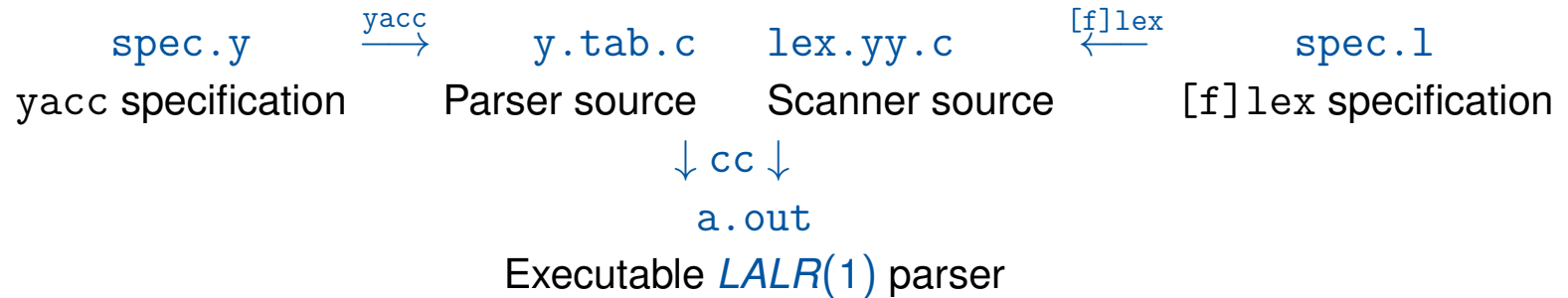
Expressiveness of LL and LR Grammars

LL and LR Parsing in Practice

Generating Parsers Using `yacc` and `bison`

The `yacc` and `bison` Tools

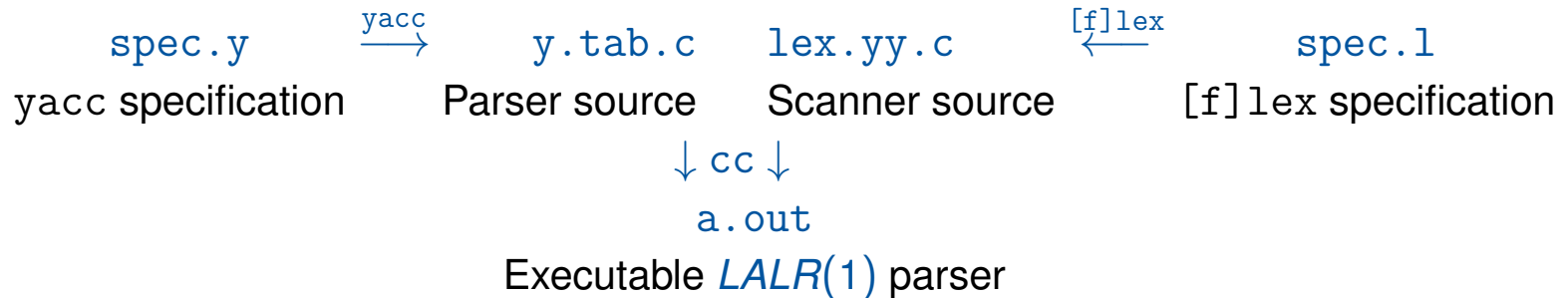
Usage of `yacc` (“yet another compiler compiler”):



Generating Parsers Using `yacc` and `bison`

The `yacc` and `bison` Tools

Usage of `yacc` (“yet another compiler compiler”):



Like for `[f]lex`, a `yacc specification` is of the form

Declarations (optional)

%%

Rules

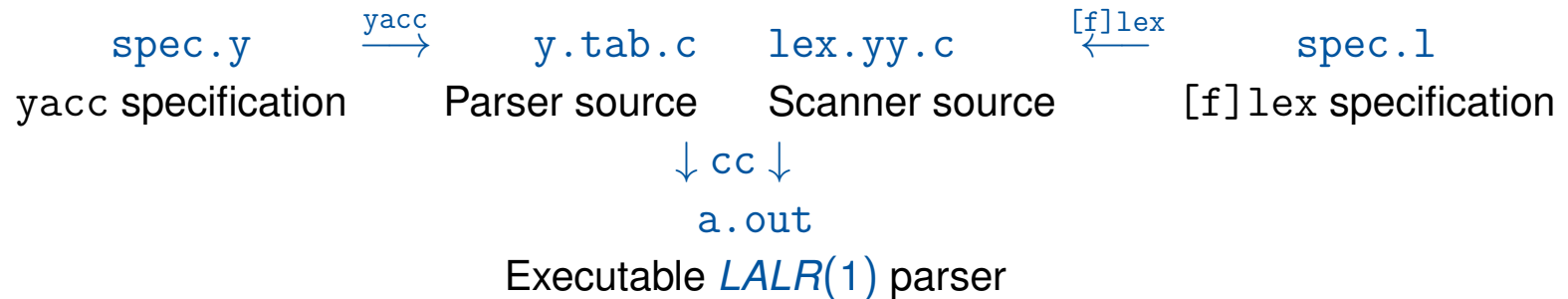
%%

Auxiliary procedures (optional)

Generating Parsers Using `yacc` and `bison`

The `yacc` and `bison` Tools

Usage of `yacc` (“yet another compiler compiler”):



Like for `[f]lex`, a `yacc specification` is of the form

Declarations (optional)

%%

Rules

%%

Auxiliary procedures (optional)

`bison` : upward-compatible GNU implementation of `yacc`
(more flexible w.r.t. file names, ...)

Generating Parsers Using yacc and bison

yacc Specifications

Declarations: • Token definitions: `%token Tokens`

- Not every token needs to be declared (`'+' , '=' , ...`)
- Start symbol: `%start Symbol` (optional)
- C code for declarations etc.: `%{ Code %}`

Generating Parsers Using yacc and bison

yacc Specifications

Declarations: • Token definitions: `%token Tokens`

- Not every token needs to be declared (`'+' , '=' , ...`)
- Start symbol: `%start Symbol` (optional)
- C code for declarations etc.: `%{ Code %}`

Rules: context-free productions and semantic actions

- $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ represented as

$$\begin{array}{l} A : \alpha_1 \{Action_1\} \\ \quad \mid \alpha_2 \{Action_2\} \\ \quad \vdots \\ \quad \mid \alpha_n \{Action_n\}; \end{array}$$

- Semantic actions = C statements for computing attribute values
- `$$` = attribute value of A
- `$i` = attribute value of i -th symbol on right-hand side
- Default action: `$$ = $1`

Generating Parsers Using yacc and bison

yacc Specifications

Declarations: • Token definitions: `%token Tokens`

- Not every token needs to be declared (`'+' , '=' , ...`)
- Start symbol: `%start Symbol` (optional)
- C code for declarations etc.: `%{ Code %}`

Rules: context-free productions and semantic actions

- $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$ represented as

$$\begin{array}{l} A : \alpha_1 \{Action_1\} \\ \quad \mid \alpha_2 \{Action_2\} \\ \quad \vdots \\ \quad \mid \alpha_n \{Action_n\}; \end{array}$$

- Semantic actions = C statements for computing attribute values
- `$$` = attribute value of A
- `$i` = attribute value of i -th symbol on right-hand side
- Default action: `$$ = $1`

Auxiliary procedures: scanner (if not `[f]lex`), error routines, ...

Generating Parsers Using yacc and bison

Example: Simple Desk Calculator I

```
%{/* SLR(1) grammar for arithmetic expressions (Example 9.11) */
#include <stdio.h>
#include <ctype.h>
%}
%token DIGIT
%%
line      : expr '\n'      { printf("%d\n", $1); };
expr     : expr '+' term  { $$ = $1 + $3; }
          | term          { $$ = $1; };
term     : term '*' factor { $$ = $1 * $3; }
          | factor       { $$ = $1; };
factor   : '(' expr ')'   { $$ = $2; }
          | DIGIT        { $$ = $1; };
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) yylval = c - '0'; return DIGIT;
    return c;
}
```

Generating Parsers Using yacc and bison

Example: Simple Desk Calculator II

```
> yacc calc.y
> cc y.tab.c -ly
> a.out
2+3
5
> a.out
2+3*5
17
```

Generating Parsers Using yacc and bison

An Ambiguous Grammar I

```
%{/* Ambiguous grammar for arithmetic expressions (Example 11.2) */
#include <stdio.h>
#include <ctype.h>
%}
%token DIGIT
%%
line      : expr '\n'          { printf("%d\n", $1); };
expr      : expr '+' expr     { $$ = $1 + $3; }
          | expr '*' expr     { $$ = $1 * $3; }
          | DIGIT             { $$ = $1; };
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {yylval = c - '0'; return DIGIT;}
    return c;
}
```

Generating Parsers Using yacc and bison

An Ambiguous Grammar II

Invoking yacc with the option `-v` produces a report `y.output`:

...
State 8

```
2 expr: expr . '+' expr
2     | expr '+' expr .
3     | expr . '*' expr

'+'  shift and goto state 6
'*'  shift and goto state 7

'+'      [reduce with rule 2 (expr)]
'*'      [reduce with rule 2 (expr)]
```

State 9

```
2 expr: expr . '+' expr
3     | expr . '*' expr
3     | expr '*' expr .

'+'  shift and goto state 6
'*'  shift and goto state 7

'+'      [reduce with rule 3 (expr)]
'*'      [reduce with rule 3 (expr)]
```

Generating Parsers Using yacc and bison

Conflict Handling in yacc

Default conflict resolving strategy in yacc:

reduce/reduce: choose **first conflicting production** in specification

Conflict Handling in `yacc`

Default conflict resolving strategy in `yacc`:

reduce/reduce: choose **first conflicting production** in specification

shift/reduce: prefer **shift**

- resolves dangling-else ambiguity (Example 11.3) correctly
- also adequate for strong following weak operator (`*` after `+`; Example 11.2) and for right-associative operators
- not appropriate for weak following strong operator and for left-associative binary operators (\implies reduce; see Example 11.2)

Conflict Handling in `yacc`

Default conflict resolving strategy in `yacc`:

reduce/reduce: choose **first conflicting production** in specification

shift/reduce: prefer **shift**

- resolves dangling-else ambiguity (Example 11.3) correctly
- also adequate for strong following weak operator (`*` after `+`; Example 11.2) and for right-associative operators
- not appropriate for weak following strong operator and for left-associative binary operators (\implies reduce; see Example 11.2)

For ambiguous grammar:

```
> yacc ambig.y
conflicts: 4 shift/reduce
> cc y.tab.c -ly
> a.out
2+3*5
17
> a.out
2*3+5
16
```


Precedences and Associativities in yacc I

General mechanism for resolving conflicts:

$$\begin{array}{l} \%[\text{left|right}] \textit{Operators}_1 \\ \vdots \\ \%[\text{left|right}] \textit{Operators}_n \end{array}$$

- operators in one line have given associativity and same precedence
- precedence increases over lines

Generating Parsers Using `yacc` and `bison`

Precedences and Associativities in `yacc` I

General mechanism for resolving conflicts:

```
%[left|right] Operators1
      ⋮
%[left|right] Operatorsn
```

- operators in one line have given associativity and same precedence
- precedence increases over lines

Example 11.4

```
%left '+' '-'
%left '*' '/'
%right '^'
```

`^` (right associative) binds stronger than `*` and `/` (left associative), which in turn bind stronger than `+` and `-` (left associative)

Generating Parsers Using yacc and bison

Precedences and Associativities in yacc II

```
%{/* Ambiguous grammar for arithmetic expressions
   with precedences and associativities */
#include <stdio.h>
#include <ctype.h>
%}
%token DIGIT
%left '+'
%left '*'
%%
line      : expr '\n'  { printf("%d\n", $1); };
expr      : expr '+' expr  { $$ = $1 + $3; }
          | expr '*' expr  { $$ = $1 * $3; }
          | DIGIT          { $$ = $1; };
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {yylval = c - '0'; return DIGIT;}
    return c;
}
```

Precedences and Associativities in yacc III

```
> yacc ambig-prio.y
```

```
> cc y.tab.c -ly
```

```
> a.out
```

```
2*3+5
```

```
11
```

```
> a.out
```

```
2+3*5
```

```
17
```

Expressiveness of LL and LR Grammars

Outline of Lecture 11

Recap: $LR(1)$ and $LALR(1)$ Parsing

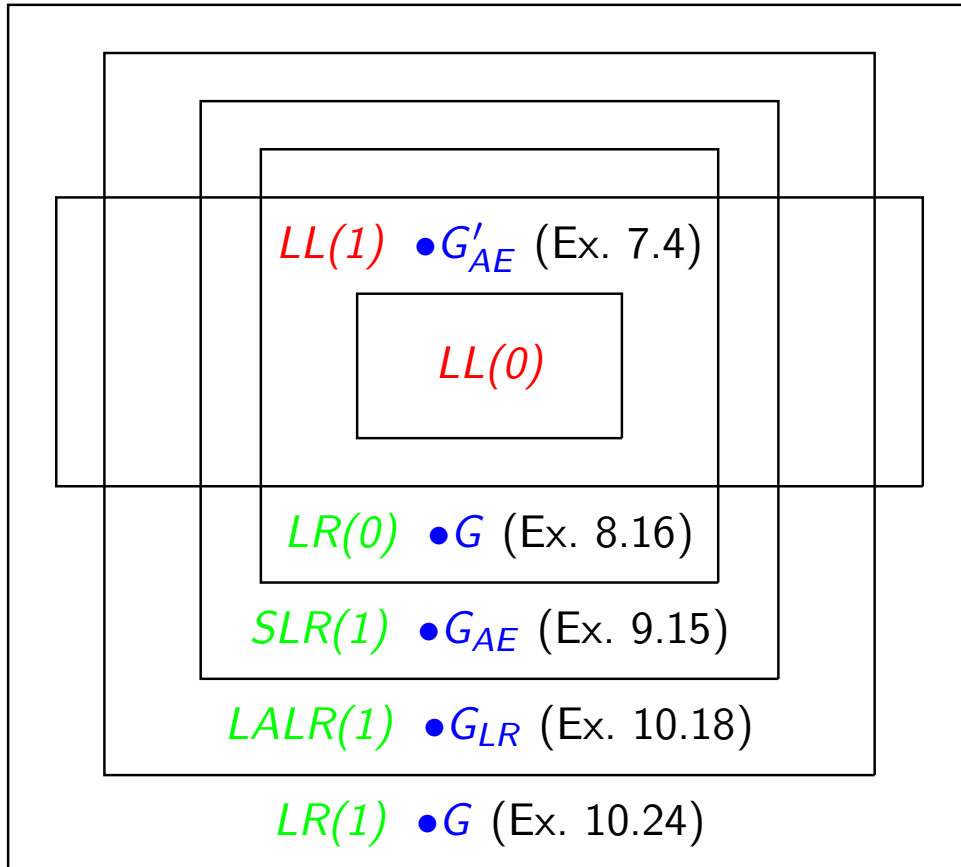
Bottom-Up Parsing of Ambiguous Grammars

Generating Parsers Using `yacc` and `bison`

Expressiveness of LL and LR Grammars

LL and LR Parsing in Practice

Overview of Grammar Classes



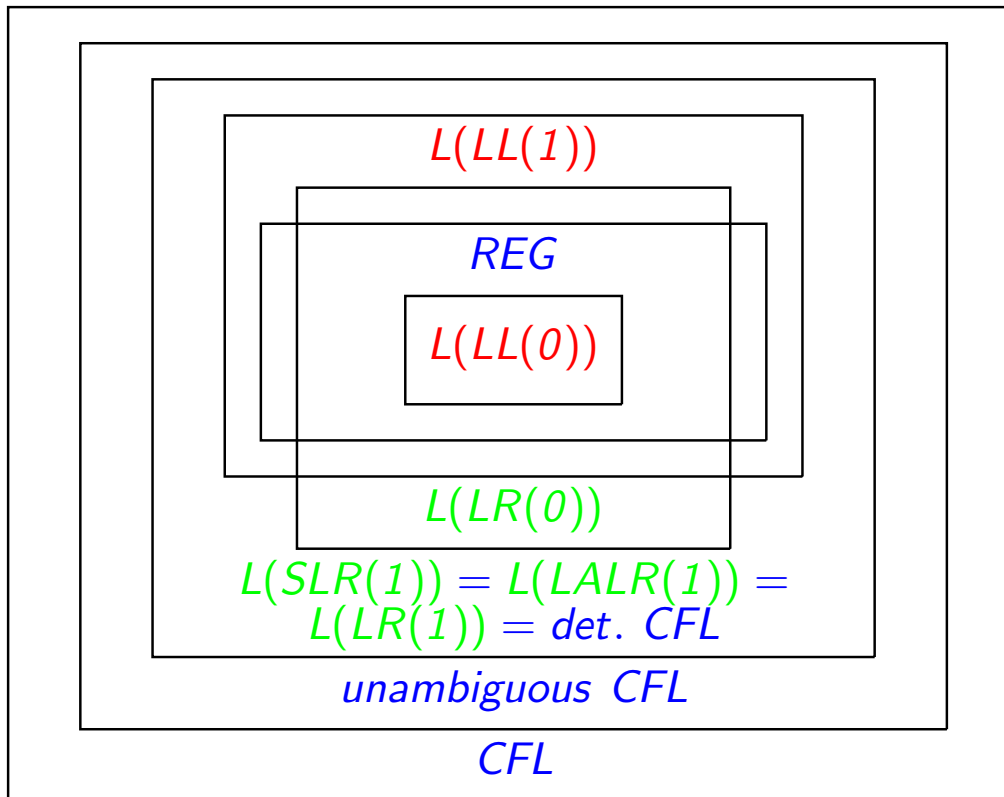
Moreover:

- $LL(k) \subsetneq LL(k+1)$
for every $k \in \mathbb{N}$
- $LR(k) \subsetneq LR(k+1)$
for every $k \in \mathbb{N}$
- $LL(k) \subseteq LR(k)$
for every $k \in \mathbb{N}$

Expressiveness of LL and LR Grammars

Overview of Language Classes

(cf. O. Mayer: *Syntaxanalyse*, BI-Verlag, 1978, p. 409ff)



Moreover:

- $L(LL(k)) \subsetneq L(LL(k+1)) \subsetneq L(LR(1))$
for every $k \in \mathbb{N}$
- $L(LR(k)) = L(LR(1))$
for every $k \geq 1$

Expressiveness of LL and LR Grammars

Why $REG \not\subseteq L(LR(0))$?

Definition 11.5

A language $L \subseteq \Sigma^*$ is called **prefix-free** if $L \cap L \cdot \Sigma^+ = \emptyset$, i.e., if no proper prefix of an element of L is again in L .

Expressiveness of LL and LR Grammars

Why $REG \not\subseteq L(LR(0))$?

Definition 11.5

A language $L \subseteq \Sigma^*$ is called **prefix-free** if $L \cap L \cdot \Sigma^+ = \emptyset$, i.e., if no proper prefix of an element of L is again in L .

Lemma 11.6

$G \in LR(0) \implies L(G)$ prefix-free

Expressiveness of LL and LR Grammars

Why $REG \not\subseteq L(LR(0))$?

Definition 11.5

A language $L \subseteq \Sigma^*$ is called **prefix-free** if $L \cap L \cdot \Sigma^+ = \emptyset$, i.e., if no proper prefix of an element of L is again in L .

Lemma 11.6

$G \in LR(0) \implies L(G)$ prefix-free

Proof.

on the board □

Expressiveness of LL and LR Grammars

Why $REG \not\subseteq L(LR(0))$?

Definition 11.5

A language $L \subseteq \Sigma^*$ is called **prefix-free** if $L \cap L \cdot \Sigma^+ = \emptyset$, i.e., if no proper prefix of an element of L is again in L .

Lemma 11.6

$G \in LR(0) \implies L(G)$ prefix-free

Proof.

on the board □

Corollary 11.7

$\{a, aa\} \in REG \setminus L(LR(0))$

Expressiveness of LL and LR Grammars

Why $REG \not\subseteq L(LR(0))$?

Definition 11.5

A language $L \subseteq \Sigma^*$ is called **prefix-free** if $L \cap L \cdot \Sigma^+ = \emptyset$, i.e., if no proper prefix of an element of L is again in L .

Lemma 11.6

$G \in LR(0) \implies L(G)$ prefix-free

Proof.

on the board □

Corollary 11.7

$\{a, aa\} \in REG \setminus L(LR(0))$

Conjecture: $L \in REG \setminus L(LR(0)) \implies L(G)$ not prefix-free?

Expressiveness of LL and LR Grammars

Why $REG \subseteq L(LL(1))$?

Lemma 11.8 (cf. Lecture 2)

Every $L \in REG$ can be recognized by a DFA.

Expressiveness of LL and LR Grammars

Why $REG \subseteq L(LL(1))$?

Lemma 11.8 (cf. Lecture 2)

Every $L \in REG$ can be recognized by a DFA.

Lemma 11.9

Every DFA can be transformed into an equivalent $LL(1)$ grammar.

Expressiveness of LL and LR Grammars

Why $REG \subseteq L(LL(1))$?

Lemma 11.8 (cf. Lecture 2)

Every $L \in REG$ can be recognized by a DFA.

Lemma 11.9

Every DFA can be transformed into an equivalent $LL(1)$ grammar.

Proof.

on the board □

LL and LR Parsing in Practice

Outline of Lecture 11

Recap: $LR(1)$ and $LALR(1)$ Parsing

Bottom-Up Parsing of Ambiguous Grammars

Generating Parsers Using `yacc` and `bison`

Expressiveness of LL and LR Grammars

LL and LR Parsing in Practice

LL and LR Parsing in Practice

LL and LR Parsing in Practice

In practice: use of $LL(1)$ or $LALR(1)$

LL and LR Parsing in Practice

LL and LR Parsing in Practice

In practice: use of $LL(1)$ or $LALR(1)$

Detailed comparison (cf. Fischer/LeBlanc: *Crafting a Compiler*, Benjamin/Cummings, 1988):

Simplicity: LL wins

- LL parsing technique easier to understand
- recursive-descent parser easier to debug than LALR action tables

LL and LR Parsing in Practice

LL and LR Parsing in Practice

In practice: use of $LL(1)$ or $LALR(1)$

Detailed comparison (cf. Fischer/LeBlanc: *Crafting a Compiler*, Benjamin/Cummings, 1988):

Simplicity: LL wins

Generality: LALR wins

- “almost” $LL(1) \subseteq LALR(1)$ (only pathological counterexamples)
- LL requires elimination of left recursion and left factorization

LL and LR Parsing in Practice

LL and LR Parsing in Practice

In practice: use of $LL(1)$ or $LALR(1)$

Detailed comparison (cf. Fischer/LeBlanc: *Crafting a Compiler*, Benjamin/Cummings, 1988):

Simplicity: LL wins

Generality: LALR wins

Semantic actions: (see semantic analysis) LL wins

- actions can be placed anywhere in LL parsers without causing conflicts
- in LALR: implicit ϵ -productions
 - \implies may generate conflicts

LL and LR Parsing in Practice

LL and LR Parsing in Practice

In practice: use of $LL(1)$ or $LALR(1)$

Detailed comparison (cf. Fischer/LeBlanc: *Crafting a Compiler*, Benjamin/Cummings, 1988):

Simplicity: LL wins

Generality: LALR wins

Semantic actions: (see semantic analysis) LL wins

Error handling: LL wins

- top-down approach provides context information
⇒ better basis for reporting and/or repairing errors

LL and LR Parsing in Practice

LL and LR Parsing in Practice

In practice: use of *LL(1)* or *LALR(1)*

Detailed comparison (cf. Fischer/LeBlanc: *Crafting a Compiler*, Benjamin/Cummings, 1988):

Simplicity: LL wins

Generality: LALR wins

Semantic actions: (see semantic analysis) LL wins

Error handling: LL wins

Parser size: comparable

- LL: action table
- LALR: action/goto table

LL and LR Parsing in Practice

LL and LR Parsing in Practice

In practice: use of $LL(1)$ or $LALR(1)$

Detailed comparison (cf. Fischer/LeBlanc: *Crafting a Compiler*, Benjamin/Cummings, 1988):

Simplicity: LL wins

Generality: LALR wins

Semantic actions: (see semantic analysis) LL wins

Error handling: LL wins

Parser size: comparable

Parsing speed: comparable

- both linear in length of input program
($LL(1)$: see Lemma 7.13 for ϵ -free case)
- concrete figures tool dependent

LL and LR Parsing in Practice

LL and LR Parsing in Practice

In practice: use of $LL(1)$ or $LALR(1)$

Detailed comparison (cf. Fischer/LeBlanc: *Crafting a Compiler*, Benjamin/Cummings, 1988):

Simplicity: LL wins

Generality: LALR wins

Semantic actions: (see semantic analysis) LL wins

Error handling: LL wins

Parser size: comparable

Parsing speed: comparable

Conclusion: choose LL when possible

(depending on available grammars and tools)