

Compiler Construction

Lecture 14: Syntactic Analysis IX (Practical Issues)

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/cc10/`

Winter semester 2010/11

- 1 Repetition: *LALR*(1) Parsing
- 2 More on *LALR*(1) Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc
- 5 Expressiveness of LL and LR Grammars
- 6 LL and LR Parsing in Practice

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 13.16, $|LALR(1)(G)| = |LR(0)(G)|$
(but $LALR(1)$ sets provide additional lookahead information)

Example (cf. Example 13.15)

$G_{LR} : S' \rightarrow S \quad S \rightarrow L=R \mid R \quad L \rightarrow *R \mid \mathbf{a} \quad R \rightarrow L$

$LR(0)(G_{LR}) :$

$I_0(\varepsilon) :$

$[S' \rightarrow \cdot S]$	$[S \rightarrow \cdot L=R]$
$[S \rightarrow \cdot R]$	$[L \rightarrow \cdot *R]$
$[L \rightarrow \cdot \mathbf{a}]$	$[R \rightarrow \cdot L]$

$I_1(S) :$

$[S' \rightarrow S \cdot]$

$I_2(L) :$

$[S \rightarrow L \cdot =R]$	$[R \rightarrow L \cdot]$
------------------------------	---------------------------

$I_3(R) :$

$[S \rightarrow R \cdot]$

$I_4(*) :$

$[L \rightarrow * \cdot R]$	$[R \rightarrow \cdot L]$
$[L \rightarrow * R \cdot]$	$[L \rightarrow \cdot \mathbf{a}]$

$I_5(\mathbf{a}) :$

$[L \rightarrow \mathbf{a} \cdot]$

$I_6(L=) :$

$[S \rightarrow L = \cdot R]$	$[R \rightarrow \cdot L]$
$[L \rightarrow * R \cdot]$	$[L \rightarrow \cdot \mathbf{a}]$

$I_7(*R) :$

$[L \rightarrow * R \cdot]$

$I_8(*L) :$

$[R \rightarrow L \cdot]$

$I_9(L=R) :$

$[S \rightarrow L=R \cdot]$

$LALR(1)(G_{LR}) :$

$I_0'' := I_0' :$

$[S' \rightarrow \cdot S, \varepsilon]$	$[S \rightarrow \cdot L=R, \varepsilon]$
$[S \rightarrow \cdot R, \varepsilon]$	$[L \rightarrow \cdot *R, =/\varepsilon]$
$[L \rightarrow \cdot \mathbf{a}, =/\varepsilon]$	$[R \rightarrow \cdot L, \varepsilon]$

$I_1'' := I_1' :$

$[S' \rightarrow S \cdot, \varepsilon]$

$I_2'' := I_2' :$

$[S \rightarrow L \cdot =R, \varepsilon]$	$[R \rightarrow L \cdot, \varepsilon]$
---	--

$I_3'' := I_3' :$

$[S \rightarrow R \cdot, \varepsilon]$
--

$I_4'' := I_4' \cup I_{11}' :$

$[L \rightarrow * \cdot R, =/\varepsilon]$	$[R \rightarrow \cdot L, =/\varepsilon]$
$[L \rightarrow * R \cdot, =/\varepsilon]$	$[L \rightarrow \cdot \mathbf{a}, =/\varepsilon]$

$I_5'' := I_5' \cup I_{12}' :$

$[L \rightarrow \mathbf{a} \cdot, =/\varepsilon]$

$I_6'' := I_6' :$

$[S \rightarrow L = \cdot R, \varepsilon]$	$[R \rightarrow \cdot L, \varepsilon]$
$[L \rightarrow * R \cdot, \varepsilon]$	$[L \rightarrow \cdot \mathbf{a}, \varepsilon]$

$I_7'' := I_7' \cup I_{13}' :$

$[L \rightarrow * R \cdot, =/\varepsilon]$
--

$I_8'' := I_8' \cup I_{10}' :$

$[R \rightarrow L \cdot, =/\varepsilon]$
--

$I_9'' := I_9' :$

$[S \rightarrow L=R \cdot, \varepsilon]$
--

- 1 Repetition: *LALR*(1) Parsing
- 2 More on *LALR*(1) Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc
- 5 Expressiveness of LL and LR Grammars
- 6 LL and LR Parsing in Practice

LALR(1) Conflicts

But: merging of $LR(1)$ sets can produce new conflicts (also see exercises):

Example 14.1

$G : S' \rightarrow S \quad S \rightarrow aAd \mid bBd \mid aBe \mid bAe \quad A \rightarrow c \quad B \rightarrow c$

LALR(1) Conflicts

But: merging of $LR(1)$ sets can produce new conflicts (also see exercises):

Example 14.1

$G : S' \rightarrow S \quad S \rightarrow \mathbf{aAd} \mid \mathbf{bBd} \mid \mathbf{aBe} \mid \mathbf{bAe} \quad A \rightarrow \mathbf{c} \quad B \rightarrow \mathbf{c}$

$LR(1)(\varepsilon) :$	$[S' \rightarrow \cdot S, \varepsilon]$	$[S \rightarrow \cdot \mathbf{aAd}, \varepsilon]$	$[S \rightarrow \cdot \mathbf{bBd}, \varepsilon]$	$[S \rightarrow \cdot \mathbf{aBe}, \varepsilon]$
	$[S \rightarrow \cdot \mathbf{bAe}, \varepsilon]$			
$LR(1)(S) :$	$[S' \rightarrow S \cdot, \varepsilon]$			
$LR(1)(\mathbf{a}) :$	$[S \rightarrow \mathbf{a} \cdot \mathbf{Ad}, \varepsilon]$	$[S \rightarrow \mathbf{a} \cdot \mathbf{Be}, \varepsilon]$	$[A \rightarrow \cdot \mathbf{c}, \mathbf{d}]$	$[B \rightarrow \cdot \mathbf{c}, \mathbf{e}]$
$LR(1)(\mathbf{b}) :$	$[S \rightarrow \mathbf{b} \cdot \mathbf{Bd}, \varepsilon]$	$[S \rightarrow \mathbf{b} \cdot \mathbf{Ae}, \varepsilon]$	$[B \rightarrow \cdot \mathbf{c}, \mathbf{d}]$	$[A \rightarrow \cdot \mathbf{c}, \mathbf{e}]$
$LR(1)(\mathbf{aA}) :$	$[S \rightarrow \mathbf{aA} \cdot \mathbf{d}, \varepsilon]$		$LR(1)(\mathbf{aB}) : [S \rightarrow \mathbf{aB} \cdot \mathbf{e}, \varepsilon]$	
$LR(1)(\mathbf{ac}) :$	$[A \rightarrow \mathbf{c} \cdot, \mathbf{d}]$	$[B \rightarrow \mathbf{c} \cdot, \mathbf{e}]$		
$LR(1)(\mathbf{bB}) :$	$[S \rightarrow \mathbf{bB} \cdot \mathbf{d}, \varepsilon]$		$LR(1)(\mathbf{bA}) : [S \rightarrow \mathbf{bA} \cdot \mathbf{e}, \varepsilon]$	
$LR(1)(\mathbf{bc}) :$	$[B \rightarrow \mathbf{c} \cdot, \mathbf{d}]$	$[A \rightarrow \mathbf{c} \cdot, \mathbf{e}]$		
$LR(1)(\mathbf{aAd}) :$	$[S \rightarrow \mathbf{aAd} \cdot, \varepsilon]$		$LR(1)(\mathbf{aBe}) : [S \rightarrow \mathbf{aBe} \cdot, \varepsilon]$	
$LR(1)(\mathbf{bBd}) :$	$[S \rightarrow \mathbf{bBd} \cdot, \varepsilon]$		$LR(1)(\mathbf{bAe}) : [S \rightarrow \mathbf{bAe} \cdot, \varepsilon]$	

LALR(1) Conflicts

But: merging of $LR(1)$ sets can produce new conflicts (also see exercises):

Example 14.1

$G : S' \rightarrow S \quad S \rightarrow \mathbf{aAd} \mid \mathbf{bBd} \mid \mathbf{aBe} \mid \mathbf{bAe} \quad A \rightarrow \mathbf{c} \quad B \rightarrow \mathbf{c}$

$LR(1)(\varepsilon) :$ $[S' \rightarrow \cdot S, \varepsilon] \quad [S \rightarrow \cdot \mathbf{aAd}, \varepsilon] \quad [S \rightarrow \cdot \mathbf{bBd}, \varepsilon] \quad [S \rightarrow \cdot \mathbf{aBe}, \varepsilon]$
 $[S \rightarrow \cdot \mathbf{bAe}, \varepsilon]$

$LR(1)(S) :$ $[S' \rightarrow S \cdot, \varepsilon]$

$LR(1)(\mathbf{a}) :$ $[S \rightarrow \mathbf{a} \cdot \mathbf{Ad}, \varepsilon] \quad [S \rightarrow \mathbf{a} \cdot \mathbf{Be}, \varepsilon] \quad [A \rightarrow \cdot \mathbf{c}, \mathbf{d}] \quad [B \rightarrow \cdot \mathbf{c}, \mathbf{e}]$

$LR(1)(\mathbf{b}) :$ $[S \rightarrow \mathbf{b} \cdot \mathbf{Bd}, \varepsilon] \quad [S \rightarrow \mathbf{b} \cdot \mathbf{Ae}, \varepsilon] \quad [B \rightarrow \cdot \mathbf{c}, \mathbf{d}] \quad [A \rightarrow \cdot \mathbf{c}, \mathbf{e}]$

$LR(1)(\mathbf{aA}) :$ $[S \rightarrow \mathbf{aA} \cdot \mathbf{d}, \varepsilon]$ $LR(1)(\mathbf{aB}) :$ $[S \rightarrow \mathbf{aB} \cdot \mathbf{e}, \varepsilon]$

$LR(1)(\mathbf{ac}) :$ $[A \rightarrow \mathbf{c} \cdot, \mathbf{d}] \quad [B \rightarrow \mathbf{c} \cdot, \mathbf{e}]$

$LR(1)(\mathbf{bB}) :$ $[S \rightarrow \mathbf{bB} \cdot \mathbf{d}, \varepsilon]$ $LR(1)(\mathbf{bA}) :$ $[S \rightarrow \mathbf{bA} \cdot \mathbf{e}, \varepsilon]$

$LR(1)(\mathbf{bc}) :$ $[B \rightarrow \mathbf{c} \cdot, \mathbf{d}] \quad [A \rightarrow \mathbf{c} \cdot, \mathbf{e}]$

$LR(1)(\mathbf{aAd}) :$ $[S \rightarrow \mathbf{aAd} \cdot, \varepsilon]$ $LR(1)(\mathbf{aBe}) :$ $[S \rightarrow \mathbf{aBe} \cdot, \varepsilon]$

$LR(1)(\mathbf{bBd}) :$ $[S \rightarrow \mathbf{bBd} \cdot, \varepsilon]$ $LR(1)(\mathbf{bAe}) :$ $[S \rightarrow \mathbf{bAe} \cdot, \varepsilon]$

no conflicts $\implies G \in LR(1)$

LALR(1) Conflicts

But: merging of $LR(1)$ sets can produce new conflicts (also see exercises):

Example 14.1

$G : S' \rightarrow S \quad S \rightarrow \mathbf{aAd} \mid \mathbf{bBd} \mid \mathbf{aBe} \mid \mathbf{bAe} \quad A \rightarrow \mathbf{c} \quad B \rightarrow \mathbf{c}$

$LR(1)(\varepsilon) :$	$[S' \rightarrow \cdot S, \varepsilon]$	$[S \rightarrow \cdot \mathbf{aAd}, \varepsilon]$	$[S \rightarrow \cdot \mathbf{bBd}, \varepsilon]$	$[S \rightarrow \cdot \mathbf{aBe}, \varepsilon]$
	$[S \rightarrow \cdot \mathbf{bAe}, \varepsilon]$			
$LR(1)(S) :$	$[S' \rightarrow S \cdot, \varepsilon]$			
$LR(1)(\mathbf{a}) :$	$[S \rightarrow \mathbf{a} \cdot \mathbf{Ad}, \varepsilon]$	$[S \rightarrow \mathbf{a} \cdot B\mathbf{e}, \varepsilon]$	$[A \rightarrow \cdot \mathbf{c}, \mathbf{d}]$	$[B \rightarrow \cdot \mathbf{c}, \mathbf{e}]$
$LR(1)(\mathbf{b}) :$	$[S \rightarrow \mathbf{b} \cdot \mathbf{Bd}, \varepsilon]$	$[S \rightarrow \mathbf{b} \cdot A\mathbf{e}, \varepsilon]$	$[B \rightarrow \cdot \mathbf{c}, \mathbf{d}]$	$[A \rightarrow \cdot \mathbf{c}, \mathbf{e}]$
$LR(1)(\mathbf{aA}) :$	$[S \rightarrow \mathbf{aA} \cdot \mathbf{d}, \varepsilon]$		$LR(1)(\mathbf{aB}) :$	$[S \rightarrow \mathbf{aB} \cdot \mathbf{e}, \varepsilon]$
$LR(1)(\mathbf{ac}) :$	$[A \rightarrow \mathbf{c} \cdot, \mathbf{d}]$	$[B \rightarrow \mathbf{c} \cdot, \mathbf{e}]$		
$LR(1)(\mathbf{bB}) :$	$[S \rightarrow \mathbf{bB} \cdot \mathbf{d}, \varepsilon]$		$LR(1)(\mathbf{bA}) :$	$[S \rightarrow \mathbf{bA} \cdot \mathbf{e}, \varepsilon]$
$LR(1)(\mathbf{bc}) :$	$[B \rightarrow \mathbf{c} \cdot, \mathbf{d}]$	$[A \rightarrow \mathbf{c} \cdot, \mathbf{e}]$		
$LR(1)(\mathbf{aAd}) :$	$[S \rightarrow \mathbf{aAd} \cdot, \varepsilon]$		$LR(1)(\mathbf{aBe}) :$	$[S \rightarrow \mathbf{aBe} \cdot, \varepsilon]$
$LR(1)(\mathbf{bBd}) :$	$[S \rightarrow \mathbf{bBd} \cdot, \varepsilon]$		$LR(1)(\mathbf{bAe}) :$	$[S \rightarrow \mathbf{bAe} \cdot, \varepsilon]$

no conflicts $\implies G \in LR(1)$

$LR(1)(\mathbf{ac}) \sim_0 LR(1)(\mathbf{bc})$, but $LR(1)(\mathbf{ac}) \cup LR(1)(\mathbf{bc})$ has conflicts

$\implies G \notin LALR(1)$

Naive algorithm to construct $LALR(1)$ parser for $G \in CFG_\Sigma$:

- 1 Construct $LR(1)(G)$
- 2 Determine and merge $LR(0)$ -equivalent $LR(1)$ sets

Problem: no reduction of **peak space requirement**

Efficient Construction of $LALR(1)$ Parsers

Naive algorithm to construct $LALR(1)$ parser for $G \in CFG_\Sigma$:

- 1 Construct $LR(1)(G)$
- 2 Determine and merge $LR(0)$ -equivalent $LR(1)$ sets

Problem: no reduction of **peak space requirement**

Idea of **improved algorithm** (see Aho/Lam/Sethi/Ullman: *Compilers: Principles, Techniques, and Tools*, 2nd ed., p. 270ff):

- 1 Represent each set of items by its **kernel**, i.e., by the items of the form $[S' \rightarrow \cdot S, \varepsilon]$ or $[A \rightarrow \beta_1 \cdot \beta_2, x]$ where $\beta_1 \neq \varepsilon$
- 2 Construct $LALR(1)$ kernels from $LR(0)$ kernels similarly to $LR(1)$ items
- 3 Compute $LALR(1)$ sets by taking the ε -closure

(applied in yacc parser generator)

- 1 Repetition: *LALR*(1) Parsing
- 2 More on *LALR*(1) Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc
- 5 Expressiveness of LL and LR Grammars
- 6 LL and LR Parsing in Practice

Ambiguous Grammars

Reminder (Definition 6.6): 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**.

Ambiguous Grammars

Reminder (Definition 6.6): 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 14.2

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

Ambiguous Grammars

Reminder (Definition 6.6): 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 14.2

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.

Ambiguous Grammars

Reminder (Definition 6.6): 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 14.2

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., $\beta \neq \beta'$):

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

Ambiguous Grammars

Reminder (Definition 6.6): 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 14.2

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 10.9 yields that $\beta = \beta'$. Contradiction □

Ambiguous Grammars

Reminder (Definition 6.6): 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 14.2

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 10.9 yields that $\beta = \beta'$. Contradiction □

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

Example 14.3 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad E \rightarrow E+E \mid E * E \mid a$

Example 14.3 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad E \rightarrow E + E \mid E * E \mid a$

Precedence: $*$ $>$ $+$ **Associativity:** left

(thus: $a + a * a + a := (a + (a * a)) + a$)

Example 14.3 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad E \rightarrow E+E \mid E * E \mid a$

Precedence: $*$ $>$ $+$ **Associativity:** left

(thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G)$:

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

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

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

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

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

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

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

Example 14.3 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad E \rightarrow E+E \mid E * E \mid a$

Precedence: $*$ $>$ $+$ **Associativity:** left

(thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G)$:

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

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

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

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

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

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

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

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

Example 14.3 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad E \rightarrow E+E \mid E * E \mid a$

Precedence: $*$ $>$ $+$ **Associativity:** left

(thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G)$:

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

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

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

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

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

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

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

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

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

Bottom-Up Parsing of Ambiguous Grammars I

Example 14.3 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad E \rightarrow E+E \mid E * E \mid a$

Precedence: $*$ $>$ $+$ **Associativity:** left

(thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G)$:

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

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

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

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

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

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

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

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

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

Solution:

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

Bottom-Up Parsing of Ambiguous Grammars I

Example 14.3 (Simple arithmetic expressions)

$G : E' \rightarrow E \quad E \rightarrow E+E \mid E * E \mid a$

Precedence: $*$ $>$ $+$ **Associativity:** left

(thus: $a+a*a+a := (a+(a*a))+a$)

$LR(0)(G)$:

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

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

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

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

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

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

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

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

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

Solution:

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

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

Example 14.4 (“Dangling `else`”)

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

Example 14.4 (“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`

Example 14.4 (“Dangling *else*”)

$G : S' \rightarrow S \quad S \rightarrow \textcolor{blue}{i}S\textcolor{blue}{e}S \mid \textcolor{blue}{i}S \mid \textcolor{blue}{a}$

Ambiguity: $\textcolor{blue}{iiaea} := (1) \textcolor{blue}{i}(\textcolor{blue}{iaea})$ (common) or (2) $\textcolor{blue}{i}(\textcolor{blue}{ia})\textcolor{blue}{ea}$

$LR(0)(G):$

$I_0 := LR(0)(\varepsilon) :$	$[S' \rightarrow \cdot S]$	$[S \rightarrow \cdot \textcolor{blue}{i}S\textcolor{blue}{e}S]$	$[S \rightarrow \cdot \textcolor{blue}{i}S]$
	$[S \rightarrow \cdot \textcolor{blue}{a}]$		
$I_1 := LR(0)(S) :$	$[S' \rightarrow S \cdot]$		
$I_2 := LR(0)(\textcolor{blue}{i}) :$	$[S \rightarrow \textcolor{blue}{i} \cdot S\textcolor{blue}{e}S]$	$[S \rightarrow \textcolor{blue}{i} \cdot S]$	$[S \rightarrow \cdot \textcolor{blue}{i}S\textcolor{blue}{e}S]$
	$[S \rightarrow \cdot \textcolor{blue}{i}S]$	$[S \rightarrow \cdot \textcolor{blue}{a}]$	
$I_3 := LR(0)(\textcolor{blue}{a}) :$	$[S \rightarrow \textcolor{blue}{a} \cdot]$		
$I_4 := LR(0)(\textcolor{blue}{i}S) :$	$[S \rightarrow \textcolor{blue}{i}S \cdot \textcolor{blue}{e}S]$	$[S \rightarrow \textcolor{blue}{i}S \cdot]$	
$I_5 := LR(0)(\textcolor{blue}{i}S\textcolor{blue}{e}) :$	$[S \rightarrow \textcolor{blue}{i}S\textcolor{blue}{e} \cdot S]$	$[S \rightarrow \cdot \textcolor{blue}{i}S\textcolor{blue}{e}S]$	$[S \rightarrow \cdot \textcolor{blue}{i}S]$
	$[S \rightarrow \cdot \textcolor{blue}{a}]$		
$I_6 := LR(0)(\textcolor{blue}{i}S\textcolor{blue}{e}S) :$	$[S \rightarrow \textcolor{blue}{i}S\textcolor{blue}{e}S \cdot]$		

Example 14.4 (“Dangling **else**”)

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

Ambiguity: $\mathbf{iaea} := (1) \mathbf{i(iaea)}$ (common) or $(2) \mathbf{i(ia)ea}$

$LR(0)(G):$

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

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

Example 14.4 (“Dangling *else*”)

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

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

$LR(0)(G)$:

$I_0 := LR(0)(\varepsilon) :$

$[S' \rightarrow \cdot S]$	$[S \rightarrow \cdot iSeS]$	$[S \rightarrow \cdot iS]$
$[S \rightarrow \cdot a]$		

$I_1 := LR(0)(S) :$

$[S' \rightarrow S \cdot]$		
----------------------------	--	--

$I_2 := LR(0)(i) :$

$[S \rightarrow i \cdot SeS]$	$[S \rightarrow i \cdot S]$	$[S \rightarrow \cdot iSeS]$
$[S \rightarrow \cdot iS]$	$[S \rightarrow \cdot a]$	

$I_3 := LR(0)(a) :$

$[S \rightarrow a \cdot]$		
---------------------------	--	--

$I_4 := LR(0)(iS) :$

$[S \rightarrow iS \cdot eS]$	$[S \rightarrow iS \cdot]$	
-------------------------------	----------------------------	--

$I_5 := LR(0)(iSe) :$

$[S \rightarrow iSe \cdot S]$	$[S \rightarrow \cdot iSeS]$	$[S \rightarrow \cdot iS]$
$[S \rightarrow \cdot a]$		

$I_6 := LR(0)(iSeS) :$

$[S \rightarrow iSeS \cdot]$		
------------------------------	--	--

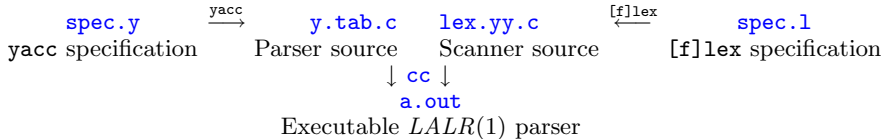
Conflict in I_4 : $e \in \text{fo}(S) \implies$ not $SLR(1)$ -solvable

Solution (1): $\text{act}(I_4, e) := \text{shift}$

- 1 Repetition: *LALR*(1) Parsing
- 2 More on *LALR*(1) Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using *yacc*
- 5 Expressiveness of LL and LR Grammars
- 6 LL and LR Parsing in Practice

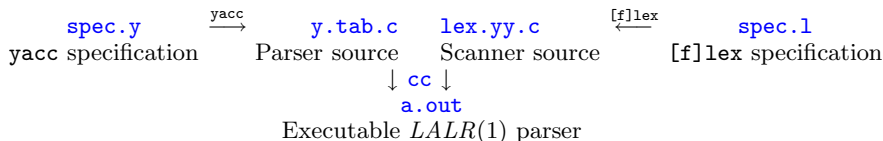
The yacc Tool

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



The yacc Tool

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



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

Declarations (optional)

%%

Rules

%%

Auxiliary procedures (optional)

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

- 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
$$\begin{array}{rcl} A & : & \alpha_1 \quad \{Action_1\} \\ & | & \alpha_2 \quad \{Action_2\} \\ & & \vdots \\ & | & \alpha_n \quad \{Action_n\}; \end{array}$$
- 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**

- 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
$$\begin{array}{rcl} A & : & \alpha_1 \quad \{Action_1\} \\ & | & \alpha_2 \quad \{Action_2\} \\ & & \vdots \\ & | & \alpha_n \quad \{Action_n\}; \end{array}$$
- 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, ...

Example: Simple Desk Calculator I

```
%{/* SLR(1) grammar for arithmetic expressions (Example 12.9) */
#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;
}
```

Example: Simple Desk Calculator II

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

An Ambiguous Grammar I

```
%{/* Ambiguous grammar for arithm. expressions (Ex. 14.3) */  
    #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;  
}
```

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)]
```

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 14.4) correctly
- also adequate for strong following weak operator ($*$ after $+$; Example 14.3) and for right-associative operators
- not appropriate for weak following strong operator and for left-associative binary operators
($\implies$ reduce; see Example 14.3)

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 14.4) correctly
- also adequate for strong following weak operator (***** after **+**; Example 14.3) and for right-associative operators
- not appropriate for weak following strong operator and for left-associative binary operators
($\implies$ reduce; see Example 14.3)

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:

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

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

Precedences and Associativities in yacc I

General mechanism for resolving conflicts:

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

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

Example 14.5

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

$\wedge$ (right associative) binds stronger than $*$ and $/$ (left associative), which in turn bind stronger than $+$ and $-$ (left associative)

Precedences and Associativities in yacc II

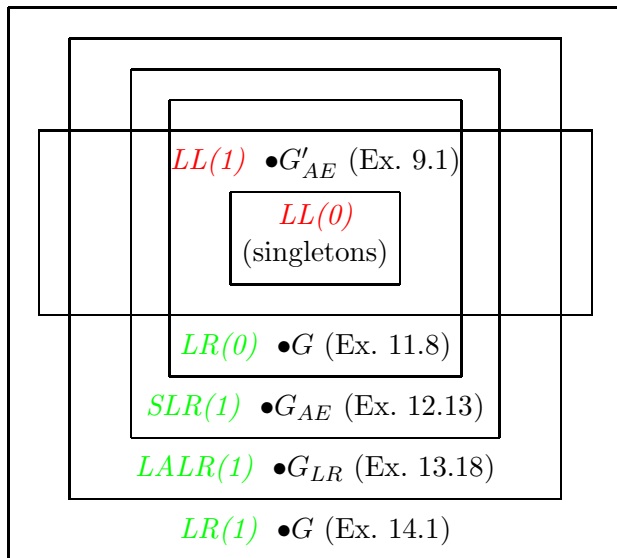
```
%{ /* Ambiguous grammar for arithmetic expressions
    with precedences and associativities */
#include <stdio.h>
#include <ctype.h>
%}
%token DIGIT
%left '+'
%left '*'
%%
line    : expr '\n' { printf("%d\n", $1); };
expr    : expr '+' expr { $$ = $1 + $3; }
        | expr '*' expr { $$ = $1 * $3;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

- 1 Repetition: *LALR*(1) Parsing
- 2 More on *LALR*(1) Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc
- 5 Expressiveness of LL and LR Grammars
- 6 LL and LR Parsing in Practice

Overview of Grammar Classes

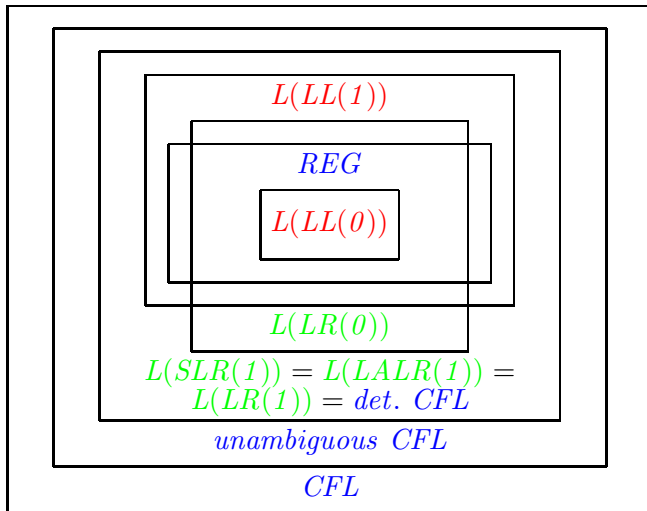


Moreover:

- $LL(k) \subsetneq LL(k+1)$
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 $k \in \mathbb{N}$

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 $k \geq 1$

- 1 Repetition: *LALR*(1) Parsing
- 2 More on *LALR*(1) Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc
- 5 Expressiveness of LL and LR Grammars
- 6 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

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

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

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

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
 $\implies$ better basis for reporting and/or repairing errors

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

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 9.10 for ε -free case)
- concrete figures tool dependent

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)