Compiler Construction

Lecture 12: Syntactic Analysis VIII ($LALR(1)$ 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

Outline

The $SLR(1)$ Action Function

Definition ($SLR(1)$ action function)

The $SLR(1)$ action function

$$\mathrm{act} : LR(0)(G) \times \Sigma_\varepsilon \to \{\mathsf{red}\, i \mid i \in [p]\} \cup \{\mathsf{shift}, \mathsf{accept}, \mathsf{error}\}$$

is defined by

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

Definition ($SLR(1)$ grammar)

A grammar $G \in CFG_\Sigma$ has the $SLR(1)$ property (notation: $G \in SLR(1)$) if its $SLR(1)$ action function is well defined.

Together, act and the $LR(0)$ goto function (cf. Definition 10.1) form the $SLR(1)$ parsing table of G.

$LR(1)$ Items and Sets I

Observation: not every element of $\mathrm{fo}(A)$ can follow every occurrence of A

$\Longrightarrow$ 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 $S' \to S$.

- If $S' \Rightarrow_r^* \alpha A a w \Rightarrow_r \alpha\beta_1\beta_2 a w$, then $[A \to \beta_1 \cdot \beta_2, a]$ is called an $LR(1)$ item for $\alpha\beta_1$.
- If $S' \Rightarrow_r^* \alpha A \Rightarrow_r \alpha\beta_1\beta_2$, then $[A \to \beta_1 \cdot \beta_2, \varepsilon]$ 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.
- $LR(1)(G) := \{LR(1)(\gamma) \mid \gamma \in X^*\}$.

$LR(0)$ vs. $LR(1)$

Example

$$
\begin{array}{lll}
G_{LR}: & S' \to S & S \to L{=}R \mid R \\
 & L \to {*}R \mid \mathtt{a} & R \to L
\end{array}
$$

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

$$
\begin{array}{lll}
I_0(\varepsilon): & [S' \to \cdot S] & [S \to \cdot L{=}R] \\
 & [S \to \cdot R] & [L \to \cdot {*}R] \\
 & [L \to \cdot \mathtt{a}] & [R \to \cdot L] \\
I_1(S): & [S' \to S\cdot] & \\
I_2(L): & [S \to L \cdot {=}R] & [R \to L\cdot] \\
I_3(R): & [S \to R\cdot] & \\
I_4({*}): & [L \to {*} \cdot R] & [R \to \cdot L] \\
 & [L \to \cdot {*}R] & [L \to \cdot \mathtt{a}] \\
I_5(\mathtt{a}): & [L \to \mathtt{a}\cdot] & \\
I_6(L{=}): & [S \to L{=} \cdot R] & [R \to \cdot L] \\
 & [L \to \cdot {*}R] & [L \to \cdot \mathtt{a}] \\
I_7({*}R): & [L \to {*}R\cdot] & \\
I_8({*}L): & [R \to L\cdot] & \\
I_9(L{=}R): & [S \to L{=}R\cdot] &
\end{array}
$$

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

$$
\begin{array}{lll}
I'_0(\varepsilon): & [S' \to \cdot S, \varepsilon] & [S \to \cdot L{=}R, \varepsilon] \\
 & [S \to \cdot R, \varepsilon] & [L \to \cdot {*}R, {=}] \\
 & [L \to \cdot \mathtt{a}, {=}] & [R \to \cdot L, \varepsilon] \\
 & [L \to \cdot {*}R, \varepsilon] & [L \to \cdot \mathtt{a}, \varepsilon] \\
I'_1(S): & [S' \to S\cdot, \varepsilon] & \\
I'_2(L): & [S \to L \cdot {=}R, \varepsilon] & [R \to L\cdot, \varepsilon] \\
I'_3(R): & [S \to R\cdot, \varepsilon] & \\
I'_4({*}): & [L \to {*} \cdot R, {=}] & [L \to {*} \cdot R, \varepsilon] \\
 & [R \to \cdot L, {=}] & [R \to \cdot L, \varepsilon] \\
 & [L \to \cdot {*}R, {=}] & [L \to \cdot \mathtt{a}, {=}] \\
 & [L \to \cdot {*}R, \varepsilon] & [L \to \cdot \mathtt{a}, \varepsilon] \\
I'_5(\mathtt{a}): & [L \to \mathtt{a}\cdot, {=}] & [L \to \mathtt{a}\cdot, \varepsilon] \\
I'_6(L{=}): & [S \to L{=} \cdot R, \varepsilon] & [R \to \cdot L, \varepsilon] \\
 & [L \to \cdot {*}R, \varepsilon] & [L \to \cdot \mathtt{a}, \varepsilon] \\
I'_7({*}R): & [L \to {*}R\cdot, {=}] & [L \to {*}R\cdot, \varepsilon] \\
I'_8({*}L): & [R \to L\cdot, {=}] & [R \to L\cdot, \varepsilon] \\
I'_9(L{=}R): & [S \to L{=}R\cdot, \varepsilon] & \\
I'_{10}(L{=}L): & [R \to L\cdot, \varepsilon] & \\
I'_{11}(L{=}{*}): & [L \to {*} \cdot R, \varepsilon] & [R \to \cdot L, \varepsilon] \\
 & [L \to \cdot {*}R, \varepsilon] & [L \to \cdot \mathtt{a}, \varepsilon] \\
I'_{12}(L{=}\mathtt{a}): & [L \to \mathtt{a}\cdot, \varepsilon] & \\
I'_{13}(L{=}{*}R): & [L \to {*}R\cdot, \varepsilon] &
\end{array}
$$

The $LR(1)$ Action Function

Definition ($LR(1)$ action function)

The $LR(1)$ action function

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

is defined by

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

Corollary

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

Outline

$LALR(1)$ Parsing

- **Motivation:** resolving conflicts using $LR(1)$ too expensive
- Example 11.7/11.13: $|LR(0)(G_{LR})| = 11$, $|LR(1)(G_{LR})| = 15$
- A. Johnstone, E. Scott: *Generalised Reduction Modified LR Parsing for Domain Specific Language Prototyping*, HICSS '02, IEEE, 2002, `http://doi.ieeecomputersociety.org/10.1109/HICSS.2002.994495`:

Grammar	$\|LR(0)(G)\|$	$\|LR(1)(G)\|$
Ansi-C	381	1788
Pascal	368	1395

- **Observation:** potential redundancy by containment of $LR(0)$ sets in $LR(1)$ sets (cf. Corollary 11.9)

Definition 12.1 ($LR(0)$ equivalence)

Let $\mathrm{lr}_0 : LR(1)(G) \to LR(0)(G)$ be defined by

$$\mathrm{lr}_0(I) := \{[A \to \beta_1 \cdot \beta_2] \mid [A \to \beta_1 \cdot \beta_2, x] \in I\}.$$

Two sets $I_1, I_2 \in LR(1)(G)$ are called $LR(0)$ equivalent (notation: $I_1 \sim_0 I_2$) if $\mathrm{lr}_0(I_1) = \mathrm{lr}_0(I_2)$.

$LR(0)$ Equivalence

Example 12.2 (cf. Example 11.7/11.13)

$$
\begin{array}{lll}
G_{LR}: & S' \to S & S \to L\texttt{=}R \mid R \\
 & L \to *R \mid \mathtt{a} & R \to L
\end{array}
$$

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

$$
\begin{array}{lll}
I_0(\varepsilon): & [S' \to \cdot S] & [S \to \cdot L\texttt{=}R] \\
 & [S \to \cdot R] & [L \to \cdot *R] \\
 & [L \to \cdot \mathtt{a}] & [R \to \cdot L] \\
I_1(S): & [S' \to S\cdot] & \\
I_2(L): & [S \to L \cdot \texttt{=}R] & [R \to L\cdot] \\
I_3(R): & [S \to R\cdot] & \\
\color{red}{I_4(*):} & [L \to * \cdot R] & [R \to \cdot L] \\
 & [L \to \cdot *R] & [L \to \cdot \mathtt{a}] \\
\color{red}{I_5(\mathtt{a}):} & [L \to \mathtt{a}\cdot] & \\
I_6(L\texttt{=}): & [S \to L\texttt{=} \cdot R] & [R \to \cdot L] \\
 & [L \to \cdot *R] & [L \to \cdot \mathtt{a}] \\
\color{red}{I_7(*R):} & [L \to *R\cdot] & \\
\color{red}{I_8(*L):} & [R \to L\cdot] & \\
I_9(L\texttt{=}R): & [S \to L\texttt{=}R\cdot] &
\end{array}
$$

$$
\Longrightarrow \quad
\color{red}{
\begin{array}{lcl}
I'_4 & \sim_0 & I'_{11} \\
I'_5 & \sim_0 & I'_{12} \\
I'_7 & \sim_0 & I'_{13} \\
I'_8 & \sim_0 & I'_{10}
\end{array}}
$$

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

$$
\begin{array}{lll}
I'_0(\varepsilon): & [S' \to \cdot S, \varepsilon] & [S \to \cdot L\texttt{=}R, \varepsilon] \\
 & [S \to \cdot R, \varepsilon] & [L \to \cdot *R, \texttt{=}] \\
 & [L \to \cdot \mathtt{a}, \texttt{=}] & [R \to \cdot L, \varepsilon] \\
 & [L \to \cdot *R, \varepsilon] & [L \to \cdot \mathtt{a}, \varepsilon] \\
I'_1(S): & [S' \to S\cdot, \varepsilon] & \\
I'_2(L): & [S \to L \cdot \texttt{=}R, \varepsilon] & [R \to L\cdot, \varepsilon] \\
I'_3(R): & [S \to R\cdot, \varepsilon] & \\
\color{red}{I'_4(*):} & [L \to * \cdot R, \texttt{=}] & [L \to * \cdot R, \varepsilon] \\
 & [R \to \cdot L, \texttt{=}] & [R \to \cdot L, \varepsilon] \\
 & [L \to \cdot *R, \texttt{=}] & [L \to \cdot \mathtt{a}, \texttt{=}] \\
 & [L \to \cdot *R, \varepsilon] & [L \to \cdot \mathtt{a}, \varepsilon] \\
\color{red}{I'_5(\mathtt{a}):} & [L \to \mathtt{a}\cdot, \texttt{=}] & [L \to \mathtt{a}\cdot, \varepsilon] \\
I'_6(L\texttt{=}): & [S \to L\texttt{=} \cdot R, \varepsilon] & [R \to \cdot L, \varepsilon] \\
 & [L \to \cdot *R, \varepsilon] & [L \to \cdot \mathtt{a}, \varepsilon] \\
\color{red}{I'_7(*R):} & [L \to *R\cdot, \texttt{=}] & [L \to *R\cdot, \varepsilon] \\
\color{red}{I'_8(*L):} & [R \to L\cdot, \texttt{=}] & [R \to L\cdot, \varepsilon] \\
I'_9(L\texttt{=}R): & [S \to L\texttt{=}R\cdot, \varepsilon] & \\
\color{red}{I'_{10}(L\texttt{=}L):} & [R \to L\cdot, \varepsilon] & \\
\color{red}{I'_{11}(L\texttt{=}*):} & [L \to * \cdot R, \varepsilon] & [R \to \cdot L, \varepsilon] \\
 & [L \to \cdot *R, \varepsilon] & [L \to \cdot \mathtt{a}, \varepsilon] \\
\color{red}{I'_{12}(L\texttt{=}\mathtt{a}):} & [L \to \mathtt{a}\cdot, \varepsilon] & \\
\color{red}{I'_{13}(L\texttt{=}*R):} & [L \to *R\cdot, \varepsilon] &
\end{array}
$$

$LALR(1)$ Sets I

Corollary 12.3

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 12.4 ($LALR(1)$ 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 12.3, $|LALR(1)(G)| = |LR(0)(G)|$
(but $LALR(1)$ sets provide additional lookahead information)

Example 12.5 (cf. Example 12.2)

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

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

$I_0(\varepsilon)$:	$[S' \to \cdot S]$	$[S \to \cdot L{=}R]$
	$[S \to \cdot R]$	$[L \to \cdot {*}R]$
	$[L \to \cdot \mathtt{a}]$	$[R \to \cdot L]$
$I_1(S)$:	$[S' \to S\cdot]$	
$I_2(L)$:	$[S \to L \cdot {=}R]$	$[R \to L\cdot]$
$I_3(R)$:	$[S \to R\cdot]$	
$I_4(*)$:	$[L \to * \cdot R]$	$[R \to \cdot L]$
	$[L \to \cdot {*}R]$	$[L \to \cdot \mathtt{a}]$
$I_5(\mathtt{a})$:	$[L \to \mathtt{a}\cdot]$	
$I_6(L{=})$:	$[S \to L{=} \cdot R]$	$[R \to \cdot L]$
	$[L \to \cdot {*}R]$	$[L \to \cdot \mathtt{a}]$
$I_7(*R)$:	$[L \to *R\cdot]$	
$I_8(*L)$:	$[R \to L\cdot]$	
$I_9(L{=}R)$:	$[S \to L{=}R\cdot]$	

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

$I''_0 := I'_0$:	$[S' \to \cdot S, \varepsilon]$	$[S \to \cdot L{=}R, \varepsilon]$
	$[S \to \cdot R, \varepsilon]$	$[L \to \cdot {*}R, {=}/\varepsilon]$
	$[L \to \cdot \mathtt{a}, {=}/\varepsilon]$	$[R \to \cdot L, \varepsilon]$
$I''_1 := I'_1$:	$[S' \to S\cdot, \varepsilon]$	
$I''_2 := I'_2$:	$[S \to L \cdot {=}R, \varepsilon]$	$[R \to L\cdot, \varepsilon]$
$I''_3 := I'_3$:	$[S \to R\cdot, \varepsilon]$	
$I''_4 := I'_4 \cup I'_{11}$:	$[L \to * \cdot R, {=}/\varepsilon]$	$[R \to \cdot L, {=}/\varepsilon]$
	$[L \to \cdot {*}R, {=}/\varepsilon]$	$[L \to \cdot \mathtt{a}, {=}/\varepsilon]$
$I''_5 := I'_5 \cup I'_{12}$:	$[L \to \mathtt{a}\cdot, {=}/\varepsilon]$	
$I''_6 := I'_6$:	$[S \to L{=} \cdot R, \varepsilon]$	$[R \to \cdot L, \varepsilon]$
	$[L \to \cdot {*}R, \varepsilon]$	$[L \to \cdot \mathtt{a}, \varepsilon]$
$I''_7 := I'_7 \cup I'_{13}$:	$[L \to *R\cdot, {=}/\varepsilon]$	
$I''_8 := I'_8 \cup I'_{10}$:	$[R \to L\cdot, {=}/\varepsilon]$	
$I''_9 := I'_9$:	$[S \to L{=}R\cdot, \varepsilon]$	

The $LALR(1)$ Action Function

The $LALR(1)$ action function is defined in analogy to the $LR(1)$ case (Definition 11.14).

Definition 12.6 ($LALR(1)$ action function)

The $LALR(1)$ action function

$$\mathrm{act} : LALR(1)(G) \times \Sigma_\varepsilon \to \{\mathsf{red}\, i \mid i \in [p]\} \cup \{\mathsf{shift}, \mathsf{accept}, \mathsf{error}\}$$

is defined by

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

Definition 12.7 ($LALR(1)$ grammar)

A grammar $G \in CFG_\Sigma$ has the $LALR(1)$ property (notation: $G \in LALR(1)$) if its $LALR(1)$ action function is well defined.

The $LALR(1)$ goto Function

Example 12.8 (cf. Example 12.5)

$G_{LR} \in LALR(1)$

Also the $LR(1)$ goto function (Definition 11.16) carries over to the $LALR(1)$ case. Reason:

Lemma 12.9

Let $G \in CFG_\Sigma$ and $I_1, I_2 \in LR(1)(G)$ such that $I_1 \sim_0 I_2$. Then, for every $Y \in X$, $\mathrm{goto}(I_1, Y) \sim_0 \mathrm{goto}(I_2, Y)$.

Again, act and goto form the $LALR(1)$ parsing table of G.

Example 12.10 (cf. Example 12.5)

$LALR(1)(G_{LR})$	act/goto$\|_{\Sigma_\varepsilon}$ `*`	`=`	`a`	ε	goto$\|_N$ S	L	R
I''_0	shift/I''_4		shift/I''_5		I''_1	I''_2	I''_3
I''_1				accept			
I''_2		shift/I''_6		red 5			
I''_3				red 2			
I''_4	shift/I''_4		shift/I''_5			I''_8	I''_7
I''_5		red 4		red 4			
I''_6	shift/I''_4		shift/I''_5			I''_8	I''_9
I''_7		red 3		red 3			
I''_8		red 5		red 5			
I''_9				red 1			

(empty = error/$\emptyset$)

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

Example 12.11

$G : S' \to S \quad S \to \mathtt{a}A\mathtt{d} \mid \mathtt{b}B\mathtt{d} \mid \mathtt{a}B\mathtt{e} \mid \mathtt{b}A\mathtt{e} \quad A \to \mathtt{c} \quad B \to \mathtt{c}$

$LR(1)(\varepsilon)$:	$[S' \to \cdot S, \varepsilon]$	$[S \to \cdot \mathtt{a}A\mathtt{d}, \varepsilon]$	$[S \to \cdot \mathtt{b}B\mathtt{d}, \varepsilon]$	$[S \to \cdot \mathtt{a}B\mathtt{e}, \varepsilon]$
	$[S \to \cdot \mathtt{b}A\mathtt{e}, \varepsilon]$			
$LR(1)(S)$:	$[S' \to S\cdot, \varepsilon]$			
$LR(1)(\mathtt{a})$:	$[S \to \mathtt{a} \cdot A\mathtt{d}, \varepsilon]$	$[S \to \mathtt{a} \cdot B\mathtt{e}, \varepsilon]$	$[A \to \cdot\mathtt{c}, \mathtt{d}]$	$[B \to \cdot\mathtt{c}, \mathtt{e}]$
$LR(1)(\mathtt{b})$:	$[S \to \mathtt{b} \cdot B\mathtt{d}, \varepsilon]$	$[S \to \mathtt{b} \cdot A\mathtt{e}, \varepsilon]$	$[B \to \cdot\mathtt{c}, \mathtt{d}]$	$[A \to \cdot\mathtt{c}, \mathtt{e}]$
$LR(1)(\mathtt{a}A)$:	$[S \to \mathtt{a}A \cdot \mathtt{d}, \varepsilon]$		$LR(1)(\mathtt{a}B)$:	$[S \to \mathtt{a}B \cdot \mathtt{e}, \varepsilon]$
$LR(1)(\mathtt{ac})$:	$[A \to \mathtt{c}\cdot, \mathtt{d}]$	$[B \to \mathtt{c}\cdot, \mathtt{e}]$		
$LR(1)(\mathtt{b}B)$:	$[S \to \mathtt{b}B \cdot \mathtt{d}, \varepsilon]$		$LR(1)(\mathtt{b}A)$:	$[S \to \mathtt{b}A \cdot \mathtt{e}, \varepsilon]$
$LR(1)(\mathtt{bc})$:	$[B \to \mathtt{c}\cdot, \mathtt{d}]$	$[A \to \mathtt{c}\cdot, \mathtt{e}]$		
$LR(1)(\mathtt{a}A\mathtt{d})$:	$[S \to \mathtt{a}A\mathtt{d}\cdot, \varepsilon]$		$LR(1)(\mathtt{a}B\mathtt{e})$:	$[S \to \mathtt{a}B\mathtt{e}\cdot, \varepsilon]$
$LR(1)(\mathtt{b}B\mathtt{d})$:	$[S \to \mathtt{b}B\mathtt{d}\cdot, \varepsilon]$		$LR(1)(\mathtt{b}A\mathtt{e})$:	$[S \to \mathtt{b}A\mathtt{e}\cdot, \varepsilon]$

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

$LR(1)(\mathtt{ac}) \sim_0 LR(1)(\mathtt{bc})$, but $LR(1)(\mathtt{ac}) \cup LR(1)(\mathtt{bc})$ 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

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)

Outline

Ambiguous Grammars

Reminder (Definition 5.6): 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 12.12

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 $\alpha A v$ be the last common sentence of the two derivations (i.e., $\beta \neq \beta'$):

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

But since $\text{first}_k(v) = \text{first}_k(v)$ for every $v \in \Sigma^*$, Definition 9.8 yields that $\beta = \beta'$. Contradiction □

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

Bottom-Up Parsing of Ambiguous Grammars I

Example 12.13 (Simple arithmetic expressions)

$G : E' \to E \quad E \to E\texttt{+}E \mid E\texttt{*}E \mid \texttt{a}$

Precedence: $\texttt{*} > \texttt{+}$ Associativity: left

(thus: $\texttt{a+a*a+a} := \texttt{(a+(a*a))+a}$)

$LR(0)(G)$:

$I_0 := LR(0)(\varepsilon)$:	$[E' \to \cdot E]$	$[E \to \cdot E\texttt{+}E]$	$[E \to \cdot E\texttt{*}E]$	$[E \to \cdot\texttt{a}]$
$I_1 := LR(0)(E)$:	$[E' \to E\cdot]$	$[E \to E \cdot \texttt{+}E]$	$[E \to E \cdot \texttt{*}E]$	
$I_2 := LR(0)(\texttt{a})$:	$[E \to \texttt{a}\cdot]$			
$I_3 := LR(0)(E\texttt{+})$:	$[E \to E\texttt{+} \cdot E]$	$[E \to \cdot E\texttt{+}E]$	$[E \to \cdot E\texttt{*}E]$	$[E \to \cdot\texttt{a}]$
$I_4 := LR(0)(E\texttt{*})$:	$[E \to E\texttt{*} \cdot E]$	$[E \to \cdot E\texttt{+}E]$	$[E \to \cdot E\texttt{*}E]$	$[E \to \cdot\texttt{a}]$
$I_5 := LR(0)(E\texttt{+}E)$:	$[E \to E\texttt{+}E\cdot]$	$[E \to E \cdot \texttt{+}E]$	$[E \to E \cdot \texttt{*}E]$	
$I_6 := LR(0)(E\texttt{*}E)$:	$[E \to E\texttt{*}E\cdot]$	$[E \to E \cdot \texttt{+}E]$	$[E \to E \cdot \texttt{*}E]$	

Conflicts: I_1: $SLR(1)$-solvable (reduce on ε, shift on $\texttt{+}/\texttt{*}$)
I_5, I_6: not $SLR(1)$-solvable ($\texttt{+}, \texttt{*} \in \mathrm{fo}(E)$)

Solution:

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

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

Bottom-Up Parsing of Ambiguous Grammars II

Example 12.14 ("Dangling `else`")

$G : S' \to S \quad S \to \mathtt{i}S\mathtt{e}S \mid \mathtt{i}S \mid \mathtt{a}$

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

$LR(0)(G)$:

$I_0 := LR(0)(\varepsilon)$: $[S' \to \cdot S]$ $[S \to \cdot\mathtt{i}S\mathtt{e}S]$ $[S \to \cdot\mathtt{i}S]$ $[S \to \cdot\mathtt{a}]$

$I_1 := LR(0)(S)$: $[S' \to S\cdot]$

$I_2 := LR(0)(\mathtt{i})$: $[S \to \mathtt{i}\cdot S\mathtt{e}S]$ $[S \to \mathtt{i}\cdot S]$ $[S \to \cdot\mathtt{i}S\mathtt{e}S]$ $[S \to \cdot\mathtt{i}S]$ $[S \to \cdot\mathtt{a}]$

$I_3 := LR(0)(\mathtt{a})$: $[S \to \mathtt{a}\cdot]$

$I_4 := LR(0)(\mathtt{i}S)$: $[S \to \mathtt{i}S\cdot\mathtt{e}S]$ $[S \to \mathtt{i}S\cdot]$

$I_5 := LR(0)(\mathtt{i}S\mathtt{e})$: $[S \to \mathtt{i}S\mathtt{e}\cdot S]$ $[S \to \cdot\mathtt{i}S\mathtt{e}S]$ $[S \to \cdot\mathtt{i}S]$ $[S \to \cdot\mathtt{a}]$

$I_6 := LR(0)(\mathtt{i}S\mathtt{e}S)$: $[S \to \mathtt{i}S\mathtt{e}S\cdot]$

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

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

Outline

The yacc Tool

Usage of `yacc` ("yet another compiler compiler"):

$$\begin{array}{ccccc} \texttt{spec.y} & \xrightarrow{\texttt{yacc}} & \texttt{y.tab.c} \quad \texttt{lex.yy.c} & \xleftarrow{\texttt{[f]lex}} & \texttt{spec.l} \\ \texttt{yacc}\text{ specification} & & \text{Parser source} \quad \text{Scanner source} & & \texttt{[f]lex}\text{ specification} \\ & & \downarrow \texttt{cc} \downarrow & & \\ & & \texttt{a.out} & & \\ & & \text{Executable } LALR(1) \text{ parser} & & \end{array}$$

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

Declarations (optional)
%%
Rules
%%
Auxiliary procedures (optional)

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

- $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ represented as

$$\begin{array}{lcll} A & : & \alpha_1 & \{\mathit{Action_1}\} \\ & | & \alpha_2 & \{\mathit{Action_2}\} \\ & \vdots & & \\ & | & \alpha_n & \{\mathit{Action_n}\}; \end{array}$$

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

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

Example: Simple Desk Calculator I

```
%{/* SLR(1) grammar for arithmetic expressions (Example 11.1) */
  #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. 12.13) */
  #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

shift/reduce: prefer shift

- resolves dangling-else ambiguity (Example 12.14) correctly
- also adequate for $*$ after sum (Example 12.13) and for right-associative binary operators
- not appropriate for left-associative binary operators ($\Longrightarrow$ reduce; see Example 12.13)

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:

$$\texttt{\%[left|right]}\ \mathit{Operators}_1$$
$$\vdots$$
$$\texttt{\%[left|right]}\ \mathit{Operators}_n$$

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

Example 12.15

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

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

Precedences and Associativities in yacc II

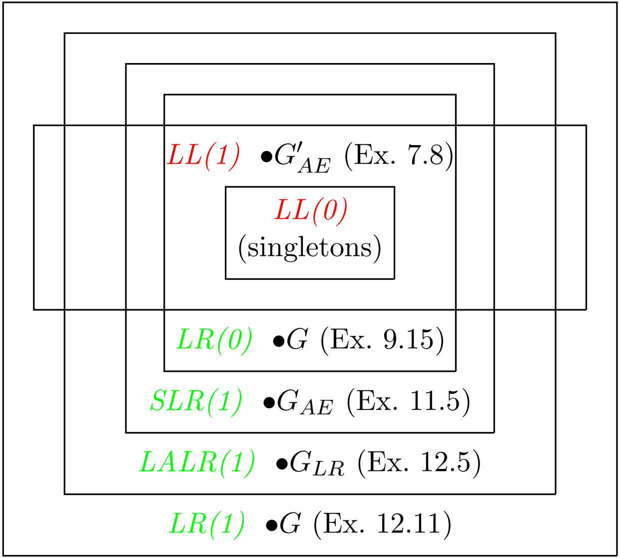
```
%{/* Ambiguous grammar for arithmetic expressions
     with precedences and associativities */
  #include <stdio.h>
  #include <ctype.h>
%}
%token DIGIT
%left '+'
%left '*'
%%
line    : expr '\n'  { printf("%d\n", $1); };
expr    : expr '+' expr     { $$ = $1 + $3; }
        | expr '*' expr     { $$ = $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 nonambig.y
> cc y.tab.c -ly
> a.out
2*3+5
11
> a.out
2+3*5
17
```

Outline

Overview of Grammar Classes

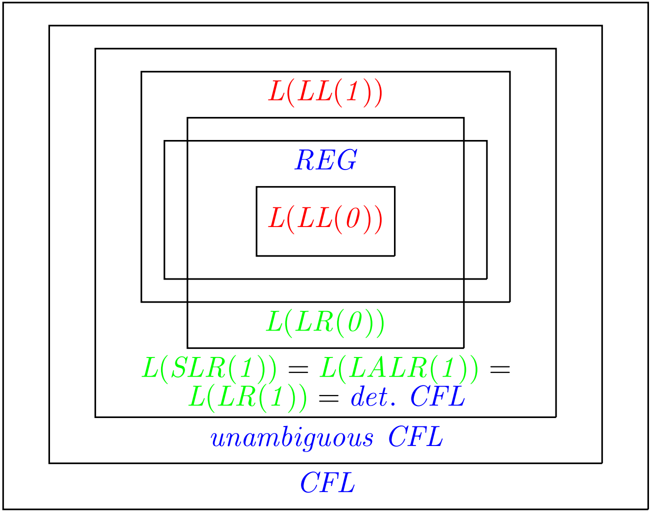


Moreover:

- $LL(k) \subsetneq LL(k+1)$ 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 $k \geq 1$