Compiler Construction

Lecture 11: Syntax Analysis VIII
($LALR(1)$ Parsing & Practical Issues)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://moves.rwth-aachen.de/teaching/ss-14/cc14/

Summer Semester 2014

Outline

$LR(1)$ 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.
- $LR(1)(G) := \{LR(1)(\gamma) \mid \gamma \in X^*\}$.

The $LR(1)$ Action Function

Definition ($LR(1)$ action function)

The $LR(1)$ action function

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

is defined by

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

Corollary

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

Outline

$LALR(1)$ Parsing

- **Motivation:** resolving conflicts using $LR(1)$ too expensive
- Example 10.11/10.17: $|LR(0)(G_{LR})| = 11$, $|LR(1)(G_{LR})| = 15$
- Empirical evaluations:
  - A. Johnstone, E. Scott: *Generalised Reduction Modified LR Parsing for Domain Specific Language Prototyping*, HICSS'02, IEEE, 2002
  - X. Chen, D. Pager: *Full LR(1) Parser Generator Hyacc and Study on the Performance of LR(1) Algorithms*, C3S2E'11, ACM, 2011

Grammar	$\|LR(0)(G)\|$	$\|LR(1)(G)\|$
Pascal	368	1395
Ansi-C	381	1788
C++	1236	9723

$LR(0)$ Equivalence I

Observation: potential redundancy by containment of $LR(0)$ sets in $LR(1)$ sets (cf. Corollary 10.13)

Definition 11.1 ($LR(0)$ 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 II

Example 11.2 (cf. Example 10.11/10.17)

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

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

$I_0(\varepsilon)$: $[S' \to \cdot S]$ $[S \to \cdot L{=}R]$ $[S \to \cdot R]$ $[L \to \cdot *R]$ $[L \to \cdot \mathrm{a}]$ $[R \to \cdot L]$

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

$I_2(L)$: $[S \to L \cdot {=}R]$ $[R \to L\cdot]$

$I_3(R)$: $[S \to R\cdot]$

$I_4(*)$: $[L \to * \cdot R]$ $[R \to \cdot L]$ $[L \to \cdot *R]$ $[L \to \cdot \mathrm{a}]$

$I_5(\mathrm{a})$: $[L \to \mathrm{a}\cdot]$

$I_6(L{=})$: $[S \to L{=} \cdot R]$ $[R \to \cdot L]$ $[L \to \cdot *R]$ $[L \to \cdot \mathrm{a}]$

$I_7(*R)$: $[L \to *R\cdot]$

$I_8(*L)$: $[R \to L\cdot]$

$I_9(L{=}R)$: $[S \to L{=}R\cdot]$

$\Longrightarrow$ $I'_4 \sim_0 I'_{11}$, $I'_5 \sim_0 I'_{12}$, $I'_7 \sim_0 I'_{13}$, $I'_8 \sim_0 I'_{10}$

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

$I'_0(\varepsilon)$: $[S' \to \cdot S, \varepsilon]$ $[S \to \cdot L{=}R, \varepsilon]$ $[S \to \cdot R, \varepsilon]$ $[L \to \cdot *R, =]$ $[L \to \cdot \mathrm{a}, =]$ $[R \to \cdot L, \varepsilon]$ $[L \to \cdot *R, \varepsilon]$ $[L \to \cdot \mathrm{a}, \varepsilon]$

$I'_1(S)$: $[S' \to S\cdot, \varepsilon]$

$I'_2(L)$: $[S \to L \cdot {=}R, \varepsilon]$ $[R \to L\cdot, \varepsilon]$

$I'_3(R)$: $[S \to R\cdot, \varepsilon]$

$I'_4(*)$: $[L \to * \cdot R, =]$ $[L \to * \cdot R, \varepsilon]$ $[R \to \cdot L, =]$ $[R \to \cdot L, \varepsilon]$ $[L \to \cdot *R, =]$ $[L \to \cdot \mathrm{a}, =]$ $[L \to \cdot *R, \varepsilon]$ $[L \to \cdot \mathrm{a}, \varepsilon]$

$I'_5(\mathrm{a})$: $[L \to \mathrm{a}\cdot, =]$ $[L \to \mathrm{a}\cdot, \varepsilon]$

$I'_6(L{=})$: $[S \to L{=} \cdot R, \varepsilon]$ $[R \to \cdot L, \varepsilon]$ $[L \to \cdot *R, \varepsilon]$ $[L \to \cdot \mathrm{a}, \varepsilon]$

$I'_7(*R)$: $[L \to *R\cdot, =]$ $[L \to *R\cdot, \varepsilon]$

$I'_8(*L)$: $[R \to L\cdot, =]$ $[R \to L\cdot, \varepsilon]$

$I'_9(L{=}R)$: $[S \to L{=}R\cdot, \varepsilon]$

$I'_{10}(L{=}L)$: $[R \to L\cdot, \varepsilon]$

$I'_{11}(L{=}*)$: $[L \to * \cdot R, \varepsilon]$ $[R \to \cdot L, \varepsilon]$ $[L \to \cdot *R, \varepsilon]$ $[L \to \cdot \mathrm{a}, \varepsilon]$

$I'_{12}(L{=}\mathrm{a})$: $[L \to \mathrm{a}\cdot, \varepsilon]$

$I'_{13}(L{=}*R)$: $[L \to *R\cdot, \varepsilon]$

$LALR(1)$ Sets I

Corollary 11.3

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 11.4 ($LALR(1)$ 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 11.3, $|LALR(1)(G)| = |LR(0)(G)|$
(but $LALR(1)$ sets provide additional lookahead information)

$LALR(1)$ Sets II

Example 11.5 (cf. Example 11.2)

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

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

$I_0(\varepsilon)$: $[S' \to \cdot S]$ $[S \to \cdot L{=}R]$ $[S \to \cdot R]$ $[L \to \cdot {*}R]$ $[L \to \cdot \mathrm{a}]$ $[R \to \cdot L]$

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

$I_2(L)$: $[S \to L \cdot {=}R]$ $[R \to L\cdot]$

$I_3(R)$: $[S \to R\cdot]$

$I_4({*})$: $[L \to {*} \cdot R]$ $[R \to \cdot L]$ $[L \to \cdot {*}R]$ $[L \to \cdot \mathrm{a}]$

$I_5(\mathrm{a})$: $[L \to \mathrm{a}\cdot]$

$I_6(L{=})$: $[S \to L{=} \cdot R]$ $[R \to \cdot L]$ $[L \to \cdot {*}R]$ $[L \to \cdot \mathrm{a}]$

$I_7({*}R)$: $[L \to {*}R\cdot]$

$I_8({*}L)$: $[R \to L\cdot]$

$I_9(L{=}R)$: $[S \to L{=}R\cdot]$

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

$I_0'' := I_0'$: $[S' \to \cdot S, \varepsilon]$ $[S \to \cdot L{=}R, \varepsilon]$ $[S \to \cdot R, \varepsilon]$ $[L \to \cdot {*}R, {=}/\varepsilon]$ $[L \to \cdot \mathrm{a}, {=}/\varepsilon]$ $[R \to \cdot L, \varepsilon]$

$I_1'' := I_1'$: $[S' \to S\cdot, \varepsilon]$

$I_2'' := I_2'$: $[S \to L \cdot {=}R, \varepsilon]$ $[R \to L\cdot, \varepsilon]$

$I_3'' := I_3'$: $[S \to R\cdot, \varepsilon]$

$I_4'' := I_4' \cup I_{11}'$: $[L \to {*} \cdot R, {=}/\varepsilon]$ $[R \to \cdot L, {=}/\varepsilon]$ $[L \to \cdot {*}R, {=}/\varepsilon]$ $[L \to \cdot \mathrm{a}, {=}/\varepsilon]$

$I_5'' := I_5' \cup I_{12}'$: $[L \to \mathrm{a}\cdot, {=}/\varepsilon]$

$I_6'' := I_6'$: $[S \to L{=} \cdot R, \varepsilon]$ $[R \to \cdot L, \varepsilon]$ $[L \to \cdot {*}R, \varepsilon]$ $[L \to \cdot \mathrm{a}, \varepsilon]$

$I_7'' := I_7' \cup I_{13}'$: $[L \to {*}R\cdot, {=}/\varepsilon]$

$I_8'' := I_8' \cup I_{10}'$: $[R \to L\cdot, {=}/\varepsilon]$

$I_9'' := I_9'$: $[S \to L{=}R\cdot, \varepsilon]$

The $LALR(1)$ Action Function

The $LALR(1)$ action function is defined in analogy to the $LR(1)$ case (Definition 10.18).

Definition 11.6 ($LALR(1)$ action function)

The $LALR(1)$ action function

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

is defined by

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

Definition 11.7 ($LALR(1)$ 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 11.8 (cf. Example 11.5)

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

Also the $LR(1)$ goto function (Definition 10.20) carries over to the $LALR(1)$ case. Reason:

Lemma 11.9

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 $Y \in X$, $\text{goto}(I_1, Y) \sim_0 \text{goto}(I_2, Y)$.

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

The $LALR(1)$ Parsing Table

Example 11.10 (cf. Example 11.5)

$LALR(1)(G_{LR})$	$\text{act/goto}\vert_\Sigma$				$\text{goto}\vert_N$		
	$*$	$=$	a	ε	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/$\emptyset$)

$LALR(1)$ Conflicts

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

Example 11.11

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

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

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

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

$\implies$ $G \notin LALR(1)$

Outline

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 11.12

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 $\mathrm{first}_k(v) = \mathrm{first}_k(v)$ for every $v \in \Sigma^*$, Definition 9.3 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 I

Example 11.13 (Simple arithmetic expressions)

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

Precedence: $* > +$ Associativity: left

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

$LR(0)(G)$:

$I_0 := LR(0)(\varepsilon) : \quad [E' \to \cdot E] \quad [E \to \cdot E{+}E] \quad [E \to \cdot E{*}E] \quad [E \to \cdot \mathtt{a}]$

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

$I_2 := LR(0)(\mathtt{a}) : \quad [E \to \mathtt{a}\cdot]$

$I_3 := LR(0)(E{+}) : \quad [E \to E{+} \cdot E] \quad [E \to \cdot E{+}E] \quad [E \to \cdot E{*}E] \quad [E \to \cdot \mathtt{a}]$

$I_4 := LR(0)(E{*}) : \quad [E \to E{*} \cdot E] \quad [E \to \cdot E{+}E] \quad [E \to \cdot E{*}E] \quad [E \to \cdot \mathtt{a}]$

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

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

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

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

Solution:

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

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

Bottom-Up Parsing of Ambiguous Grammars II

Example 11.14 ("Dangling `else`")

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

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

$LR(0)(G)$:

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

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

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

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

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

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

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

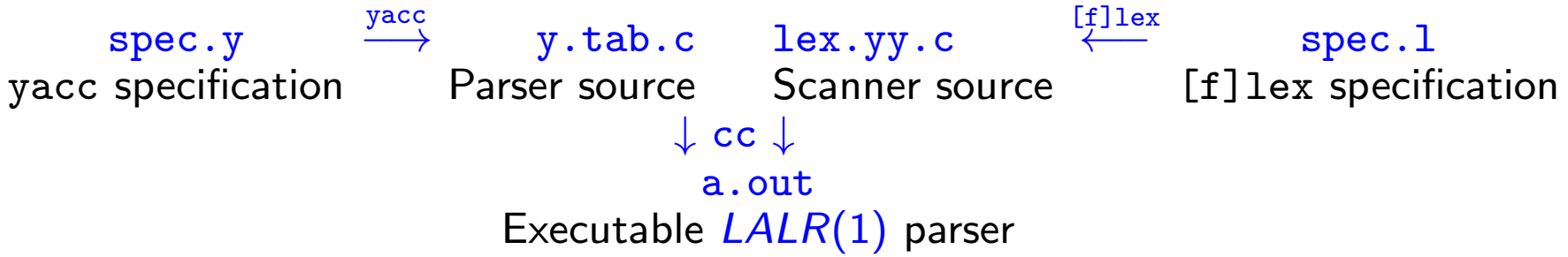
Conflict in I_4: $\mathtt{e} \in \mathrm{fo}(S) \implies$ not $SLR(1)$-solvable

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

Outline

The yacc 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)

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

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

- 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 10.5) */
  #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. 11.13) */
  #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 11.14) correctly
- also adequate for strong following weak operator (* after +; Example 11.13) and for right-associative operators
- not appropriate for weak following strong operator and for left-associative binary operators
  ($\Longrightarrow$ reduce; see Example 11.13)

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:

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

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

Example 11.15

```
%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

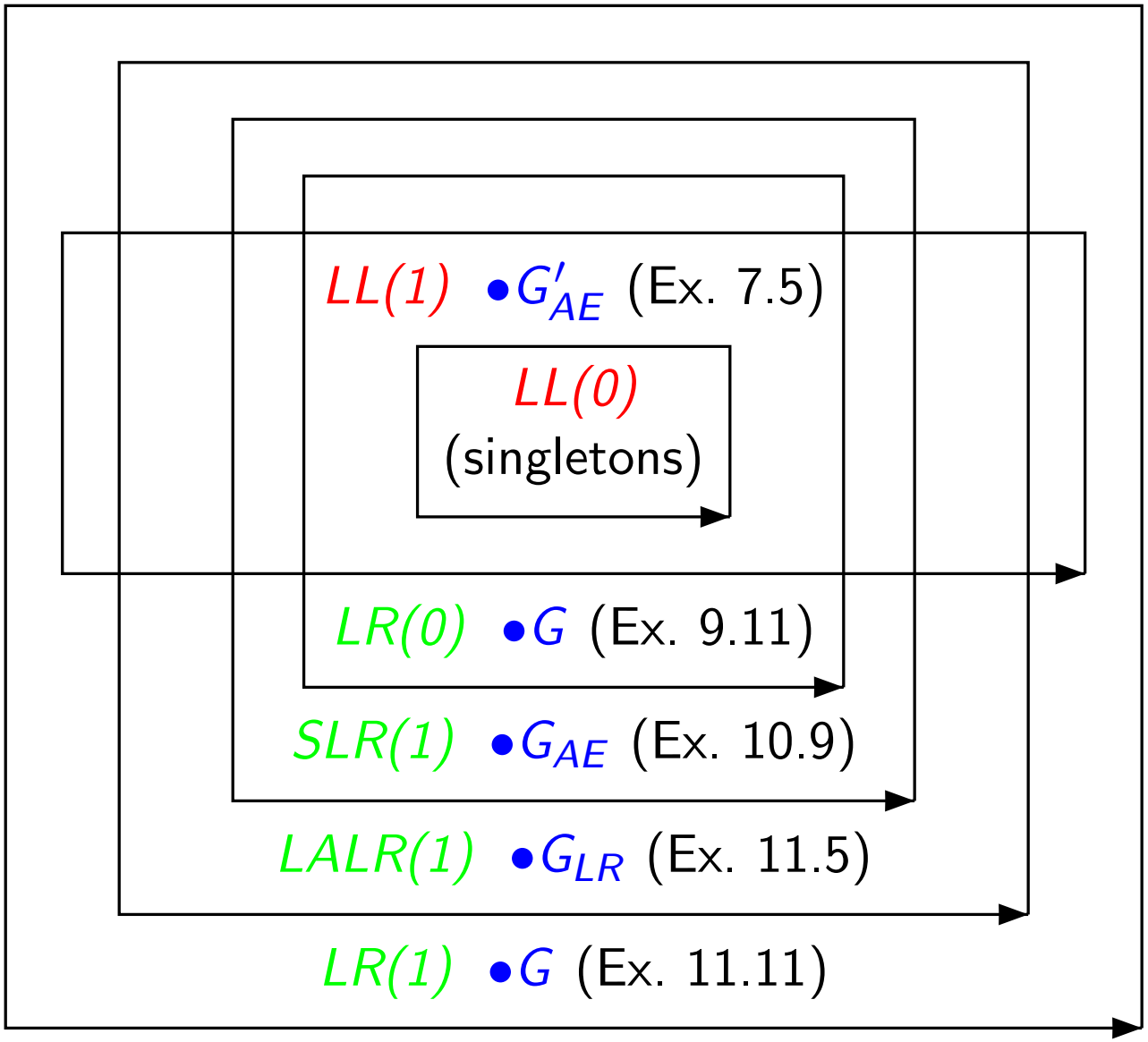
```
%{/* Ambiguous grammar for arithmetic expressions
     with precedences and associativities */
  #include <stdio.h>
  #include <ctype.h>
%}
%token DIGIT
%left '+'
%left '*'
%%
line   : expr '\n'  { printf("%d\n", $1); };
expr   : expr '+' expr    { $$ = $1 + $3; }
       | expr '*' expr    { $$ =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

Outline

Overview of Grammar Classes

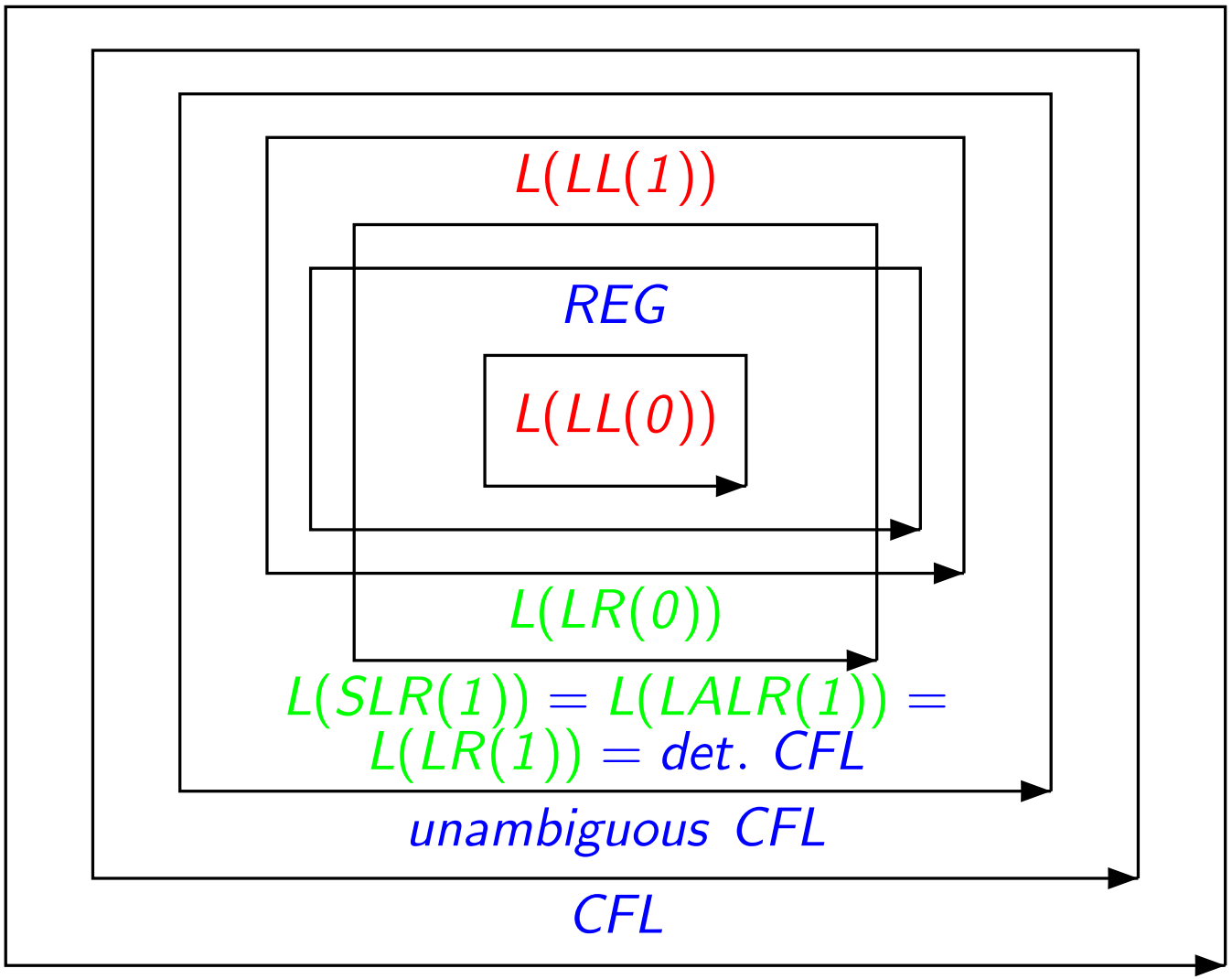


Moreover:

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

Outline

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

Generality : LALR wins
- "almost" $LL(1) \subseteq LALR(1)$ (only pathological counterexamples)
- LL requires elimination of left recursion and left factorization

Semantic actions : (see semantic analysis) LL wins
- actions can be placed anywhere in LL parsers without causing conflicts
- in LALR: implicit ε-productions