

Compiler Construction

Lecture 12: 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://www-i2.informatik.rwth-aachen.de/i2/cc12/>

Summer Semester 2012

- 1 Repetition: $LR(1)$ Parsing
- 2 $LALR(1)$ Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc and bison
- 5 LL and LR Parsing in Practice

Observation: not every element of $\text{fo}(A)$ can follow every occurrence of A
 $\Rightarrow$ 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 $S' \Rightarrow_r^* \alpha A w \Rightarrow_r \alpha \beta_1 \beta_2 w$, 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 $[A \rightarrow \beta_1 \cdot \beta_2, \epsilon]$ 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

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

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.

- 1 Repetition: $LR(1)$ Parsing
- 2 $LALR(1)$ Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc and bison
- 5 LL and LR Parsing in Practice

- **Motivation:** resolving conflicts using $LR(1)$ too expensive
- Example 11.7/11.13: $|LR(0)(G_{LR})| = 11$, $|LR(1)(G_{LR})| = 15$
- 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

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

Definition 12.1 ($LR(0)$ 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 $lr_0(I_1) = lr_0(I_2)$.

Example 12.2 (cf. Example 11.7/11.13)

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

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

$I_0(\epsilon)$: $[S' \rightarrow \cdot S]$ $[S \rightarrow \cdot L=R]$
 $[S \rightarrow \cdot R]$ $[L \rightarrow \cdot *R]$
 $[L \rightarrow \cdot a]$ $[R \rightarrow \cdot L]$

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

$I_2(L)$: $[S \rightarrow L \cdot =R]$ $[R \rightarrow L \cdot]$

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

$I_4(*)$: $[L \rightarrow * \cdot R]$ $[R \rightarrow \cdot L]$
 $[L \rightarrow \cdot *R]$ $[L \rightarrow \cdot a]$

$I_5(a)$: $[L \rightarrow a \cdot]$

$I_6(L=)$: $[S \rightarrow L= \cdot R]$ $[R \rightarrow \cdot L]$
 $[L \rightarrow \cdot *R]$ $[L \rightarrow \cdot a]$

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

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

$I_9(L=R)$: $[S \rightarrow L=R \cdot]$

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

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

$I'_0(\epsilon)$: $[S' \rightarrow \cdot S, \epsilon]$ $[S \rightarrow \cdot L=R, \epsilon]$
 $[S \rightarrow \cdot R, \epsilon]$ $[L \rightarrow \cdot *R, =]$
 $[L \rightarrow \cdot a, =]$ $[R \rightarrow \cdot L, \epsilon]$
 $[L \rightarrow \cdot *R, \epsilon]$ $[L \rightarrow \cdot a, \epsilon]$

$I'_1(S)$: $[S' \rightarrow S \cdot, \epsilon]$

$I'_2(L)$: $[S \rightarrow L \cdot =R, \epsilon]$ $[R \rightarrow L \cdot, \epsilon]$

$I'_3(R)$: $[S \rightarrow R \cdot, \epsilon]$

$I'_4(*)$: $[L \rightarrow * \cdot R, =]$ $[L \rightarrow * \cdot R, \epsilon]$
 $[R \rightarrow \cdot L, =]$ $[R \rightarrow \cdot L, \epsilon]$
 $[L \rightarrow \cdot *R, =]$ $[L \rightarrow \cdot a, =]$
 $[L \rightarrow \cdot *R, \epsilon]$ $[L \rightarrow \cdot a, \epsilon]$

$I'_5(a)$: $[L \rightarrow a \cdot, =]$ $[L \rightarrow a \cdot, \epsilon]$

$I'_6(L=)$: $[S \rightarrow L= \cdot R, \epsilon]$ $[R \rightarrow \cdot L, \epsilon]$
 $[L \rightarrow \cdot *R, \epsilon]$ $[L \rightarrow \cdot a, \epsilon]$

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

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

$I'_9(L=R)$: $[S \rightarrow L=R \cdot, \epsilon]$

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

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

$I'_{12}(L=a)$: $[L \rightarrow a \cdot, \epsilon]$

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

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)

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

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

$I_0(\varepsilon) :$ $[S' \rightarrow \cdot S] \quad [S \rightarrow \cdot L=R]$
 $[S \rightarrow \cdot R] \quad [L \rightarrow \cdot *R]$
 $[L \rightarrow \cdot a] \quad [R \rightarrow \cdot L]$

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

$I_2(L) :$ $[S \rightarrow L \cdot =R] \quad [R \rightarrow L \cdot]$

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

$I_4(*) :$ $[L \rightarrow * \cdot R] \quad [R \rightarrow \cdot L]$
 $[L \rightarrow \cdot *R] \quad [L \rightarrow \cdot a]$

$I_5(a) :$ $[L \rightarrow a \cdot]$

$I_6(L=) :$ $[S \rightarrow L = \cdot R] \quad [R \rightarrow \cdot L]$
 $[L \rightarrow \cdot *R] \quad [L \rightarrow \cdot a]$

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

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

$I_9(L=R) :$ $[S \rightarrow L=R \cdot]$

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

$I_0'' := I_0' :$ $[S' \rightarrow \cdot S, \varepsilon] \quad [S \rightarrow \cdot L=R, \varepsilon]$
 $[S \rightarrow \cdot R, \varepsilon] \quad [L \rightarrow \cdot *R, \neq/\varepsilon]$
 $[L \rightarrow \cdot a, \neq/\varepsilon] \quad [R \rightarrow \cdot L, \varepsilon]$

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

$I_2'' := I_2' :$ $[S \rightarrow L \cdot =R, \varepsilon] \quad [R \rightarrow L \cdot, \varepsilon]$

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

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

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

$I_6'' := I_6' :$ $[S \rightarrow L = \cdot R, \varepsilon] \quad [R \rightarrow \cdot L, \varepsilon]$
 $[L \rightarrow \cdot *R, \varepsilon] \quad [L \rightarrow \cdot a, \varepsilon]$

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

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

$I_9'' := I_9' :$ $[S \rightarrow L=R \cdot, \varepsilon]$

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

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

is defined by

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

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 $l_1, l_2 \in LR(1)(G)$ such that $l_1 \sim_0 l_2$. Then, for every $Y \in X$, $\text{goto}(l_1, Y) \sim_0 \text{goto}(l_2, Y)$.

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

The LALR(1) Parsing Table

Example 12.10 (cf. Example 12.5)

LALR(1)(G_{LR})	act/goto Σ				goto 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 12.11

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

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

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

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

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

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

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

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

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

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

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

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

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

$\implies G \notin LALR(1)$

- 1 Repetition: $LR(1)$ Parsing
- 2 $LALR(1)$ Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc and bison
- 5 LL and LR Parsing in Practice

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 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 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 9.4 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)

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

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 12.14 (“Dangling else”)

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

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

$LR(0)(G)$:

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

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

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

$[S' \rightarrow S \cdot]$

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

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

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

$[S \rightarrow a \cdot]$

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

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

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

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

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

$[S \rightarrow iSeS \cdot]$

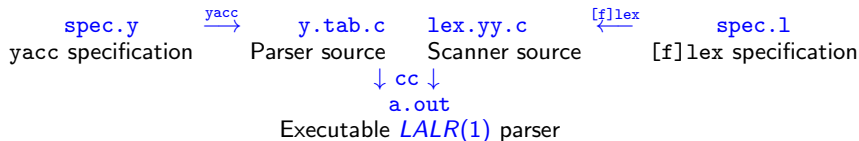
Conflict in I_4 : $e \in \text{fo}(S) \implies$ not $SLR(1)$ -solvable

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

- 1 Repetition: $LR(1)$ Parsing
- 2 $LALR(1)$ Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc and bison
- 5 LL and LR Parsing in Practice

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)

- 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

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

- 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 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 strong following weak operator (***** after **+**; Example 12.13) and for right-associative operators
- not appropriate for weak following strong operator and for left-associative binary operators
($\Rightarrow$ 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:

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

- 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 ambig-prio.y  
> cc y.tab.c -ly  
> a.out  
2*3+5  
11  
> a.out  
2+3*5  
17
```

- 1 Repetition: $LR(1)$ Parsing
- 2 $LALR(1)$ Parsing
- 3 Bottom-Up Parsing of Ambiguous Grammars
- 4 Generating Parsers Using yacc and bison
- 5 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

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