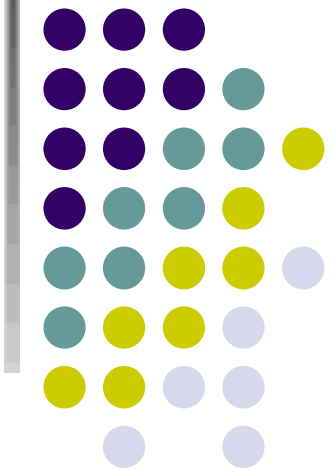
TC3048 Compiler Design

Syntax Analysis

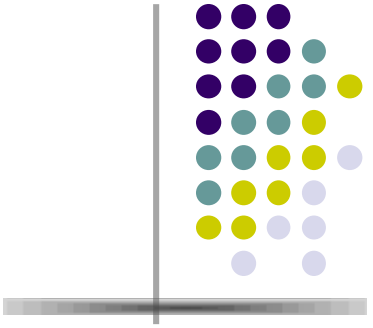
Parser Generators



Elda Quiroga (equiroga@itesm.mx)

Hector Ceballos (ceballos@itesm.mx)

Derivation on LR Parsers



$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid \mathbf{id}$

Parse:

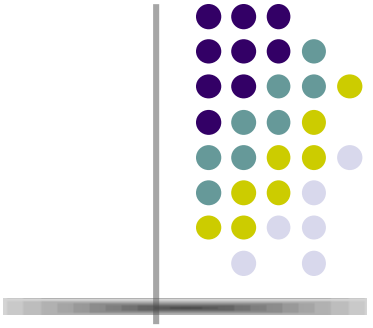
id * id

Stack	Input	Action
$	$\mathbf{id}_1 * \mathbf{id}_2$ $	*shift*
$ $\mathbf{id}_1$	* $\mathbf{id}_2$ $	*reduce by* $F \rightarrow \mathbf{id}$
$ F	* $\mathbf{id}_2$ $	*reduce by* $T \rightarrow F$
$ T	* $\mathbf{id}_2$ $	*shift*
$ T *	$\mathbf{id}_2$ $	*shift*
$ T * $\mathbf{id}_2$	$	*reduce by* $F \rightarrow \mathbf{id}$
$ T * F	$	*reduce by* $T \rightarrow T * F$
$ T	$	*reduce by* $E \rightarrow T$
$ E	$	accept

RightMost Derivation:

$$E \Rightarrow \underline{T} \Rightarrow T * \underline{F} \Rightarrow \underline{T} * \mathbf{id}_2 \Rightarrow \underline{F} * \mathbf{id}_2 \Rightarrow \mathbf{id}_1 * \mathbf{id}_2$$

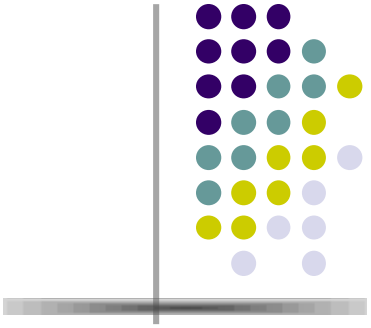
LR Parser conflicts



- *When, knowing the entire stack contents and the next input symbol:*
  - ***Shift/Reduce**: cannot decide to shift or to reduce.*
  - ***Reduce/Reduce**: cannot decide which of several reductions to make.*

- *Non-LR grammars*
  - *Not in the LR(1) class of grammars.*

Shift/Reduce Conflict



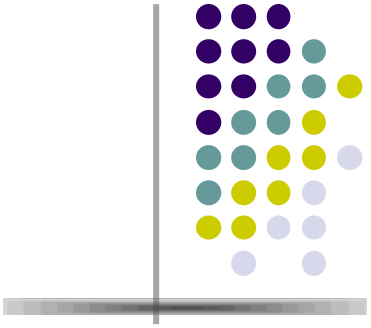
- *Consider the grammar:*
  - *stmt* → **if** *expr* **then** *stmt*
  - | **if** *expr* **then** *stmt* **else** *stmt*
  - | ***other***

reduce

shift

- *And the following configuration:*
  - *STACK* *INPUT*
  - … **if** *expr* **then** *stmt* **else** … $

Reduce/Reduce conflict



- *Consider the grammar:*

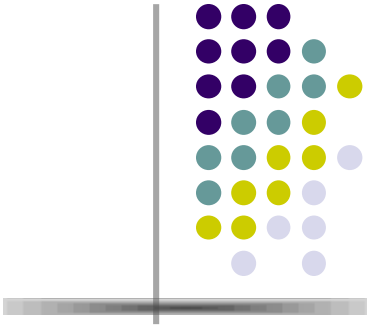
(1)	*stmt*	→	**id** (*parameter_list*)
(2)	*stmt*	→	*expr* := *expr*
(3)	*parameter_list*	→	*parameter_list* , *parameter*
(4)	*parameter_list*	→	*parameter*
(5)	*parameter*	→	**id**
(6)	*expr*	→	**id** (*expr_list*)
(7)	*expr*	→	**id**
(8)	*expr_list*	→	*expr_list* , *expr*
(9)	*expr_list*	→	*expr*

p is a procedure

p is an array

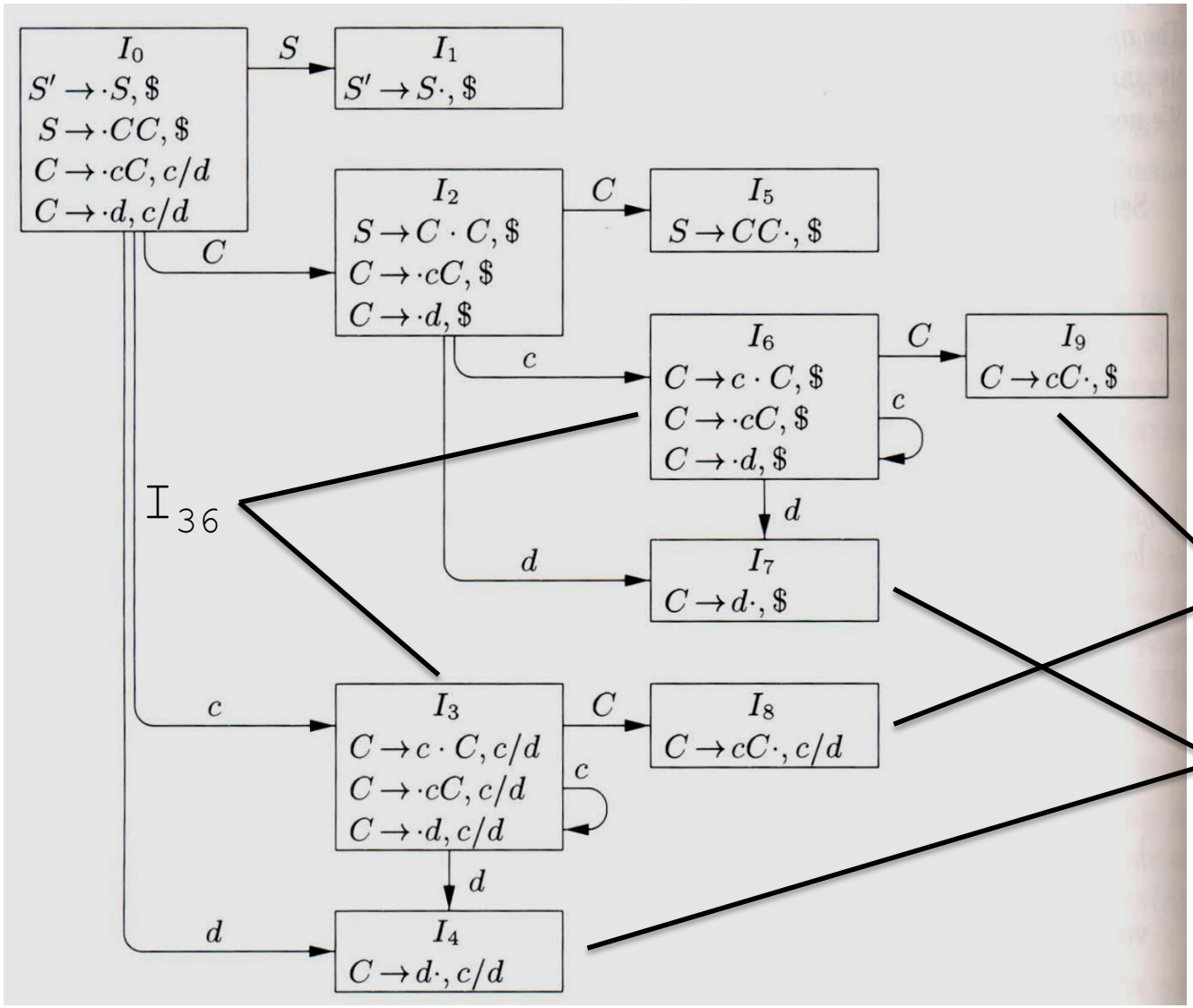
- *When processing* `p(i,j)`*:*
  - *STACK* *INPUT*
  - … **id** (**id** , **id**) …

LALR Parsers



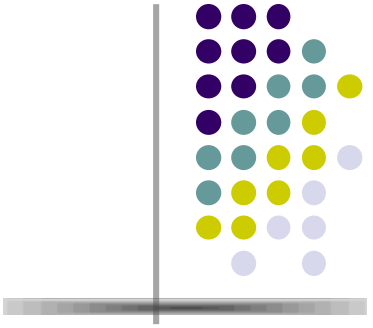
- *Based on the construction of the set of Canonical LR(1) items.*
  - [$A \rightarrow \alpha\cdot$, a] where a $\in$ FOLLOW(A)
- *Groups items with similar core* $A \rightarrow \alpha\cdot$
- LALR parsing tables are much smaller than Canonical parsing tables.

Canonical LR(1) GOTO Graph



$S' \to S$
$S \to C\ C$
$C \to c\ C \mid d$

Parsing tables: Canonical vs LALR



Canonical

STATE	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

LALR

STATE	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Using Ambiguous Grammars

$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$

I_0: $E' \rightarrow \cdot E$
$E \rightarrow \cdot E + E$
$E \rightarrow \cdot E * E$
$E \rightarrow \cdot (E)$
$E \rightarrow \cdot \mathbf{id}$

I_1: $E' \rightarrow E\cdot$
$E \rightarrow E\cdot + E$
$E \rightarrow E\cdot * E$

I_2: $E \rightarrow (\cdot E)$
$E \rightarrow \cdot E + E$
$E \rightarrow \cdot E * E$
$E \rightarrow \cdot (E)$
$E \rightarrow \cdot \mathbf{id}$

I_3: $E \rightarrow \mathbf{id}\cdot$

I_4: $E \rightarrow E + \cdot E$
$E \rightarrow \cdot E + E$
$E \rightarrow \cdot E * E$
$E \rightarrow \cdot (E)$
$E \rightarrow \cdot \mathbf{id}$

I_5: $E \rightarrow E * \cdot E$
$E \rightarrow \cdot E + E$
$E \rightarrow \cdot E * E$
$E \rightarrow \cdot (E)$
$E \rightarrow \cdot \mathbf{id}$

I_6: $E \rightarrow (E\cdot)$
$E \rightarrow E\cdot + E$
$E \rightarrow E\cdot * E$

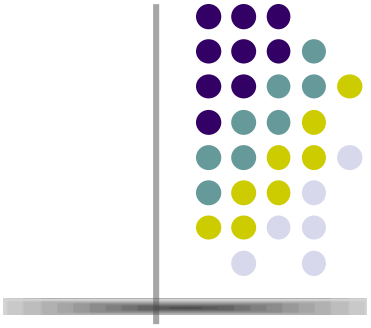
I_7: $E \rightarrow E + E\cdot$
$E \rightarrow E\cdot + E$
$E \rightarrow E\cdot * E$

I_8: $E \rightarrow E * E\cdot$
$E \rightarrow E\cdot + E$
$E \rightarrow E\cdot * E$

I_9: $E \rightarrow (E)\cdot$

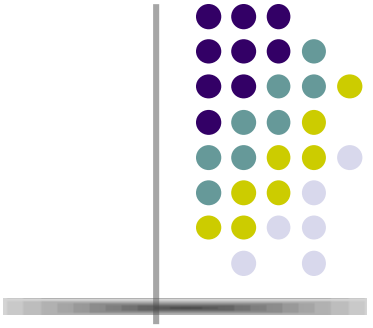
- *When processing* **id + id * id***:*
  - *STACK* *INPUT*
  - 0*E*1+4*E*7 * **id** $
- Conflict between:
  - Reduce $E \rightarrow E + E$
  - Shift *

Resolving with precedence



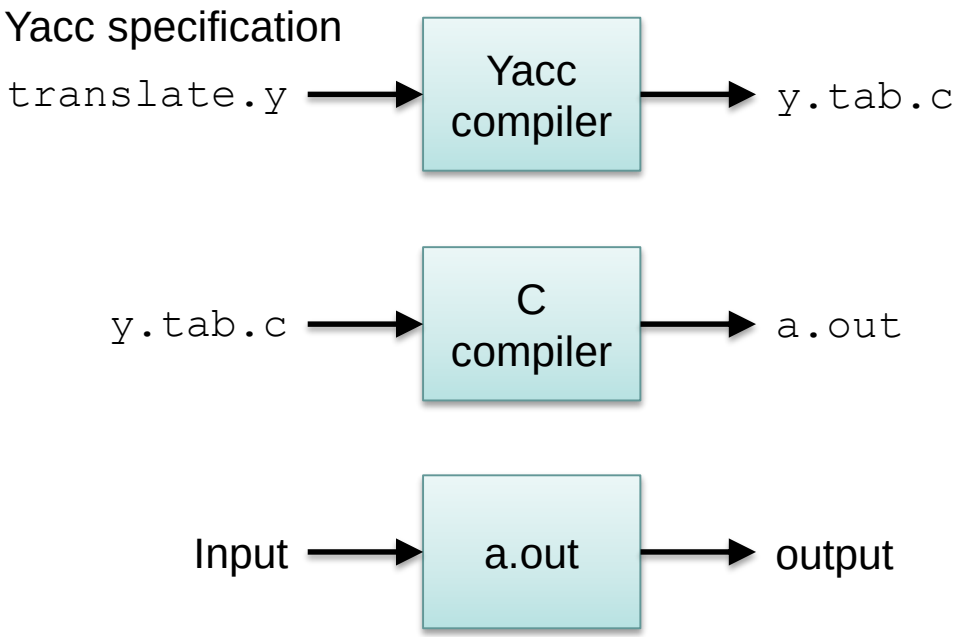
- *If * takes precedence over +*
  - *Choose: Shift **
  - *Continue shifting until have* **id** + **id * id** *in the stack*
  - *Reduce $E \rightarrow E * E$*
  - *Now the stack has:* **id** + *E*
- *If + takes precedence over **
  - *Choose: Reduce $E \rightarrow E + E$*
  - *Continue shifting until have E * **id** in the stack.*

Resolving with associativity



- *When processing* **id** + **id** + **id***:*
  - *STACK* *INPUT*
  - 0*E*1+4*E*7 + **id** $
- *Another Shift/Reduce conflict arises…*
- *If + is* ***left associative****:*
  - *Choose: Reduce by* $E \rightarrow E + E$
  - *In stack we have: E*
  - *Continue shifting until have: E* + **id**
  - *Reduce again by* $E \rightarrow E + E$

The Parser Generator Yacc



translate.y

```
Declarations
%%
Translation rules
%%
Supporting C functions
```

A Simple Calculator

```
%{
#include <ctype.h>
%}

%token DIGIT

%%
line    : expr '\n'          { printf("%d\n", $1); }
        ;
expr    : expr '+' term      { $$ = $1 + $3; }
        | term
        ;
term    : term '*' factor    { $$ = $1 * $3; }
        | factor
        ;
factor  : '(' expr ')'       { $$ = $2; }
        | DIGIT
        ;
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

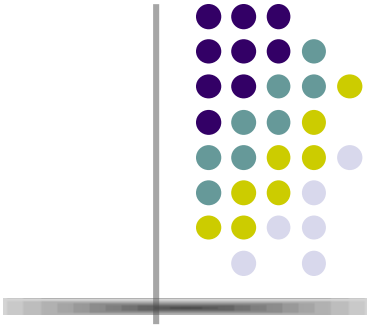
Declarations

Translation rules

<head> : <body>$_1$ { <Sem. action>$_1$ }
| ...
| <body>$_n$ { <Sem. action>$_n$ }
;

Supporting C routines

Using Yacc with Ambiguous Grammars

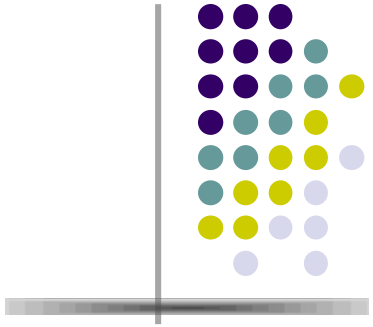


- *Default conflict resolution:*
  - *Reduce/reduce: choose the conflicting production listed first.*
  - *Shift/reduce: choose the shift action.*
- *Explicit conflict resolution:*
  - **%left t_1 t_2 .. t_n***: same precedence and left associative.*
  - **%right t***: right associative.*
  - **%nonassoc t***: nonassociative binary operator*

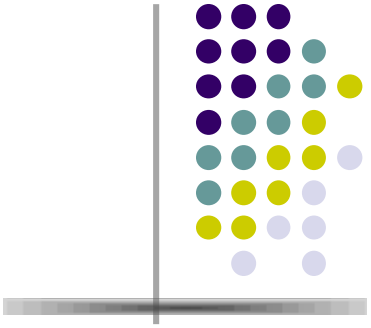
A more complex calculator

```
%{
#include <ctype.h>
#include <stdio.h>
#define YYSTYPE double  /* double type for Yacc stack */
%}
%token NUMBER
%left '+' '-'
%left '*' '/'
%right UMINUS
%%
lines : lines expr '\n'   { printf("%g\n", $2); }
      | lines '\n'
      | /* empty */
      ;
expr  : expr '+' expr     { $$ = $1 + $3; }
      | expr '-' expr     { $$ = $1 - $3; }
      | expr '*' expr     { $$ = $1 * $3; }
      | expr '/' expr     { $$ = $1 / $3; }
      | '(' expr ')'      { $$ = $2; }
      | '-' expr  %prec UMINUS { $$ = - $2; }
      | NUMBER
      ;
%%
yylex() {
    int c;
    while ( ( c = getchar() ) == ' ' );
    if ( (c == '.') || (isdigit(c)) ) {
        ungetc(c, stdin);
        scanf("%lf", &yylval);
        return NUMBER;
    }
    return c;
}
```

* *takes precedence over* +



Using Lex with Yacc



- *Replace yylex() by:*
  - `#include "lex.yy.c"`
- *Compile:*
  - `lex first.l`
  - `yacc second.y`
  - `cc y.tab.c -ly -ll`

first.l

```
number    [0-9]+\e.?|[0-9]*\e.[0-9]+
%%
[ ]       { /* skip blanks */ }
{number} { sscanf(yytext, "%lf", &yylval);
                return NUMBER; }
\n|.     { return yytext[0]; }
```

Error recovery in Yacc

```
%{
#include <ctype.h>
#include <stdio.h>
#define YYSTYPE double  /* double type for Yacc stack */
%}
%token NUMBER

%left '+' '-'
%left '*' '/'
%right UMINUS
%%

lines : lines expr '\n'   { printf("%g\n", $2); }
      | lines '\n'
      | /* empty */
      | error '\n' { yyerror("reenter previous line:");
                     yyerrok; }
      ;
expr  : expr '+' expr      { $$ = $1 + $3; }
      | expr '-' expr      { $$ = $1 - $3; }
      | expr '*' expr      { $$ = $1 * $3; }
      | expr '/' expr      { $$ = $1 / $3; }
      | '(' expr ')'       { $$ = $2; }
      | '-' expr  %prec UMINUS { $$ = - $2; }
      | NUMBER
      ;
%%
#include "lex.yy.c"
```