Lecture 10: LR parsing and resolving conflicts

- What are conflicts
- Example 1: a simple, unambiguous grammar
  - ocamlyacc output
  - Using parse trees to understand conflict
  - Fixing conflict
- Example 2: ambiguous grammar for conditional expressions
  - Eliminating conflicts using %prec declarations

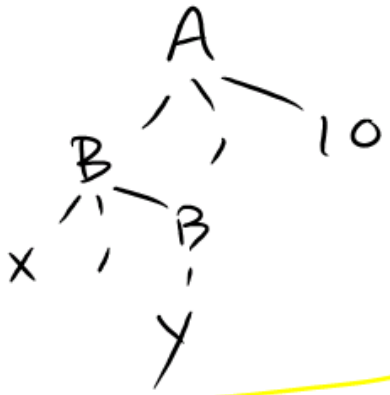
Conflicts

- ocamlyacc generates tables saying what action to take at each point in parse
  - Method is called "LALR(1)"
  - "LR(1)" is a similar, but somewhat more powerful, method — will often use "LR(1)" and "LALR(1)" as synonyms.
- Not every grammar can be parsed using this method.
  - Problem is *always* that ocamlyacc cannot decide on the proper action in some cases
  - "Shift/reduce conflict" — cannot decide whether to shift or reduce
  - "Reduce/reduce conflict" — knows to reduce, but can't decide which production to use

Example 1

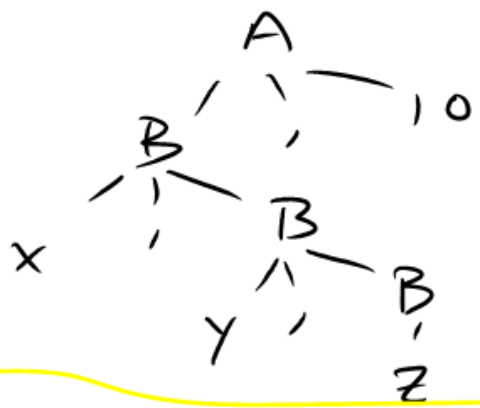
- $A \rightarrow B\ ,\ int$
  $B \rightarrow id \mid id\ ,\ B$
- **Grammar is unambiguous, but consider these two inputs:**
  - **x,y,10**
  - **x,y,z,10**
- **Both lead to an identical stack/lookahead configuration, but the correct action in one case is shift and in the other is reduce.**
- **Look at s-r parse, and at two parse trees.**

x, y, 10



Sh		x, y, 10
Sh	x	, y, 10
Sh	x,	y, 10
R	x, y	, 10
R	x, B	
	B	, 10

x, y, z, 10



Sh		x, y, z, 10
Sh	x	, y, z, 10
Sh	x,	y, z, 10
Sh	x, y	, z, 10
R	x, y, z	, 10
	x, y, B	, 10

A → B, int
B → id
| id, B

Example 1 (cont.)

- **Presented to ocamlyacc:**

```
%token int id comma
%start A
%type <int> A
%%
A: B comma int  {0}
B: id           {0}
 | id comma B   {0}
```

- **Using "ocamlyacc -v", file simple.output contains:**

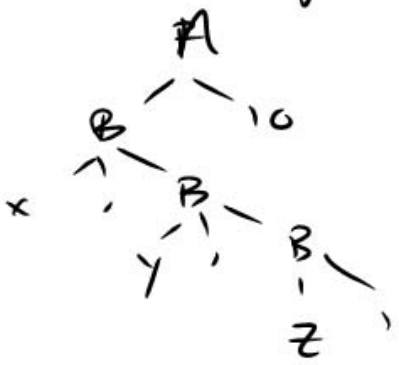
```
3: shift/reduce conflict (shift 6, reduce 2) on comma
state 3
B : id .  (2)
B : id . comma B  (3)

comma  shift 6
```

Example 1 (cont.)

- **One way to fix grammar:**

$A \rightarrow B\ int$
$b \rightarrow id\ ,\ |\ id\ ,\ B$

Sh x 2
Sh

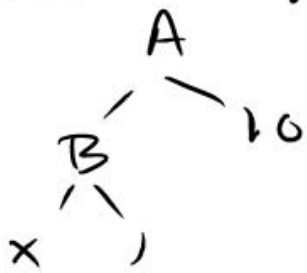


x , y , z , 10
y , z , 10
, z , 10

x , y ,

z , 10

if id on stack: shift
if id + , on stack:
if lookahead is
id : shift
if number : reduce
if comma : reject

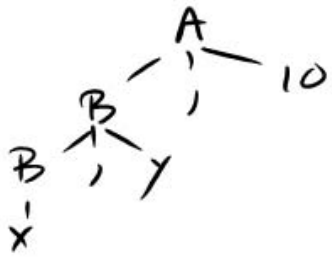


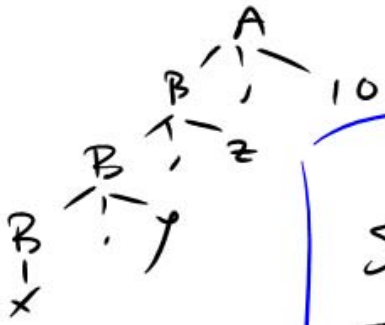
x , , z , 10

Example 1 (cont.)

- **Another way to fix grammar:**

$A \rightarrow B\ ,\ int$
$b \rightarrow id \mid B\ ,\ id$





Stack

R x , y , ...
Sh x2 B , y , ...

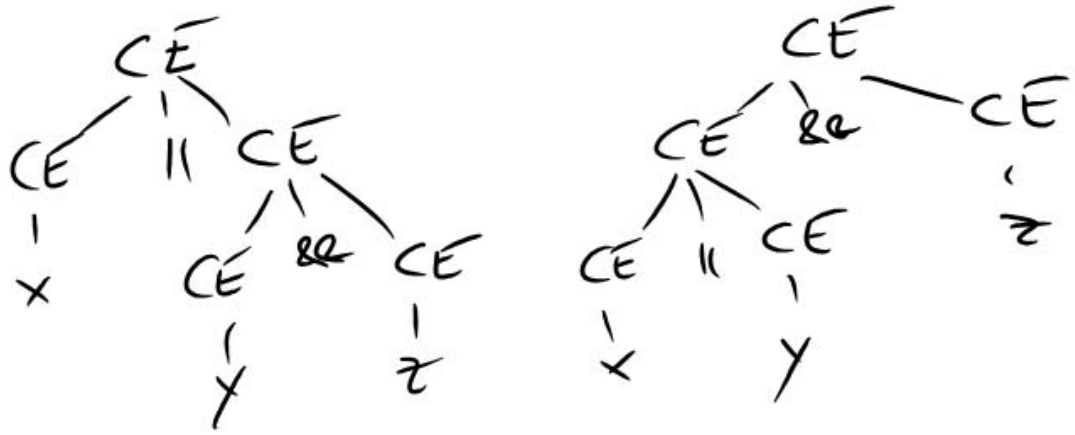
R B , y , ...
Sh x2 B , 10
B , 10

Sh B , z , 10
B , z
R B' , 10

Stack + lookahead symbol gives enough info. to take correct parse action

Example 2

- **Ambiguous grammar for conditional expressions:**
- $CondExpr \rightarrow \; id \mid CondExpr \;||\; CondExpr$
  $\mid CondExpr \;\&\&\; CondExpr \mid \;!\; CondExpr$
- **Consider this input:**
  - `x || y && z`
- **Leads to a stack/lookahead configuration in which shifting and reducing both work, but produce different parse trees.**
- **Look at s-r parse, and at two parse trees.**



x || y && z

Stack		Input
Sh		x || y && z
R	x	|| y && z
Sh x2	CE	|| y && z
R	CE || y	&& z
Sh x2	CE || CE	&& z

Sh x2 → R

R	CE || CE && z	Sh x2	CE	&& z	
R	CE || CE && CE	R	CE && z		c.f
R	CE || CE	R	CE && CE		
	CE		CE		

Example 2 (cont.)

- **ocamlyacc -v output contains:**

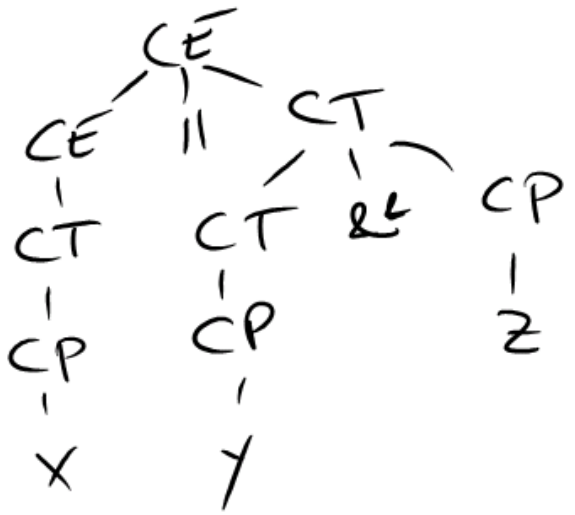
```
10: shift/reduce conflict (shift 7, reduce 2) on and
10: shift/reduce conflict (shift 8, reduce 2) on or
state 10
CondExpr : CondExpr . or CondExpr  (2)
CondExpr : CondExpr or CondExpr .  (2)
CondExpr : CondExpr . and CondExpr  (3)

and  shift 7
or  shift 8
$end  reduce 2
```

Example 2 (cont.)

- **One way to resolve conflict: fix grammar.**
- **Use "stratified grammar," as for arithmetic expressions:**

$$CondExpr \rightarrow CondTerm \mid CondExpr \;||\; CondTerm$$
$$CondTerm \rightarrow CondPrimary \mid CondTerm \;\&\&\; CondPrimary$$
$$CondPrimary \rightarrow id \mid !\, CondPrimary$$

x || y && z



Example 2 (cont.)

- Another way to resolve conflict: precedence declarations.
- Suppose t_1 is the topmost terminal symbol on the stack, and t_2 is the lookahead symbol. Then:
  - If t_1, t_2 appear in the same `%left` declaration, then reduce
  - If t_1, t_2 appear in the same `%right` declaration, then shift
  - If t_1 appears in a declaration *before* t_2, shift
  - If t_1 appears in a declaration *after* t_2, reduce

%left token, ..
%right token, ...
%nonassoc token, ...

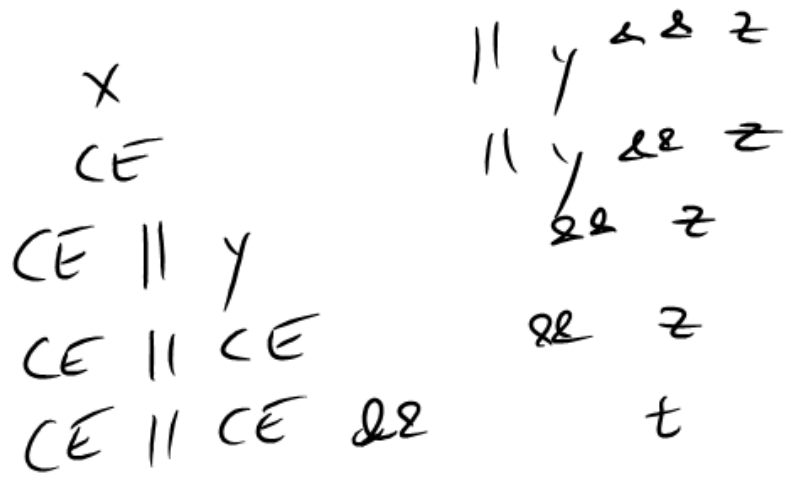
CE || CE || ...
CE || CE || ...
CE || (CE || CE)

Example 2 (cont.)

- **Use the ambiguous grammar, but add these declarations:**

```
%left or
%left and
```

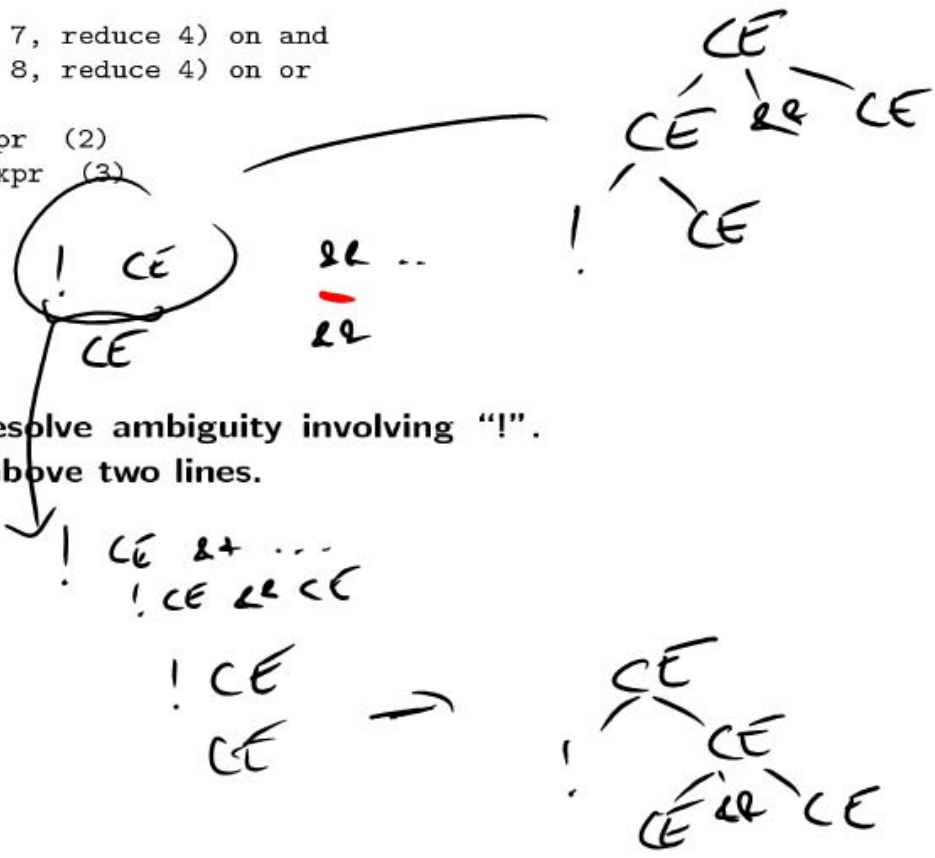
- `x || y && z` **is now handled correctly.**



Example 2 (cont.)

- **However, ocamlyacc still reports conflicts. Verbose output:**

```
6: shift/reduce conflict (shift 7, reduce 4) on and
6: shift/reduce conflict (shift 8, reduce 4) on or
state 6
CondExpr : CondExpr . or CondExpr  (2)
CondExpr : CondExpr . and CondExpr  (3)
CondExpr : not CondExpr .  (4)

and  shift 7
or  shift 8
$end  reduce 4
```



- **Problem is that we didn't resolve ambiguity involving "!". Add "%nonassoc not" after above two lines.**

More on conflicts

- Posted supplementary notes discuss four grammars that have conflicts, and how to resolve them.
- All are relevant to the current MP.