Compilers and Language Processing Tools

Summer Term 2011

Prof. Dr. Arnd Poetzsch-Heffter

Software Technology Group
TU Kaiserslautern

Parser Generators

Preparations

```
$ mkdir clp11
```

```
$ cd clp11
```

ANTLR

ANTLR

- Parser generator for LL(k) grammars (LL(*))
- Integrates Scanner generator and AST generator

ANTLR

ANTLR

- Parser generator for LL(k) grammars (LL(*))
- Integrates Scanner generator and AST generator

```
$ java -jar /home/j_schaef/public/antlrworks-1.4.2.jar
```

Simple Arithmetic Expressions

Simple arithmetic expressions: 5+4+33

Simple Arithmetic Expressions

Simple arithmetic expressions: 5+4+33

Enter the following grammar into ANTLR:

```
grammar test;
expr : expr '+' expr
     | NUM;
NUM  : ('0'..'9')*;
```

Simple Arithmetic Expressions

Simple arithmetic expressions: 5+4+33

Enter the following grammar into ANTLR:

```
grammar test;
expr : expr '+' expr
     | NUM;
NUM  : ('0'..'9')*;
```

ANTLR Error: Rule 'expr' is left-recursive

Remove Left-Recursion

Remove Left-Recursion

```
grammar test;
expr : NUM exprp
     | NUM;
exprp : '+' expr;
NUM  : ('0'..'9')*;
```

Java CUP

Java CUP

Parser generator for LALR grammars

Simple Expressions

```
expr   ::= expr '+' expr
         | number

number ::= 0
         | [1-9][0-9]+
```

Steps

Steps

1. Identify and name the terminal and non-terminal symbols

Steps

1. Identify and name the terminal and non-terminal symbols
2. Define a grammar

Steps

1. Identify and name the terminal and non-terminal symbols
2. Define a grammar
3. Define an unambiguous grammar

First Try

Parser.cup:

```
terminal PLUS, NUMBER;
non terminal expr;

expr ::= expr PLUS expr
       | NUMBER
       ;
```

First Try

Parser.cup:

```
terminal PLUS, NUMBER;
non terminal expr;

expr ::= expr PLUS expr
       | NUMBER
       ;
```

```
$ java -jar java-cup.jar Parser.cup
```

Java CUP Output

```
Warning : *** Shift/Reduce conflict found in state #5
between expr ::= expr PLUS expr (*)
and     expr ::= expr (*) PLUS expr
under symbol PLUS
Resolved in favor of shifting.
Error : *** More conflicts encountered than expected --
parser generation aborted
```

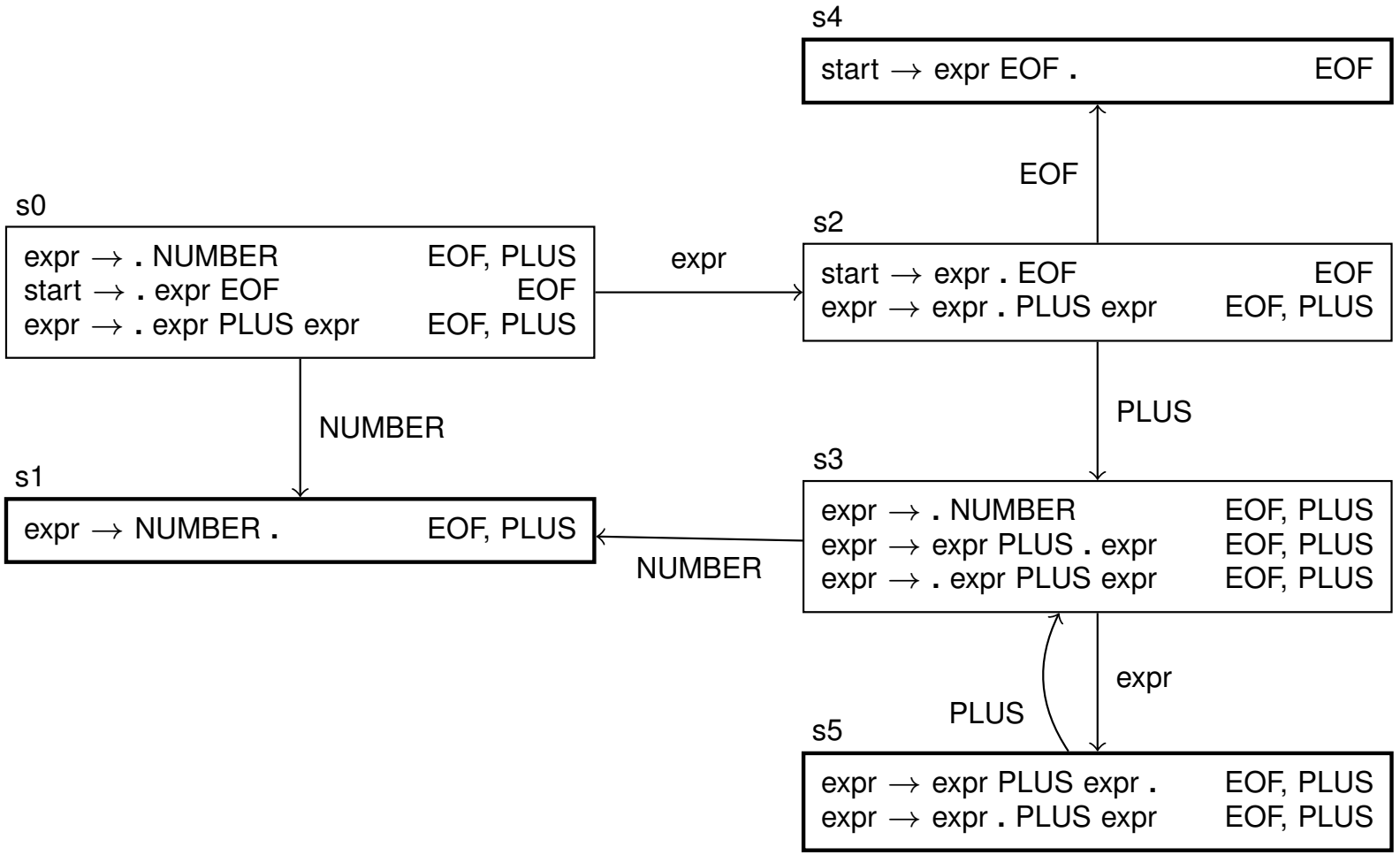
Debugging the Generated Parser

Use the `-dump_states` option of JavaCUP.

```
$ java -jar java-cup.jar -dump_states Parser.cup
```

```
===== Viable Prefix Recognizer =====
START lalr_state [0]: {
  [expr ::= (*) NUMBER , {EOF PLUS }]
  [$START ::= (*) expr EOF , {EOF }]
  [expr ::= (*) expr PLUS expr , {EOF PLUS }]
}
transition on expr to state [2]
transition on NUMBER to state [1]

-------------------
lalr_state [1]: {
  [expr ::= NUMBER (*) , {EOF PLUS }]
}

-------------------
lalr_state [2]: {
  [$START ::= expr (*) EOF , {EOF }]
  [expr ::= expr (*) PLUS expr , {EOF PLUS }]
}
transition on EOF to state [4]
transition on PLUS to state [3]

-------------------
lalr_state [3]: {
  [expr ::= (*) NUMBER , {EOF PLUS }]
  [expr ::= expr PLUS (*) expr , {EOF PLUS }]
  [expr ::= (*) expr PLUS expr , {EOF PLUS }]
}
transition on expr to state [5]
transition on NUMBER to state [1]

-------------------
lalr_state [4]: {
  [$START ::= expr EOF (*) , {EOF }]
}

-------------------
lalr_state [5]: {
  [expr ::= expr PLUS expr (*) , {EOF PLUS }]
  [expr ::= expr (*) PLUS expr , {EOF PLUS }]
}
transition on PLUS to state [3]
```

LALR(1) Item Automata



Unambiguous Grammar

```
expr ::= expr PLUS expr
       | NUMBER
       ;
```

Unambiguous Grammar?

Unambiguous Grammar

```
expr ::= expr PLUS expr
       | NUMBER
       ;
```

Unambiguous Grammar?
Grammar Γ_3 from lecture (p.71):

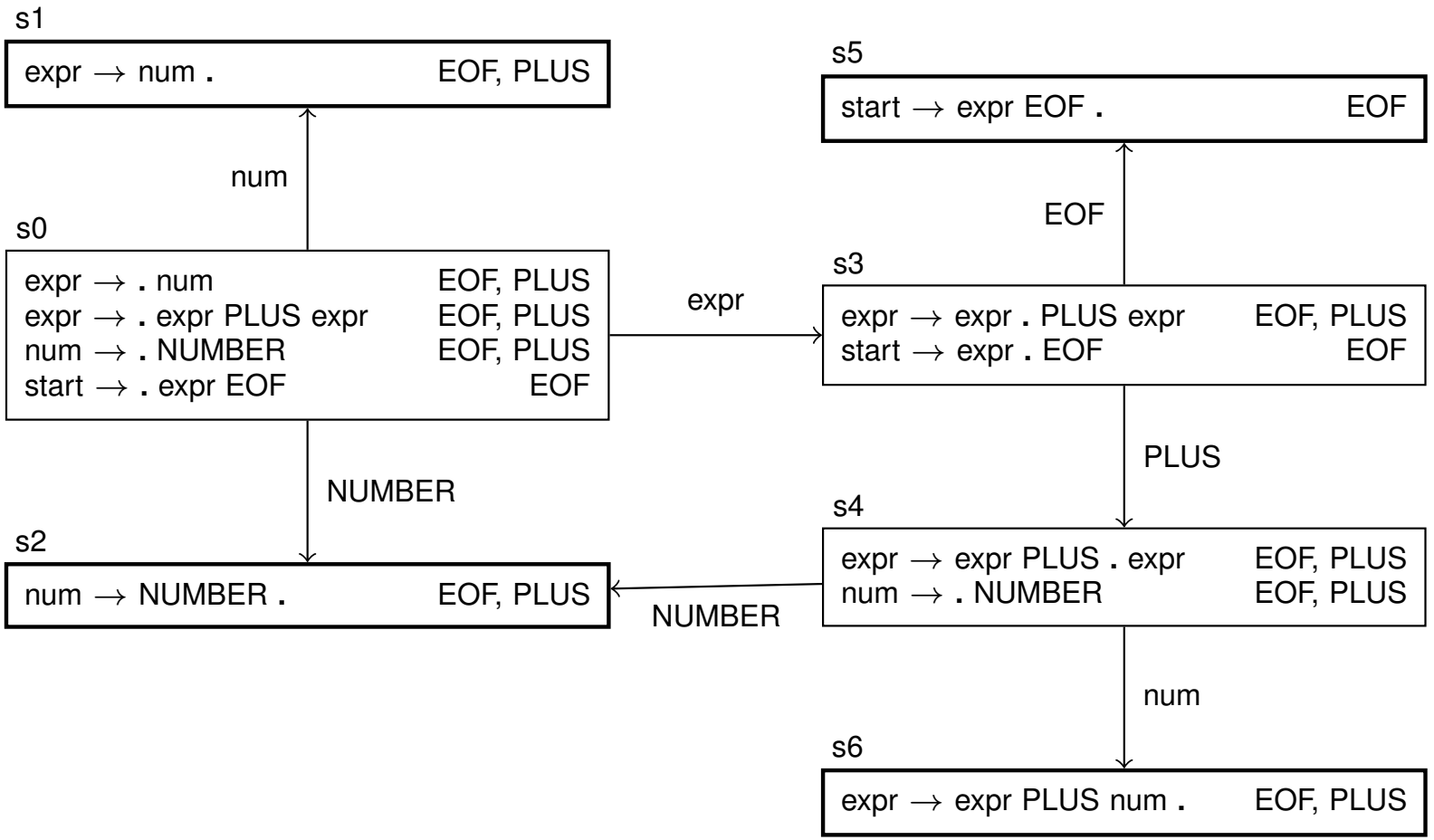
```
expr ::= expr PLUS num
       | num
       ;
num ::= NUMBER;
```

JavaCUP States

```
===== Viable Prefix Recognizer =====
START lalr_state [0]: {
  [expr ::= (*) num , {EOF PLUS }]
  [expr ::= (*) expr PLUS num , {EOF PLUS }]
  [num ::= (*) NUMBER , {EOF PLUS }]
  [$START ::= (*) expr EOF , {EOF }]
}
transition on expr to state [3]
transition on NUMBER to state [2]
transition on num to state [1]

-------------------
lalr_state [1]: {
  [expr ::= num (*) , {EOF PLUS }]
}

-------------------
lalr_state [2]: {
  [num ::= NUMBER (*) , {EOF PLUS }]
}
```

```
-------------------
lalr_state [3]: {
  [expr ::= expr (*) PLUS num , {EOF PLUS }]
  [$START ::= expr (*) EOF , {EOF }]
}
transition on EOF to state [5]
transition on PLUS to state [4]

-------------------
lalr_state [4]: {
  [expr ::= expr PLUS (*) num , {EOF PLUS }]
  [num ::= (*) NUMBER , {EOF PLUS }]
}
transition on NUMBER to state [2]
transition on num to state [6]

-------------------
lalr_state [5]: {
  [$START ::= expr EOF (*) , {EOF }]
}

-------------------
lalr_state [6]: {
  [expr ::= expr PLUS num (*) , {EOF PLUS }]
}
```

LALR(1) Item Automata



Action and Reduce Tables

```
-dump_tables -dump_grammar
-------- ACTION_TABLE --------
From state #0
 [term 3:SHIFT(to state 2)]
From state #1
 [term 0:REDUCE(with prod 2)]
 [term 2:REDUCE(with prod 2)]
From state #2
 [term 0:REDUCE(with prod 3)]
 [term 2:REDUCE(with prod 3)]
From state #3
 [term 0:SHIFT(to state 5)]
 [term 2:SHIFT(to state 4)]
From state #4
 [term 3:SHIFT(to state 2)]
From state #5
 [term 0:REDUCE(with prod 0)]
From state #6
 [term 0:REDUCE(with prod 1)]
 [term 2:REDUCE(with prod 1)]
------------------------------
-------- REDUCE_TABLE --------
From state #0
 [non term 0->state 3]
 [non term 1->state 1]
From state #1
From state #2
From state #3
From state #4
 [non term 1->state 6]
From state #5
From state #6

Terminals:
  [0]EOF [1]error [2]PLUS [3]NUMBER
Non terminals:
  [0]expr [1]num
Productions:
  [0] $START ::= expr EOF
  [1] expr ::= expr PLUS num
  [2] expr ::= num
  [3] num ::= NUMBER
```

Example

1+2+3 => (Scanner) => NUMBER PLUS NUMBER PLUS NUMBER EOF
= NPNPNE

State	Stack	Input Rest	Action
#0	#0	NPNPNE	shift and goto #2
#2	#0 N#2	PNPNE	reduce and goto #1
#1	#0 num#1	PNPNE	reduce and goto #3
#3	#0 expr#3	PNPNE	shift and goto #4
#4	#0 expr#3 P#4	NPNE	shift and goto #2
#2	#0 expr#3 P#4 N#2	PNE	reduce and goto #6
#6	#0 expr#3 P#4 num#6	PNE	reduce and goto #3
#3	#0 expr#3	PNE	shift and goto #4
#4	#0 expr#3 P#4	NE	shift and goto #2
#2	#0 expr#3 P#4 N#2	E	reduce and goto #6
#6	#0 expr#3 P#4 num#6	E	reduce and goto #3
#3	#0 expr#3	E	shift and goto #5
#5	#0 expr#3 E#5		reduce and goto #3
#3	#0 start#3		

Associativities

$4 + 5 + 6 = (4 + 5) + 6$

Associativities

4 + 5 + 6 = (4 + 5) + 6

JavaCUP: Associativities on terminal symbols:

```
left, right, nonassoc
```

Associativities

4 + 5 + 6 = (4 + 5) + 6

JavaCUP: Associativities on terminal symbols:

```
left, right, nonassoc
```

Example:

```
precedence left PLUS;

expr ::= expr PLUS expr
       | NUMBER
       ;
```

Precedences

5 + 6 * 3 = 5 + (6 * 3)

Precedences

5 + 6 * 3 = 5 + (6 * 3)

JavaCUP: Precedence on terminal symbols:

- same line = equal precedences,
- farther down = higher precedence

Precedences

5 + 6 * 3 = 5 + (6 * 3)

JavaCUP: Precedence on terminal symbols:

- same line = equal precedences,
- farther down = higher precedence

Example:

```
precedence left PLUS, MINUS;  // PLUS same as MINUS
precedence left MULT, DIV;    // higher than PLUS and MINUS

expr ::= expr infixop expr
       | NUMBER ;
infixop ::= PLUS | MINUS | MULT | DIV;
```

Rule Precedences

Rules have the precedence of their last terminal symbol

Rule Precedences

Rules have the precedence of their last terminal symbol

5 - - 4?

Rule Precedences

Rules have the precedence of their last terminal symbol

5 - - 4?

```
expr ::= expr MINUS expr
       | MINUS expr
       | NUMBER ;
```

Rule Precedences

Rules have the precedence of their last terminal symbol

5 - - 4?

```
expr ::= expr MINUS expr
       | MINUS expr
       | NUMBER ;
```

Solution: Introduce "dummy" terminal symbol UMINUS:

```
precedence left MINUS;
precedence left UMINUS;
expr ::= expr MINUS expr
       | MINUS expr %prec UMINUS
       | NUMBER ;
```

Note: UMINUS is never returned by the scanner!

Semantic versus Syntax

Arithmetic and Boolean expressions For example:

```
a := 5
b := 4+a
c := 3 = 3
d := 2+1 = 3 & c
```

Semantic versus Syntax

Arithmetic and Boolean expressions For example:

```
a := 5
b := 4+a
c := 3 = 3
d := 2+1 = 3 & c
```

Grammar:

```
terminal ID, EQUAL, PLUS, ASSIGN, AND;
non terminal ae, be, stm;

precedence left AND;
precedence left PLUS;

stm ::= ID ASSIGN ae | ID ASSIGN be;
be  ::= ae EQUAL ae | be AND be | ID;
ae  ::= ae PLUS ae | ID;
```

Handle Semantics Later

```
terminal ID, EQUAL, PLUS, ASSIGN, AND;
non terminal e, stm;

precedence left AND;
precedence left PLUS;
precedence left EQUAL;

stm ::= ID ASSIGN e;
e   ::= ID
      | e EQUAL e
      | e AND e
      | e PLUS e;
```

Error Recovery

Problem

- Typically JavaCUP stops when it encounters an error
  - Only the first error is shown

Solution

- Local error recovery

Example

```
expr     ::= expr PLUS expr | ID;
exprlist ::= expr SEMI expr;
```

We want to be able to skip erroneous expressions in a list.

Example

```
expr     ::= expr PLUS expr | ID;
exprlist ::= expr SEMI expr;
```

We want to be able to skip erroneous expressions in a list.

Introduce error production with special non-terminal error symbol.

```
exprlist ::= error SEMI expr;
```

Important: the `error` symbol should always be followed by a terminal synchronization symbol, `SEMI` in this example.