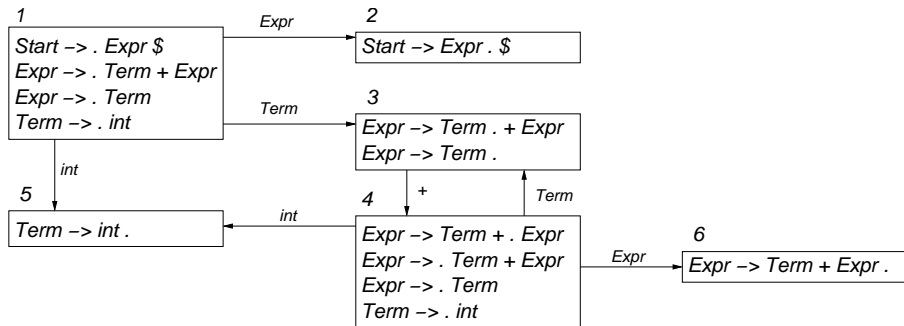


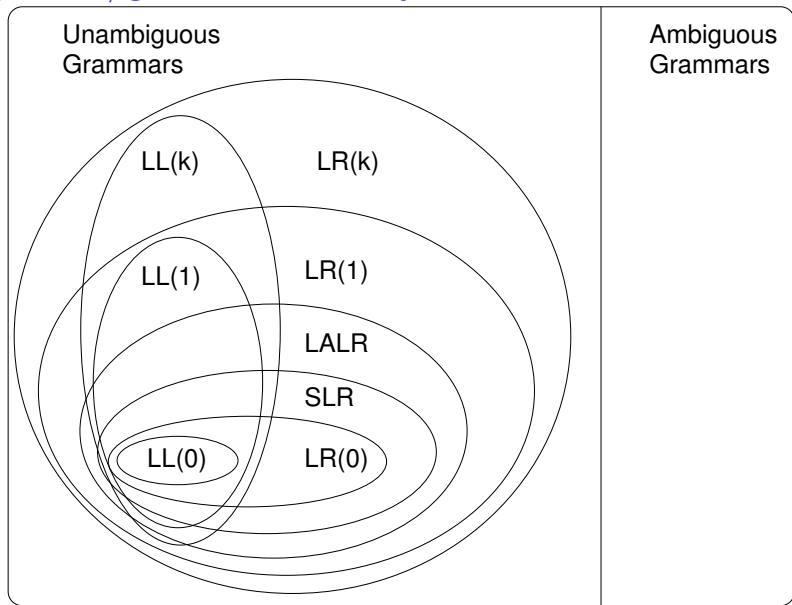
MathEx- -

Start → *Expression* \$
Expression → *Term* PLUS *Expression* | *Term*
Term → INTEGER

SLR states



Algorithm/grammar hierarchy



Redrawn after Appel, *Modern Compiler Implementation*, 1998, pg 68.

Jay Lexical Specification

Identifiers	<code>[a-z_][a-zA-Z0-9_]*</code>
capitalized identifiers	<code>[A-Z][a-zA-Z0-9_]*</code>
Integer literals	<code>0 [1-9][0-9]*</code>
Boolean literals	<code>true false</code>
Separators	<code>() { } ; ,</code>
Operators	<code>= > < ! == <= >= != && + - * /</code>
Keywords	<code>public class static String[] args void main System.out.println boolean else if int while</code>

Jay Concrete Syntax

<i>Program</i>	→	public class CAPID '{' public static void main (String[] args) '{ Declarations Statements }' }'
<i>Declarations</i>	→	<i>Declaration</i> *
<i>Declaration</i>	→	<i>Type</i> <i>Identifiers</i> ;
<i>Type</i>	→	int boolean
<i>Identifiers</i>	→	ID { , ID } *
<i>Statements</i>	→	<i>Statement</i> *
<i>Statement</i>	→	<i>Skip</i> <i>Block</i> <i>Assignment</i> <i>IfStatement</i> <i>WhileStatement</i> <i>PrintStatement</i>
<i>Skip</i>	→	;
<i>Block</i>	→	'{ Statements }'
<i>Assignment</i>	→	ID = <i>Expression</i> ;
<i>IfStatement</i>	→	if (<i>Expression</i>) <i>Statement</i> { else <i>Statement</i> }?
<i>WhileStatement</i>	→	while (<i>Expression</i>) <i>Statement</i>
<i>PrintStatement</i>	→	System.out.println (<i>Expression</i>) ;

Jay Concrete Syntax, continued

<i>Expression</i>	→	<i>Conjunction</i> { <i>Conjunction</i> }*
<i>Conjunction</i>	→	<i>Relation</i> { && <i>Relation</i> }*
<i>Relation</i>	→	<i>Addition</i> { <i>RelOp</i> <i>Addition</i> }?
<i>RelOp</i>	→	< <= > >= == !=
<i>Addition</i>	→	<i>Term</i> { <i>AddOp</i> <i>Term</i> } *
<i>AddOp</i>	→	+ -
<i>Term</i>	→	<i>Negation</i> { <i>MulOp</i> <i>Negation</i> } *
<i>MulOp</i>	→	'*' /
<i>Negation</i>	→	<i>NegOp</i> ? <i>Factor</i>
<i>NegOp</i>	→	! -
<i>Factor</i>	→	ID LITERAL (<i>Expression</i>)

Jay Abstract Syntax

<i>Program</i>	→	<i>Declaration* Statement</i>
<i>Declaration</i>	→	<i>Type ID*</i>
<i>Statement</i>	→	<i>Skip Block Assignment Conditional Loop Print</i>
<i>Block</i>	→	<i>Statement*</i>
<i>Assignment</i>	→	<i>ID Expression</i>
<i>Conditional</i>	→	<i>Expression Statement Statement</i>
<i>Loop</i>	→	<i>Expression Statement</i>
<i>Print</i>	→	<i>Expression</i>
<i>Expression</i>	→	<i>Variable IntLitExpr BoolLitExpr BinaryExpr UnaryExpr</i>
<i>Variable</i>	→	<i>ID</i>
<i>IntLitExpr</i>	→	<i>INT_LIT</i>
<i>BoolLitExpr</i>	→	<i>BOOL_LIT</i>
<i>BinaryExpr</i>	→	<i>Expression OPERATOR Expression</i>
<i>UnaryExpr</i>	→	<i>OPERATOR Expression</i>