Compiler Practical 2013
Syntax Analysis of Expressions

Berthold Hoffmann (B. Gersdorf, T. Röfer)
hof@informatik.uni-bremen.de
Cartesium 2.48
Structure

1. Syntax Analysis of Expressions
2. Tasks: Adding AND, OR, and NOT
3. Task: Adding the Class Boolean
4. Bonus Task: Adding AND THEN and OR ELSE
# Precedence of Operators

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operators</th>
<th>Assoziativity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>(prefix) right</td>
<td>object generation</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>left</td>
<td>object access</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>(prefix) right</td>
<td>sign</td>
<td></td>
</tr>
<tr>
<td>* / MOD</td>
<td>left</td>
<td>multiplication, division, modulus</td>
<td></td>
</tr>
<tr>
<td>+ -</td>
<td>left</td>
<td>addition, subtraction</td>
<td></td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;= = #</td>
<td>Not associative</td>
<td>Less, less-or-equal, greater, greater-or-equal, equality, inequality</td>
<td></td>
</tr>
</tbody>
</table>
Expressions: Grammar

```
relation ::= expression
    [ ( '=' | '#' | '<' | '>' | '<=' | '>=') expression ]
expression ::= term { ( '+' | '-' ) term }
term ::= factor { ( '*' | '/' | MOD ) factor }
factor ::= '!' factor
    | memberaccess
memberaccess ::= literal { '.' varorcall }
Literal ::= number
    | NULL
    | SELF
    | NEW identifier
    | '(' expression ')' 
    | varorcall
```
Expressions: Abstract Syntax

Syntaxanalyse von Ausdrücken

Diagram showing the abstract syntax of expressions, including nodes for `AccessExpression`, `UnaryExpression`, `UnBoxExpression`, `NewExpression`, `DeRefExpression`, `BoxExpression`, `Symbol`, `Position`, `TreeStream`, `SyntaxAnalysis`, `LexicalAnalysis`, `CompileException`, `VarOrCall`, `BinaryExpression`, `LiteralExpression`, `VarDeclaration`, `ClassDeclaration`, `MethodDeclaration`, `ReadStatement`, `WhileStatement`, `IfStatement`, `CallStatement`, and `WriteStatement`. The diagram also includes a `Program`, `Declarations`, `OOPSC`, `CodeStream`, and `ResolvableIdentifier` nodes.
Task: AND, OR, NOT

• Lexical analysis
  – Add keywords AND, OR, NOT

• Extend the grammar
  – Respect precedence of operators
  – In analogy with expressions / term / factor

• Bracketed expressions can be more general now

IF NOT (a < b AND b < c)
  AND (a < b) # (b < c)
  OR c > a
THEN
...
Task: AND, OR, and NOT

- Extend syntax analysis
  - New methods for new productions
  - Extending factor

- Extend syntax tree
  - UnaryExpression, BinaryExpression
  - and contextAnalysis(...), generateCode(...), resp.

5%
### Aufgabe: AND, OR, NOT Bindung

<table>
<thead>
<tr>
<th>precedence</th>
<th>Operators</th>
<th>Assoziativity</th>
<th>Decription</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NEW</td>
<td>(prefix) right</td>
<td>object generation</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td>left</td>
<td>object access</td>
</tr>
<tr>
<td></td>
<td>-, NOT</td>
<td>(prefix) right</td>
<td>sign</td>
</tr>
<tr>
<td></td>
<td>*, /, MOD</td>
<td>left</td>
<td>multiplication, division, modulus</td>
</tr>
<tr>
<td></td>
<td>+, -</td>
<td>left</td>
<td>addition, subtraction</td>
</tr>
<tr>
<td></td>
<td>&lt;, &lt;=, &gt;, &gt;=, =, #</td>
<td>not associative</td>
<td>Less, less-or-equal, greater, greater-or-equal, equality, inequality</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>left</td>
<td>conjunction</td>
</tr>
<tr>
<td></td>
<td>OR</td>
<td>left</td>
<td>Disjunction</td>
</tr>
</tbody>
</table>

21.03.2011
Task: Adding the class Boolean

• New type
  – Class attribute for new predefined type in ClassDeclaration
  – Initializing type, and introducing it in Program.contextAnalysis()

• [Un]Boxing
  – Expression.box/unBox(...) 
  – [Un]BoxExpression.[Un]BoxExpression(...)
  – Analoguously to ClassDeclaration.intClass

10%

METHOD main IS
  a, b : Boolean;
BEGIN
  a := 5 > 7;
  b := 1 < 2 OR a;
  a := a # b;
  IF b THEN
    END IF
END IF
END METHOD
• Both operators stop evaluation as soon as the result is clear
  – $a \text{ AND THEN } b \text{ AND THEN } c$
  – $a \text{ OR ELSE } b \text{ OR ELSE } c$

• Problems
  – How should the grammar be defined?
  – How should the code be generated?
  – What happens with $a \text{ AND THEN } b \text{ AND } c$?