Operator Precedence Levels.

Jeg startet med denne tabelen:

14	:= :-
13	OR_ELSE
12	AND_THEN
11	EQV
10	IMP
9	OR
8	AND
7	NOT
6	< <= > >= = <> == =/= IS IN
5	+ -
4	* / //
3	**
2	unary(+ -) NOT QUA THIS
1	. Remote access

Og denne algoritmen:

```java
protected static Expression parseSimpleExpression() {
    return(parseBinaryOperation(14));  // 14: assignment
}

private static Expression parseBinaryOperation(final int level) {
    Expression expr=(level>0)?parseBinaryOperation(level-1):parsePrimaryExpression();

    while(Parser.acceptBinaryOperator(level)) {
        KeyWord opr=Parser.prevToken.getKeyWord();
        if(level==0) expr=new BinaryOperation(expr,opr,parsePrimaryExpression());
        else if(opr==KeyWord.ASSIGNVALUE || opr==KeyWord.ASSIGNREF)
             expr=new BinaryOperation(expr,opr,parseExpression());
        else expr=new BinaryOperation(expr,opr,parseBinaryOperation(level-1));
    }
    return(expr);
}
```

Dette virka 'nesten bestandig' Jeg fikk bare trøbbel med level 1 og 2.
Jeg har prøvd å bytte om DOT og QUA og trekke ut unary og det bedret situasjonen.

Men ikke helt.

Til info: parsePrimaryExpression ser slik ut:

```java
public static Expression parsePrimaryExpression()
{ // PrimaryExpression =  ( Expression ) | Constant | ObjectGenerator
  //                    | LocalObject | UnaryOperation | Variable | SubscriptedVariable
  //     Constant = IntegerConstant | RealConstant | CharacterConstant
  //              | TextConstant | BooleanConstant | SymbolicValue
  //     BooleanConstant = TRUE | FALSE
  //     Boolean-secondary =  [ NOT ]  Boolean-primary
  //     SymbolicValue = NONE | NOTEXT
  //     LocalObject = THIS ClassIdentifier

  if (Parser.accept(KeyWord.BEGPAR)) { Expression expr=parseExpression();
                                       Parser.expect(KeyWord.ENDPAR); return(expr); }
  else if(Parser.accept(KeyWord.INTEGERKONST)) return(new
                                 Constant(Type.Integer,Parser.prevToken.getValue()));
  else if(Parser.accept(KeyWord.REALKONST)) return(new
                                 Constant(Type.LongReal,Parser.prevToken.getValue()));
  else if(Parser.accept(KeyWord.BOOLEANKONST)) return(new
                                 Constant(Type.Boolean,Parser.prevToken.getValue()));
  else if(Parser.accept(KeyWord.CHARACTERKONST)) return(new
                                 Constant(Type.Character,Parser.prevToken.getValue()));
  else if(Parser.accept(KeyWord.TEXTKONST)) return(new
                                 Constant(Type.Text,Parser.prevToken.getValue()));
  else if(Parser.accept(KeyWord.NONE)) return(new Constant(Type.Ref,null));
  else if(Parser.accept(KeyWord.NOTEXT)) return(new Constant(Type.Text,null));
  else if(Parser.accept(KeyWord.NEW)) return(ObjectGenerator.parse()); // TODO
  else if(Parser.accept(KeyWord.THIS)) return(LocalObject.acceptThisIdentifier());
  else if(Parser.accept(KeyWord.PLUS)) return(parseUnaryOperation());
  else if(Parser.accept(KeyWord.MINUS)) return(parseUnaryOperation());
  else if(Parser.accept(KeyWord.NOT))
      { // Boolean-secondary =  [ NOT ]  Boolean-primary
        Expression expr=parseBooleanPrimary();
        expr=new UnaryOperation(KeyWord.NOT,expr);
        return(expr);

      }
  else if(Parser.accept(KeyWord.IF))
    { Expression condition=parseExpression();
      Parser.expect(KeyWord.THEN); Expression thenExpression=parseSimpleExpression();
      Parser.expect(KeyWord.ELSE); Expression elseExpression=parseExpression();
      Expression expr=new
                ConditionalExpression(Type.Boolean,condition,thenExpression,elseExpression);
      if(Option.TRACE_PARSE) Util.TRACE("Expression: parsePrimaryExpression, result="+expr);
      return(expr);
    }
  else
    { String ident=acceptIdentifier();
      if(ident!=null) return(Variable.parse(ident));
      return(null);
    }
}
```