Université libre de Bruxelles

Project - Part 2
Parser

Aldar Saranov, Przemyslaw Gasinski

Aldar.Saranov@ulb.ac.be
Przemyslaw.Gasinski@ulb.ac.be

INFO-F403 Introduction to language theory and compiling (M-INFOS/F277)

Gilles Geeraerts

November 2016

Initial grammar:

```
<Program>
     -> PROGRAM [ProgName] [EndLine] <Vars> <Code> END
<Vars>
     -> INTEGER <VarList> [EndLine]
     -> ε
<VarList>
     -> [VarName], <VarList>
     -> [VarName]
<Code>
     -> <Instruction> [EndLine] <Code>
     -> ε
<Instruction>
     -> <Assign>
     -> <If>
     -> <Do>
     -> <Print>
     -> <Read>
<Assign>
     -> [VarName] = <ExprArith>
<ExprArith>
     -> [VarName]
     -> [Number]
     -> (<ExprArith>)
     -> -<ExprArith>
     -> <ExprArith> <Op> <ExprArith>
<Op>
     -> +
     -> -
     -> *
     -> /
<If>
     -> IF (<Cond>) THEN [EndLine] <Code> ENDIF
     -> IF (<Cond>) THEN [EndLine] <Code> ELSE [EndLine] <Code>
     ENDIF
<Cond>
     -> <Cond> <BinOp> <Cond>
     -> .NOT. <SimpleCond>
     -> <SimpleCond>
<SimpleCond>
     -> <ExprArith> <Comp> <ExprArith>
<BinOp>
     -> .AND.
     -> .OR.
<Comp>
     -> .EQ.
     -> .GE.
     -> .GT.
     -> .LE.
     -> .LT.
     -> .NE.
<Do>
     -> DO [VarName] = [Number], [Number] [EndLine] <Code>
     ENDDO
<Print>
     -> PRINT*, <ExpList>
```

```
<Read>
     -> READ*, <VarList>
<ExpList>
     -> <ExprArith>, <ExpList>
     -> <ExprArith>
```

No unproductive or inaccessible symbols found.

Removing left-recursion:

```
<Program>
     -> PROGRAM [ProgName] [EndLine] <Vars> <Code> END
<Vars>
     -> INTEGER <VarList> [EndLine]
     -> ε
<VarList>
     -> [VarName], <VarList>
     -> [VarName]
<Code>
     -> <Instruction> [EndLine] <Code>
     -> ε
<Instruction>
     -> <Assign>
     -> <If>
     -> <Do>
     -> <Print>
     -> <Read>
<Assign>
     -> [VarName] = <ExprArith>
<ExprArith>
     -> [VarName] <ExprArithRec>
     -> [Number] <ExprArithRec>
     -> (<ExprArith>) <ExprArithRec>
     -> -<ExprArith> <ExprArithRec>
<ExprArithRec>
     -> <Op> <ExprArith> <ExprArithRec>
     -> ε
<Op>
     -> +
     -> -
     -> *
     -> /
<If>
     -> IF (<Cond>) THEN [EndLine] <Code> ENDIF
     -> IF (<Cond>) THEN [EndLine] <Code> ELSE [EndLine] <Code>
     ENDIF
<Cond>
     -> .NOT. <SimpleCond> <CondRec>
     -> <SimpleCond> <CondRec>
<CondRec>
     -> <BinOp> <Cond> <CondRec>
     -> ε
<SimpleCond>
```

```
      -> <ExprArith> <Comp> <ExprArith>
<BinOp>
      -> .AND.
      -> .OR.
<Comp>
      -> .EQ.
      -> .GE.
      -> .GT.
      -> .LE.
      -> .LT.
      -> .NE.
<Do>
      -> DO [VarName] = [Number], [Number] [EndLine] <Code>
      ENDDO
<Print>
      -> PRINT*, <ExpList>
<Read>
      -> READ*, <VarList>
<ExpList>
      -> <ExprArith>, <ExpList>
      -> <ExprArith>
```

Applying factorization:

```
<Program>
      -> PROGRAM [ProgName] [EndLine] <Vars> <Code> END
<Vars>
      -> INTEGER <VarList> [EndLine]
      -> ε
<VarList>
      -> [VarName], <FactVarList>
<FactVarList>
      -> <VarList>
      -> ε
<Code>
      -> <Instruction> [EndLine] <Code>
      -> ε
<Instruction>
      -> <Assign>
      -> <If>
      -> <Do>
      -> <Print>
      -> <Read>
<Assign>
      -> [VarName] = <ExprArith>
<ExprArith>
      -> <FactExprArith> <ExprArithRec>
<FactExprArith>
      -> [VarName]
      -> [Number]
      -> (<ExprArith>)
      -> -<ExprArith>
<ExprArithRec>
      -> <Op> <ExprArith> <ExprArithRec>
      -> ε
<Op>
      -> +
      -> -
      -> *
      -> /
<If>
      -> IF (<Cond>) THEN [EndLine] <Code> <FactIf>
<FactIf>
      -> ENDIF
      -> ELSE [EndLine] <Code> ENDIF
<Cond>
      -> <CondPrefix> <SimpleCond> <CondRec>
<CondPrefix>
      -> .NOT.
      -> ε
<CondRec>
      -> <BinOp> <Cond> <CondRec>
      -> ε
<SimpleCond>
      -> <ExprArith> <Comp> <ExprArith>
<BinOp>
      -> .AND.
      -> .OR.
<Comp>
      -> .EQ.
      -> .GE.
      -> .GT.
      -> .LE.
      -> .LT.
      -> .NE.
<Do>
      -> DO [VarName] = [Number], [Number] [EndLine] <Code>
      ENDDO
<Print>
      -> PRINT*, <ExpList>
<Read>
      -> READ*, <VarList>
<ExpList>
      -> <ExprArith>, <FactExprArith>
<FactExprArith>
      -> <ExpList>
      -> ε
```

Making non-ambiguous

```
<Program>
      -> PROGRAM [ProgName] [EndLine] <Vars> <Code> END
<Vars>
      -> INTEGER <VarList> [EndLine]
      -> ε
<VarList>
      -> [VarName], <FactVarList>
<FactVarList>
      -> <VarList>
      -> ε
<Code>
      -> <Instruction> [EndLine] <Code>
```

```
     -> ε
<Instruction>
     -> <Assign>
     -> <If>
     -> <Do>
     -> <Print>
     -> <Read>
<Assign>
     -> [VarName] = <ExprArith>
<ExprArith>
     -> <ArithT> <RecArithE>
<RecArithE>
     -> <Op1> <ArithT> <RecArithE>
     -> ε
<Op1>
     -> +
     -> -
<ArithT>
     -> <ArithF> <RecArithT>
<RecArithT>
     -> <Op2> <ArithF> <RecArithT>
     -> ε
<Op2>
     -> *
     -> /
<ArithF>
     -> [VarName]
     -> [Number]
     -> (ExprArith)
     -> -<ExprArith>
<If>
     -> IF (<Cond>) THEN [EndLine] <Code> <FactIf>
<FactIf>
     -> ENDIF
     -> ELSE [EndLine] <Code> ENDIF
<CondPrefix>
     -> .NOT.
     -> ε
<Cond>
     -> <CondT> <CondRecE>
<CondRecE>
     -> .OR. <CondT> <CondRecE>
     -> ε
<CondT>
     -> <CondPrefix> <SimpleCond> <CondRecT>
<CondRecT>
     -> .AND. <CondPrefix> <SimpleCond> <CondRecT>
     -> ε
<CondF>
     -> <ExprArith> <Comp> <ExprArith>
<Comp>
     -> .EQ.
     -> .GE.
     -> .GT.
     -> .LE.
     -> .LT.
     -> .NE.
```

```
<Do>
     -> DO [VarName] = [Number], [Number] [EndLine] <Code>
     ENDDO
<Print>
     -> PRINT*, <ExpList>
<Read>
     -> READ*, <VarList>
<ExpList>
     -> <ExprArith>, <FactExprArith>
<FactExprArith>
     -> <ExpList>
     -> ε
```

Target grammar:

Number	Left side	Right side
0.	<All>	<Program> $
1.	<Program>	PROGRAM [ProgName] [EndLine] <Vars> <Code>
2.	<Vars>	INTEGER <VarList> [EndLine]
3.		ε
4.	<VarList>	[VarName], <FactVarList>
5.	<FactVarList>	<VarList>
6.		ε
7.	<Code>	<Instruction> [EndLine] <Code>
8.		ε
9.	<Instruction>	<Assign>
10.		<If>
11.		<Do>
12.		<Print>
13.		<Read>
14.	<Assign>	[VarName] = <ExprArith>
15.	<ExprArith>	<ArithT> <RecArithE>
16.	<RecArithE>	<Op1> <ArithT> <RecArithE>
17.		ε
18.	<Op1>	+
19.		-
20.	<ArithT>	<ArithF> <RecArithT>
21.	<RecArithT>	<Op2> <ArithF> <RecArithT>
22.		ε
23.	<Op2>	*
24.		/
25.	<ArithF>	[VarName]
26.		[Number]
27.		(ExprArith)
28.		-<ExprArith>
29.	<If>	IF (<Cond>) THEN [EndLine] <Code> <FactIf>
30.	<FactIf>	ENDIF
31.		ELSE [EndLine] <Code> ENDIF

32.	<CondPrefix>	.NOT.
33.		ε
34.	<Cond>	<CondT> <CondRecE>
35.	<CondRecE>	.OR. <CondT> <CondRecE>
36.		ε
37.	<CondT>	<CondPrefix> <CondF> <CondRecT>
38.	<CondRecT>	.AND. <CondPrefix> <SimpleCond> <CondRecT>
39.		ε
40.	<CondF>	<ExprArith> <Comp> <ExprArith>
41.	<Comp>	.EQ.
42.		.GE.
43.		.GT.
44.		.LE.
45.		.LT.
46.		.NE.
47.	<Do>	DO [VarName] = [Number], [Number] [EndLine] <Code> ENDDO
48.	<Print>	PRINT*, <ExpList>
49.	<Read>	READ*, <VarList>
50.	<ExpList>	<ExprArith>, <FactExprArith>
51.	<FactExprArith>	<ExpList>
52.		ε

First input	First output
<Program> $	PROGRAM
PROGRAM [ProgName] [EndLine] <Vars> <Code>	PROGRAM
INTEGER <VarList> [EndLine]	INTEGER
[VarName], <FactVarList>	VARNAME
<VarList>	VARNAME
<Instruction> [EndLine] <Code>	VARNAME, IF, DO, PRINT, READ
<Assign>	VARNAME
<If>	IF
<Do>	DO
<Print>	PRINT
<Read>	READ
[VarName] = <ExprArith>	VARNAME
<ArithT> <RecArithE>	VARNAME, NUMBER, (, -
<Op1> <ArithT> <RecArithE>	+, -
+	+
-	-
<ArithF> <RecArithT>	VARNAME, NUMBER, (, -
<Op2> <ArithF> <RecArithT>	*, /
*	*
/	/
[VarName]	VARNAME
[Number]	NUMBER
(ExprArith)	(
-<ExprArith>	-
IF (<Cond>) THEN [EndLine]	IF

<Code> <FactIf>	
ENDIF	ENDIF
ELSE [EndLine] <Code> ENDIF	ELSE
.NOT.	.NOT.
<CondT> <CondRecE>	.NOT.,
.AND. <CondPrefix> <SimpleCond> <CondRecT>	