Université libre de Bruxelles

Project - Part 2

Parser

**Aldar Saranov, Przemyslaw Gasinski**

[Aldar.Saranov@ulb.ac.be](mailto:Aldar.Saranov@ulb.ac.be)

[Przemyslaw.Gasinski@ulb.ac.be](mailto: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]  ->  <VarList>  -> [VarName], <VarList>  -> [VarName]  <Code>  -> <Instruction> [EndLine] <Code>  ->  <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]  ->  <VarList>  -> [VarName], <VarList>  -> [VarName]  <Code>  -> <Instruction> [EndLine] <Code>  ->  <Instruction>  -> <Assign>  -> <If>  -> <Do>  -> <Print>  -> <Read>  <Assign>  -> [VarName] = <ExprArith>  <ExprArith>  -> [VarName] <ExprArithRec>  -> [Number] <ExprArithRec>  -> (<ExprArith>) <ExprArithRec>  -> -<ExprArith> <ExprArithRec>  <ExprArithRec>  -> <Op> <ExprArith> <ExprArithRec>  ->  <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>  ->  <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]  ->  <VarList>  -> [VarName], <FactVarList>  <FactVarList>  -> <VarList>  ->  <Code>  -> <Instruction> [EndLine] <Code>  ->  <Instruction>  -> <Assign>  -> <If>  -> <Do>  -> <Print>  -> <Read>  <Assign>  -> [VarName] = <ExprArith>  <ExprArith>  -> <FactExprArith> <ExprArithRec>  <FactExprArith>  -> [VarName]  -> [Number]  -> (<ExprArith>)  -> -<ExprArith>  <ExprArithRec>  -> <Op> <ExprArith> <ExprArithRec>  ->  <Op>  -> +  -> -  -> \*  -> /  <If>  -> IF (<Cond>) THEN [EndLine] <Code> <FactIf>  <FactIf>  -> ENDIF  -> ELSE [EndLine] <Code> ENDIF  <Cond>  -> <CondPrefix> <SimpleCond> <CondRec>  <CondPrefix>  -> .NOT.  ->  <CondRec>  -> <BinOp> <Cond> <CondRec>  ->  <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>  -> |

Making non-ambiguous

|  |
| --- |
| <Program>  -> PROGRAM [ProgName] [EndLine] <Vars> <Code> END  <Vars>  -> INTEGER <VarList> [EndLine]  ->  <VarList>  -> [VarName], <FactVarList>  <FactVarList>  -> <VarList>  ->  <Code>  -> <Instruction> [EndLine] <Code>  ->  <Instruction>  -> <Assign>  -> <If>  -> <Do>  -> <Print>  -> <Read>  <Assign>  -> [VarName] = <ExprArith>  <ExprArith>  -> <ArithT> <RecArithE>  <RecArithE>  -> <Op1> <ArithT> <RecArithE>  ->  <Op1>  -> +  -> -  <ArithT>  -> <ArithF> <RecArithT>  <RecArithT>  -> <Op2> <ArithF> <RecArithT>  ->  <Op2>  -> \*  -> /  <ArithF>  -> [VarName]  -> [Number]  -> (ExprArith)  -> -<ExprArith>  <If>  -> IF (<Cond>) THEN [EndLine] <Code> <FactIf>  <FactIf>  -> ENDIF  -> ELSE [EndLine] <Code> ENDIF  <CondPrefix>  -> .NOT.  ->  <Cond>  -> <CondT> <CondRecE>  <CondRecE>  -> .OR. <CondT> <CondRecE>  ->  <CondT>  -> <CondPrefix> <SimpleCond> <CondRecT>  <CondRecT>  -> .AND. <CondPrefix> <CondF> <CondRecT>  ->  <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>  -> |

Obtained grammar:

|  |  |  |
| --- | --- | --- |
| Number | Left side | Right side |
| 0. | <All> | <Program> $ |
|  | <Program> | PROGRAM [ProgName] [EndLine] <Vars> <Code> END |
|  | <Vars> | INTEGER <VarList> [EndLine] |
|  |  |  |
|  | <VarList> | [VarName], <FactVarList> |
|  | <FactVarList> | <VarList> |
|  |  |  |
|  | <Code> | <Instruction> [EndLine] <Code> |
|  |  |  |
|  | <Instruction> | <Assign> |
|  |  | <If> |
|  |  | <Do> |
|  |  | <Print> |
|  |  | <Read> |
|  | <Assign> | [VarName] = <ExprArith> |
|  | <ExprArith> | <ArithT> <RecArithE> |
|  | <RecArithE> | <Op1> <ArithT> <RecArithE> |
|  |  |  |
|  | <Op1> | + |
|  |  | - |
|  | <ArithT> | <ArithF> <RecArithT> |
|  | <RecArithT> | <Op2> <ArithF> <RecArithT> |
|  |  |  |
|  | <Op2> | \* |
|  |  | / |
|  | <ArithF> | [VarName] |
|  |  | [Number] |
|  |  | (ExprArith) |
|  |  | -<ExprArith> |
|  | <If> | IF (<Cond>) THEN [EndLine] <Code> <FactIf> |
|  | <FactIf> | ENDIF |
|  |  | ELSE [EndLine] <Code> ENDIF |
|  | <CondPrefix> | .NOT. |
|  |  |  |
|  | <Cond> | <CondT> <CondRecE> |
|  | <CondRecE> | .OR. <CondT> <CondRecE> |
|  |  |  |
|  | <CondT> | <CondPrefix> <CondF> <CondRecT> |
|  | <CondRecT> | .AND. <CondPrefix> <CondF> <CondRecT> |
|  |  |  |
|  | <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> |
|  |  |  |

|  |  |
| --- | --- |
| 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] <Code> <FactIf> | IF |
| ENDIF | ENDIF |
| ELSE [EndLine] <Code> ENDIF | ELSE |
| .NOT. | .NOT. |
| <CondT> <CondRecE> | .NOT., VARNAME, NUMBER, (, - |
| .AND. <CondPrefix> <SimpleCond> <CondRecT> | .AND. |
| <ExprArith> <Comp> <ExprArith> | VARNAME, NUMBER, (, - |
| .EQ. | .EQ. |
| .GE. | .GE. |
| .GT. | .GT. |
| .LE. | .LE. |
| .LT. | .LT. |
| .NE. | .NE. |
| DO [VarName] = [Number], [Number] [EndLine] <Code> ENDDO | DO |
| PRINT\*, <ExpList> | PRINT\*, |
| READ\*, <VarList> | READ\*, |
| <ExprArith> <FactExprArith> | VARNAME, NUMBER, (, - |
| , <ExpList> | , |

|  |  |  |
| --- | --- | --- |
| Input | First | Follow |
| <All> | PROGRAM |  |
| <Program> | PROGRAM | $ |
| <Vars> | , INTEGER | $, VARNAME, IF,DO, PRINT, READ |
| <VarList> | VARNAME | ENDLINE |
| <FactVarList> | , VARNAME | ENDLINE |
| <Code> | , VARNAME, IF,DO, PRINT, READ | $, ENDIF, ELSE, ENDDO |
| <Instruction> | VARNAME, IF,DO, PRINT, READ | ENDLINE |
| <Assign> | VARNAME | ENDLINE |
| <ExprArith> | VARNAME, NUMBER, (, - | ENDLINE, \*, /, +, -, .EQ., .GE., .GT., .LE, .LT., .NE., .AND., ), .OR., COMMA |
| <RecArithE> | , +, - | ENDLINE, \*, /, +, -, .EQ., .GE., .GT., .LE, .LT., .NE., .AND., ), .OR., COMMA |
| <Op1> | +, - | +, - |
| <ArithT> | VARNAME, NUMBER, (, - | ENDLINE, \*, /, +, -, .EQ., .GE., .GT., .LE, .LT., .NE., .AND., ), .OR., COMMA |
| <RecArithT> | , \*, / | ENDLINE, \*, /, +, -, .EQ., .GE., .GT., .LE, .LT., .NE., .AND., ), .OR., COMMA |
| <Op2> | \*, / | \*, / |
| <ArithF> | VARNAME, NUMBER, (, - | ENDLINE, \*, /, +, -, .EQ., .GE., .GT., .LE, .LT., .NE., .AND., ), .OR., COMMA |
| <If> | IF | ENDLINE |
| <FactIf> | ENDIF, ELSE | ENDLINE |
| <CondPrefix> | , .NOT. | VARNAME, NUMBER, (, - |
| <Cond> | .NOT., VARNAME, NUMBER, (, - | ) |
| <CondRecE> | , .OR. | ) |
| <CondT> | .NOT., VARNAME, NUMBER, (, - | ), .OR. |
| <CondRecT> | , .AND. | ), .OR. |
| <CondF> | VARNAME, NUMBER, (, - | .AND.,  ), .OR. |
| <Comp> | .EQ., .GE., .GT., .LE, .LT., .NE. | VARNAME, NUMBER, (, - |
| <Do> | DO | ENDLINE |
| <Print> | PRINT | ENDLINE |
| <Read> | READ | ENDLINE |
| <ExpList> | VARNAME, NUMBER, (, - | ENDLINE |
| <FactExprArith> | , COMMA | ENDLINE |