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>  -> + <ExprArith> <RecArithE>  -> - <ExprArith> <RecArithE>  ->  <ArithT>  -> <ArithF> <RecArithT>  <RecArithT>  -> \* <ArithT> <RecArithT>  -> / <ArithT> <RecArithT>  ->  <ArithF>  -> [VarName]  -> [Number]  -> (ExprArith)  -> -<ExprArith>  <If>  -> IF (<Cond>) THEN [EndLine] <Code> <FactIf>  <FactIf>  -> ENDIF  -> ELSE [EndLine] <Code> ENDIF  <CondPrefix>  -> .NOT.  ->  <Cond>  -> <CondPrefix> <CondT> <CondRecE>  <CondRecE>  -> .OR. <Cond> <CondRecE>  ->  <CondT>  -> <CondPrefix> <SimpleCond> <CondRecT>  <CondRecT>  -> .AND. <Cond> <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>  -> |