Esterel Grammar

```
/**
 *
 * Esterel Grammar for ANTLR based on the grammar in the Esterel Primer, v91
 * Also includes deprecated syntax: see commentary below.
 *
 * Author: Stephen A. Edwards
 *         sedwards@cs.columbia.edu
 *
 * This generates a generic AST using ANTLR&#039;s built-in facilities for AST
 * synthesis.
 *
 * Change Log
 */
Definition of parser EsterelParser, which is a subclass of LLkParser.


file
  :  ( module )+ EOF
  ;


module
  :  &quot;module&quot; moduleIdentifier COLON declarations statement
    (  &quot;end&quot; &quot;module&quot;
    |  PERIOD
    )
  ;


moduleIdentifier
  :  ID
  ;
```

```
declarations
  :  ( interfaceDecls )*
  ;


statement
  :  sequence ( PARALLEL sequence )*
  ;


interfaceDecls
  :  typeDecls
  |  constantDecls
  |  functionDecls
  |  procedureDecls
  |  taskDecls
  |  interfacesignalDecls
  |  sensorDecls
  |  relationDecls
  ;


typeDecls
  :  &quot;type&quot; typeIdentifier ( COMMA typeIdentifier )* SEMICOLON
  ;


constantDecls
  :  &quot;constant&quot; constantDecl ( COMMA constantDecl )* SEMICOLON
  ;


functionDecls
  :  &quot;function&quot; functionDecl ( COMMA functionDecl )* SEMICOLON
  ;


procedureDecls
  :  &quot;procedure&quot; procedureDecl ( COMMA procedureDecl )* SEMICOLON
  ;


taskDecls
  :  &quot;task&quot; taskDecl ( COMMA taskDecl )* SEMICOLON
  ;
```

```
/**
    The grammar allows full expressions in the initializers
    but only constants are permitted in interface signals.
*/
interfacesignalDecls
  :  &quot;input&quot; signalDecl ( COMMA signalDecl )* SEMICOLON
  |  &quot;output&quot; signalDecl ( COMMA signalDecl )* SEMICOLON
  |  &quot;inputoutput&quot; signalDecl ( COMMA signalDecl )* SEMICOLON
  |  &quot;return&quot; signalDecl ( COMMA signalDecl )* SEMICOLON
  ;


sensorDecls
  :  &quot;sensor&quot; sensorDecl ( COMMA sensorDecl )* SEMICOLON
  ;


relationDecls
  :  &quot;relation&quot; relationDecl ( COMMA relationDecl )* SEMICOLON
  ;


typeIdentifier
  :  ID
  ;


constantDecl
  :  (  constantIdentifier
     (  constantInitializer
     |
     )
    |  identifierList
    )
    COLON typeIdentifier
  ;


constantIdentifier
  :  ID
  ;


constantInitializer
  :  EQUALS constantAtom
```

```
  ;


identifierList
  :  ID COMMA ID ( COMMA ID )*
  ;


constantAtom
  :  constantLiteral
  |  signedNumber
  ;


functionDecl
  :  functionIdentifier optTypeIdentifierList COLON typeIdentifier
  ;


functionIdentifier
  :  ID
  ;


optTypeIdentifierList
  :  LPAREN
    (  typeIdentifier ( COMMA typeIdentifier )*
    |
    )
    RPAREN
  ;


procedureDecl
  :  procedureIdentifier optTypeIdentifierList optTypeIdentifierList
  ;


procedureIdentifier
  :  ID
  ;


taskDecl
  :  taskIdentifier optTypeIdentifierList optTypeIdentifierList
  ;
```

```
taskIdentifier
  :  ID
  ;


signalDecl
  :  signalIdentifier
    (
    |  (  signalInitializer
      |
      )
      COLON channelType
    |  LPAREN channelType RPAREN
    )

  ;


signalDeclList
  :  signalDecl ( COMMA signalDecl )*
  ;


signalIdentifier
  :  ID
  ;


signalInitializer
  :  COLEQUALS expression
  ;


channelType
  :  typeIdentifier
  |  &quot;combine&quot; typeIdentifier &quot;with&quot;
    (  functionIdentifier
    |  predefinedCombineFunction
    )
  ;


/*****************************************************************************
 *
```

```
 * Expressions
 *
 ************************************************************************/
expression
  :  orexpr
  ;


predefinedCombineFunction
  :  PLUS
  |  STAR
  |  &quot;or&quot;
  |  &quot;and&quot;
  ;


sensorDecl
  :  sensorIdentifier
    (  COLON typeIdentifier
    |  LPAREN typeIdentifier RPAREN
    )

  ;


sensorIdentifier
  :  ID
  ;


relationDecl
  :  implicationDecl
  |  exclusionDecl
  ;


implicationDecl
  :  signalIdentifier IMPLIES signalIdentifier
  ;


exclusionDecl
  :  signalIdentifier POUND signalIdentifier ( POUND signalIdentifier )*
  ;
```

```
orexpr
  :  andexpr ( &quot;or&quot; andexpr )*
  ;


andexpr
  :  notexpr ( &quot;and&quot; notexpr )*
  ;


notexpr
  :  &quot;not&quot; cmpexpr
  |  cmpexpr
  ;


cmpexpr
  :  addexpr ( (  EQUALS
       |  NEQUAL
       |  LESSTHAN
       |  GREATERTHAN
       |  LEQUAL
       |  GEQUAL
       )
       addexpr )*
  ;


addexpr
  :  mulexpr ( (  PLUS
       |  DASH
       )
       mulexpr )*
  ;


mulexpr
  :  unaryexpr ( (  STAR
       |  SLASH
       |  &quot;mod&quot;
       )
       unaryexpr )*
  ;


unaryexpr
```

```
  :  DASH unaryexpr
  |  LPAREN expression RPAREN
  |  QUESTION signalIdentifier
  |  &quot;pre&quot; LPAREN QUESTION signalIdentifier RPAREN
  |  DQUESTION trapIdentifier
  |  functionCall
  |  constant
  ;


trapIdentifier
  :  ID
  ;


functionCall
  :  functionIdentifier LPAREN
    (  expression ( COMMA expression )*
    |
    )
    RPAREN
  ;


constant
  :  constantLiteral
  |  unsignedNumber
  ;


constantLiteral
  :  constantIdentifier
  |  &quot;true&quot;
  |  &quot;false&quot;
  |  stringConstant
  ;


unsignedNumber
  :  Integer
  |  FloatConst
  |  DoubleConst
  ;


stringConstant
```

```
  :  StringConstant
  ;


signedNumber
  :  unsignedNumber
  |  DASH unsignedNumber
  ;


signalExpression
  :  sandexpr ( &quot;or&quot; sandexpr )*
  ;


sandexpr
  :  sunaryexpr ( &quot;and&quot; sunaryexpr )*
  ;


sunaryexpr
  :  signalIdentifier
  |  &quot;pre&quot; LPAREN signalIdentifier RPAREN
  |  &quot;not&quot; sunaryexpr
  |  LPAREN signalExpression RPAREN
  ;


delayExpression
  :  (  delayPair
    |  bracketedSignalExpression
    |  &quot;immediate&quot; bracketedSignalExpression
    )
  ;


bracketedSignalExpression
  :  signalIdentifier
  |  LBRACKET signalExpression RBRACKET
  ;


delayPair
  :  expression bracketedSignalExpression
  ;
```

```
sequence
  :  atomicStatement ( SEMICOLON atomicStatement )*
    (  SEMICOLON
    |
    )

  ;


atomicStatement
  :  &quot;nothing&quot;
  |  &quot;pause&quot;
  |  &quot;halt&quot;
  |  emit
  |  sustain
  |  assignment
  |  procedureCall
  |  present
  |  ifstatement
  |  loop
  |  repeat
  |  abort
  |  await
  |  every
  |  suspend
  |  trap
  |  exit
  |  exec
  |  localvariableDecl
  |  localSignalDecl
  |  runModule
  |  LBRACKET statement RBRACKET
  |  doStatements
  ;


emit
  :  &quot;emit&quot; signalIdentifier
    (  LPAREN expression RPAREN
    |
    )
  ;


sustain
```

```
  :  &quot;sustain&quot; signalIdentifier
    (  LPAREN expression RPAREN
    |
    )
  ;


assignment
  :  variableIdentifier COLEQUALS expression
  ;


procedureCall
  :  &quot;call&quot; procedureIdentifier refArgs valueArgs
  ;


present
  :  &quot;present&quot;
    (  presentThenPart
    |  ( presentCase )+
    )
    (  elsePart
    |
    )
    &quot;end&quot;
    (  &quot;present&quot;
    |
    )
  ;


ifstatement
  :  &quot;if&quot; expression
    (  thenPart
    |
    )
    ( elsif )*
    (  elsePart
    |
    )
    &quot;end&quot;
    (  &quot;if&quot;
    |
    )
  ;
```

```
loop
  :  &quot;loop&quot; statement
    (  &quot;end&quot;
      (  &quot;loop&quot;
      |
      )
    |  &quot;each&quot; delayExpression
    )
  ;


repeat
  :  (  &quot;positive&quot;
    |
    )
    &quot;repeat&quot; expression &quot;times&quot; statement &quot;end&quot;
    (  &quot;repeat&quot;
    |
    )
  ;


abort
  :  &quot;abort&quot; statement &quot;when&quot;
    (  abortOneCaseStrong
    |  ( acase )+ &quot;end&quot;
      (  &quot;abort&quot;
      |
      )
    )
  |  &quot;weak&quot; &quot;abort&quot; statement &quot;when&quot;
    (  abortOneCaseWeak
    |  ( acase )+ &quot;end&quot;
      (  (  &quot;weak&quot;
        |
        )
        &quot;abort&quot;
      |
      )
    )
  ;


await
```

```
  :  &quot;await&quot;
    (  awaitOneCase
    |  ( acase )+ &quot;end&quot;
      (  &quot;await&quot;
      |
      )
    )
  ;


every
  :  &quot;every&quot; delayExpression &quot;do&quot; statement &quot;end&quot;
    (  &quot;every&quot;
    |
    )
  ;


suspend
  :  &quot;suspend&quot; statement &quot;when&quot; delayExpression
  ;


trap
  :  &quot;trap&quot; trapDeclList &quot;in&quot; statement ( trapHandler )* &quot;end&quot;
    (  &quot;trap&quot;
    |
    )
  ;


exit
  :  &quot;exit&quot; trapIdentifier
    (  LPAREN expression RPAREN
    |
    )
  ;


exec
  :  &quot;exec&quot; execOneCase
  |  &quot;exec&quot; ( execCase )+ &quot;end&quot;
    (  &quot;exec&quot;
    |
    )
  ;
```

```
localvariableDecl
  :  &quot;var&quot; variableDeclList &quot;in&quot; statement &quot;end&quot;
    (  &quot;var&quot;
    |
    )
  ;


localSignalDecl
  :  &quot;signal&quot; signalDeclList &quot;in&quot; statement &quot;end&quot;
    (  &quot;signal&quot;
    |
    )
  ;


runModule
  :  (  &quot;run&quot;
    |  &quot;copymodule&quot;
    )
    moduleIdentifier
    (  SLASH moduleIdentifier
    |
    )
    (  LBRACKET renaming ( SEMICOLON renaming )* RBRACKET
    |
    )

  ;


doStatements
  :  &quot;do&quot; statement
    (  &quot;watching&quot; delayExpression
      (  &quot;timeout&quot; statement &quot;end&quot;
        (  &quot;timeout&quot;
        |
        )
      |
      )

    |  &quot;upto&quot; delayExpression
    )
  ;
```

```
variableIdentifier
  :  ID
  ;


refArgs
  :  LPAREN
    (  variableIdentifier ( COMMA variableIdentifier )*
    |
    )
    RPAREN
  ;


valueArgs
  :  LPAREN
    (  expression ( COMMA expression )*
    |
    )
    RPAREN
  ;


presentThenPart
  :  presentEvent
    (  &quot;then&quot; statement
    |
    )

  ;


presentCase
  :  &quot;case&quot; presentEvent
    (  &quot;do&quot; statement
    |
    )

  ;


elsePart
  :  &quot;else&quot; statement
  ;
```

```
presentEvent
  :  (  signalExpression
    |  LBRACKET signalExpression RBRACKET
    )
  ;


thenPart
  :  &quot;then&quot; statement
  ;


elsif
  :  &quot;elsif&quot; expression &quot;then&quot; statement
  ;


abortOneCaseStrong
  :  delayExpression
    (  &quot;do&quot; statement &quot;end&quot;
      (  &quot;abort&quot;
      |
      )
    |
    )

  ;


acase
  :  &quot;case&quot; delayExpression
    (  &quot;do&quot; statement
    |
    )

  ;


abortOneCaseWeak
  :  delayExpression
    (  &quot;do&quot; statement &quot;end&quot;
      (  (  &quot;weak&quot;
        |
        )
```

```
        &quot;abort&quot;
      |
      )
    |
    )

  ;


awaitOneCase
  :  delayExpression
    (  &quot;do&quot; statement &quot;end&quot;
      (  &quot;await&quot;
      |
      )
    |
    )

  ;


trapDeclList
  :  trapDecl ( COMMA trapDecl )*
  ;


trapHandler
  :  &quot;handle&quot; trapEvent &quot;do&quot; statement
  ;


trapDecl
  :  trapIdentifier
    (  (  trapInitializer
      |
      )
      COLON channelType
    |
    )

  ;


trapInitializer
  :  COLEQUALS expression
  ;
```

```
trapEvent
  :  eand ( &quot;or&quot; eand )*
  ;


eand
  :  eunary ( &quot;and&quot; eunary )*
  ;


eunary
  :  trapIdentifier
  |  LPAREN trapEvent RPAREN
  |  &quot;not&quot; eunary
  ;


execOneCase
  :  taskIdentifier refArgs valueArgs &quot;return&quot; signalIdentifier
    (  &quot;do&quot; statement &quot;end&quot;
      (  &quot;exec&quot;
      |
      )
    |
    )

  ;


execCase
  :  &quot;case&quot; taskIdentifier refArgs valueArgs &quot;return&quot; signalIdentifier
    (  &quot;do&quot; statement
    |
    )

  ;


variableDeclList
  :  variableDecl ( COMMA variableDecl )*
  ;


variableDecl
```

```
  :  (  variableIdentifier
      (  variableInitializer
      |
      )
    |  identifierList
    )
    COLON typeIdentifier
  ;


variableInitializer
  :  COLEQUALS expression
  ;


renaming
  :  &quot;type&quot; typeRenaming ( COMMA typeRenaming )*
  |  &quot;constant&quot; constantRenaming ( COMMA constantRenaming )*
  |  &quot;function&quot; functionRenaming ( COMMA functionRenaming )*
  |  &quot;procedure&quot; procedureRenaming ( COMMA procedureRenaming )*
  |  &quot;task&quot; taskRenaming ( COMMA taskRenaming )*
  |  &quot;signal&quot; signalRenaming ( COMMA signalRenaming )*
  ;


typeRenaming
  :  typeIdentifier SLASH typeIdentifier
  ;


constantRenaming
  :  constantAtom SLASH constantIdentifier
  ;


functionRenaming
  :  (  functionIdentifier SLASH functionIdentifier
    |  predefinedFunction SLASH functionIdentifier
    )

  ;


procedureRenaming
  :  procedureIdentifier SLASH procedureIdentifier
  ;
```

```
taskRenaming
  :  taskIdentifier SLASH taskIdentifier
  ;


signalRenaming
  :  signalIdentifier SLASH signalIdentifier
  ;


predefinedFunction
  :  &quot;and&quot;
  |  &quot;or&quot;
  |  &quot;not&quot;
  |  PLUS
  |  DASH
  |  STAR
  |  SLASH
  |  &quot;mod&quot;
  |  LESSTHAN
  |  GREATERTHAN
  |  LEQUAL
  |  GEQUAL
  |  NEQUAL
  |  EQUALS
  ;
```