

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's built-in facilities for AST
 * synthesis.
 *
 * Change Log
 */
Definition of parser EsterelParser, which is a subclass of LLkParser.

file
: ( module )+ EOF
;

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

moduleIdentifier
: ID
;
```

```

declarations
: ( interfaceDecls )*
;

statement
: sequence ( PARALLEL sequence )*
;

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

typeDecls
: "type" typeIdentifier ( COMMA typeIdentifier )* SEMICOLON
;

constantDecls
: "constant" constantDecl ( COMMA constantDecl )* SEMICOLON
;

functionDecls
: "function" functionDecl ( COMMA functionDecl )* SEMICOLON
;

procedureDecls
: "procedure" procedureDecl ( COMMA procedureDecl )* SEMICOLON
;

taskDecls
: "task" taskDecl ( COMMA taskDecl )* SEMICOLON
;

```

```

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

sensorDecls
: "sensor" sensorDecl ( COMMA sensorDecl )* SEMICOLON
;

relationDecls
: "relation" 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
  | "combine" typeIdentifier "with"
  ( functionIdentifier
    | predefinedCombineFunction
  )
;

/*****
*
```

```

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

predefinedCombineFunction
: PLUS
| STAR
| "or"
| "and"
;

sensorDecl
: sensorIdentifier
( COLON typeIdentifier
| LPAREN typeIdentifier RPAREN
)

;

sensorIdentifier
: ID
;

relationDecl
: implicationDecl
| exclusionDecl
;

implicationDecl
: signalIdentifier IMPLIES signalIdentifier
;

exclusionDecl
: signalIdentifier POUND signalIdentifier ( POUND signalIdentifier )*
;

```

```

orexpr
:  andexpr ( "or" andexpr )*
;

andexpr
:  notexpr ( "and" notexpr )*
;

notexpr
:  "not" cmpexpr
|  cmpexpr
;

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

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

mulexpr
:  unaryexpr ( ( STAR
|  SLASH
|  "mod"
)
unaryexpr )*
;

unaryexpr

```

```

: DASH unaryexpr
| LPAREN expression RPAREN
| QUESTION signalIdentifier
| "pre" LPAREN QUESTION signalIdentifier RPAREN
| DQUESTION trapIdentifier
| functionCall
| constant
;

trapIdentifier
: ID
;

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

constant
: constantLiteral
| unsignedNumber
;

constantLiteral
: constantIdentifier
| "true"
| "false"
| stringConstant
;

unsignedNumber
: Integer
| FloatConst
| DoubleConst
;

stringConstant

```



```

: StringConstant
;

signedNumber
: unsignedNumber
| DASH unsignedNumber
;

signalExpression
: sandexpr ( "or" sandexpr )*
;

sandexpr
: sunaryexpr ( "and" sunaryexpr )*
;

sunaryexpr
: signalIdentifier
| "pre" LPAREN signalIdentifier RPAREN
| "not" sunaryexpr
| LPAREN signalExpression RPAREN
;

delayExpression
: ( delayPair
  | bracketedSignalExpression
  | "immediate" bracketedSignalExpression
  )
;

bracketedSignalExpression
: signalIdentifier
| LBRACKET signalExpression RBRACKET
;

delayPair
: expression bracketedSignalExpression
;

```

```

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

;

```

```

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

;

```

```

emit
:  "emit" signalIdentifier
  ( LPAREN expression RPAREN
  |
  )

;

```

```

sustain

```

```

: "sustain" signalIdentifier
( LPAREN expression RPAREN
|
)
;

assignment
: variableIdentifier COLEQUALS expression
;

procedureCall
: "call" procedureIdentifier refArgs valueArgs
;

present
: "present"
( presentThenPart
| ( presentCase )+
)
( elsePart
|
)
"end"
( "present"
|
)
;

ifstatement
: "if" expression
( thenPart
|
)
( elsif )*
( elsePart
|
)
"end"
( "if"
|
)
;

```

```

loop
  : "loop" statement
    ( "end"
      ( "loop"
        |
      )
      | "each" delayExpression
    )
  ;

repeat
  : ( "positive"
    |
  )
  "repeat" expression "times" statement "end"
  ( "repeat"
    |
  )
  ;

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

await

```

```

: "await"
(  awaitOneCase
|  ( acase )+ "end"
  (  "await"
    |
  )
)
;

every
: "every" delayExpression "do" statement "end"
(  "every"
  |
)
;

suspend
: "suspend" statement "when" delayExpression
;

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

exit
: "exit" trapIdentifier
(  LPAREN expression RPAREN
  |
)
;

exec
: "exec" execOneCase
|  "exec" ( execCase )+ "end"
(  "exec"
  |
)
;

```

```

localvariableDecl
: "var" variableDeclList "in" statement "end"
  ( "var"
    |
  )
;

localSignalDecl
: "signal" signalDeclList "in" statement "end"
  ( "signal"
    |
  )
;

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

doStatements
: "do" statement
  ( "watching" delayExpression
    ( "timeout" statement "end"
      ( "timeout"
        |
      )
    )
    |
  )
  | "upto" delayExpression
  )
;

```

```
variableIdentifier
: ID
;
```

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

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

```
presentThenPart
: presentEvent
  ( "then" statement
  |
  )
;
```

```
presentCase
: "case" presentEvent
  ( "do" statement
  |
  )
;
```

```
elsePart
: "else" statement
;
```

```

presentEvent
: (  signalExpression
  |  LBRACKET signalExpression RBRACKET
)
;

```

```

thenPart
: "then" statement
;

```

```

elsif
: "elsif" expression "then" statement
;

```

```

abortOneCaseStrong
: delayExpression
  ( "do" statement "end"
    ( "abort"
      |
    )
  |
  )
;

```

```

acase
: "case" delayExpression
  ( "do" statement
    |
  )
;

```

```

abortOneCaseWeak
: delayExpression
  ( "do" statement "end"
    ( ( "weak"
      |
    )
  )
;

```



```

        "abort"
        |
      )
    |
  )

;

awaitOneCase
: delayExpression
  ( "do" statement "end"
    ( "await"
      |
    )
  |
)

;

trapDeclList
: trapDecl ( COMMA trapDecl )*
;

trapHandler
: "handle" trapEvent "do" statement
;

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

;

trapInitializer
: COLEQUALS expression
;

```

```

trapEvent
: eand ( "or" eand )*
;

eand
: eunary ( "and" eunary )*
;

eunary
: trapIdentifier
| LPAREN trapEvent RPAREN
| "not" eunary
;

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

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

variableDeclList
: variableDecls ( COMMA variableDecls )*
;

variableDecls

```

```

: variableDeclList2 COLON typeIdentifier
;

variableDeclList2
: ( variableDecl ( COMMA variableDecl )* )
;

variableDecl
: variableIdentifier
  ( variableInitializer
    |
  )
;

variableInitializer
: COLEQUALS expression
;

renaming
: "type" typeRenaming ( COMMA typeRenaming )*
| "constant" constantRenaming ( COMMA constantRenaming )*
| "function" functionRenaming ( COMMA functionRenaming )*
| "procedure" procedureRenaming ( COMMA procedureRenaming )*
| "task" taskRenaming ( COMMA taskRenaming )*
| "signal" 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
: "and"
| "or"
| "not"
| PLUS
| DASH
| STAR
| SLASH
| "mod"
| LESSTHAN
| GREATERTHAN
| LEQUAL
| GEQUAL
| NEQUAL
| EQUALS
;

```