Syntax of *Lustre** for the Open Source L2C Compiler

L2C team, System Software and Software Engineering Laboratory
Department of Computer Science and Technology, Tsinghua University

August 29, 2020

1 Program

⟨*program*⟩ ::= { ⟨*decls*⟩ }

⟨*decls*⟩ ::= ⟨*type_decl*⟩
| ⟨*const_decl*⟩
| [**main**] ⟨*node_decl*⟩

2 Type Block

⟨*type_decl*⟩ ::= **type** ⟨*typeDeclList*⟩

⟨*typeDeclList*⟩ ::= ⟨*one_type_decl*⟩ ';' { ⟨*one_type_decl*⟩ ';' }

⟨*one_type_decl*⟩ ::= *IDENT* '=' ⟨*kind*⟩

⟨*kind*⟩ ::= *IDENT* //type identifiers
| **bool** //bool types
| **short** //integer types, signed 16 bits
| **ushort** //integer types, unsigned 16 bits
| **int** //integer types, signed 32 bits
| **uint** //integer types, unsigned 32 bits
| **float** //floating-point types, 32 bits
| **real** //floating-point types, 64 bits
| **char** //char types
| ⟨*kind*⟩ '^ ' *INTEGER* //array types
| '{' ⟨*field_decl*⟩ { ',' ⟨*field_decl*⟩ } '}' //struct types
| **enum** '{' *IDENT* { ',' *IDENT* } '}' //enum types

⟨*field_decl*⟩ ::= ⟨*id_list*⟩ ':' ⟨*kind*⟩

⟨*id_list*⟩ ::= *IDENT* { ',' *IDENT* }

3 Const Block

⟨*const_decl*⟩ ::= **const** ⟨*constDeclList*⟩

⟨*constDeclList*⟩ ::= ⟨*one_const_decl*⟩ ';' { ⟨*one_const_decl*⟩ ';' }

⟨*one_const_decl*⟩ ::= *IDENT* ':' ⟨*kind*⟩ '=' ⟨*const_expr*⟩

⟨*const_expr*⟩ ::= ⟨*atom*⟩
| ⟨*unop*⟩ ⟨*const_expr*⟩ //unary const_expressions
| ⟨*const_expr*⟩ ⟨*binop*⟩ ⟨*const_expr*⟩ //binary const_expressions
| '(' ⟨*const_expr*⟩ ')'
| '[' ⟨*const_expr*⟩ { ',' ⟨*const_expr*⟩ } ']' //array constants
| '{' ⟨*field_const_decl*⟩ { ',' ⟨*field_const_decl*⟩ } '}' //struct constants

⟨*atom*⟩ ::= *IDENT* // a constant identifier or an enum constants
| **true** //bool constant
| **false** //bool constant
| *SHORT* //integer constants, signed 16 bits
| *USHORT* //integer constants, unsigned 16 bits
| *INTEGER* //integer constants, signed 32 bits
| *UINT* //integer constants, unsigned 32 bits
| *FLOAT* //floating-point constants, 32 bits
| *REAL* //floating-point constants, 64 bits
| *CHAR* //char constants

⟨*field_const_decl*⟩ ::= ⟨*id_list*⟩ ':' ⟨*const_expr*⟩

4 Node Block

⟨*node_decl*⟩ ::= [**main**] ⟨*funcType*⟩ *IDENT* '(' ⟨*decls*⟩ ')' **returns** '(' ⟨*decls*⟩ ')' ⟨*body*⟩
//node or function declareation

⟨*funcType*⟩ ::= **node** | **function**

⟨*decls*⟩ ::= [⟨*var_decl*⟩ { ';' ⟨*var_decl*⟩ }]

⟨*var_decl*⟩ ::= *IDENT* { ',' *IDENT* } ':' ⟨*kind*⟩ [**when** ⟨*clock_expr*⟩]

⟨*clock_expr*⟩ ::= *IDENT*
| **not** *IDENT*
| **not** '(' *IDENT* ')'
| *IDENT* '(' *IDENT* ')' //an enum value [1]

⟨*body*⟩ ::= [**var** ⟨*decls*⟩] **let** ⟨*equations*⟩ **tel** [';']

⟨*equations*⟩ ::= ⟨*equation*⟩ { ⟨*equation*⟩ }

⟨*equation*⟩ ::= ⟨*lhs*⟩ '=' ⟨*expr*⟩ ';'

[1] A text in red color shows some candidate consideration in the future.

⟨*lhs*⟩ ::= ⟨*lhs_id*⟩ { ',' ⟨*lhs_id*⟩ }

⟨*lhs_id*⟩ ::= *IDENT*

⟨*expr*⟩ ::= ⟨*atom*⟩ //atom expressions
| ⟨*expr_list*⟩ //expression list
| ⟨*tempo_expr*⟩ //temporal expressions
| ⟨*unop*⟩ ⟨*expr*⟩ //unary expressions
| ⟨*expr*⟩ ⟨*binop*⟩ ⟨*expr*⟩ //binary expressions
| ⟨*nary*⟩ ⟨*expr*⟩
| **if** ⟨*expr*⟩ **then** ⟨*expr*⟩ **else** ⟨*expr*⟩ //conditional expressions
| **case** ⟨*expr*⟩ **of** '(' '|' ⟨*pattern_expr*⟩ { '|' ⟨*pattern_expr*⟩ } ')' //case expressions
| **boolred** '≪' *INTEGER* ',' *INTEGER* '≫' ⟨*expr*⟩ //boolred expressions
| ⟨*struct_expr*⟩ //struct expressions
| ⟨*array_expr*⟩ //array expressions
| ⟨*apply_expr*⟩ //apply expressions

⟨*expr_list*⟩ ::= '(' ⟨*expr*⟩ { ',' ⟨*expr*⟩ } ')' //expression list

⟨*tempo_expr*⟩ ::= **pre** ⟨*expr*⟩ //pre expressions
| **fby** '(' ⟨*expr*⟩ ';' *INTEGER* ';' ⟨*expr*⟩ ')' //fby expressions
| ⟨*expr*⟩ **fby** ⟨*expr*⟩ //fby expressions
| ⟨*expr*⟩ '->' ⟨*expr*⟩ //arrow expressions
| ⟨*expr*⟩ **when** ⟨*clock_expr*⟩ //when expressions
| **current** ⟨*expr*⟩ //current expressions
| **merge** *IDENT* ⟨*expr*⟩ ⟨*expr*⟩ //merge expressions
| **merge** *IDENT* ⟨*merge_case_list*⟩ //merge expressions

⟨*merge_case_list*⟩ ::= ⟨*merge_case*⟩ { ⟨*merge_case*⟩ } //merge case list

⟨*merge_case*⟩ ::= '(' ⟨*merge_head*⟩ '->' ⟨*expr*⟩ ')' //merge case

⟨*merge_head*⟩ ::= *IDENT* //enum identifiers
| **true** //merge bool
| **false** //merge bool

⟨*unop*⟩ ::= '+' //unary plus
| '-' //unary minus
| **short** //convert to short(signed 16 bits)
| **int** //convert to int(signed 32 bits)
| **float** //convert to float(32 bits)
| **real** //convert to real(64 bits)
| **not** //boolean negation

⟨*binop*⟩ ::= '+' //addition
| '-' //subtraction
| '*' //multiplication
| '/' //division real
| **div** //division integer
| **mod** //remainder
| **and** //logical and

```
            | or          //logical or
            | xor         //logical exclusive or
            | '='         //equality between any type of values
            | '<>'        //inequality between any type of values
            | '<'         //lower on numerics
            | '>'         //greater on numerics
            | '<='        //lower or equal on numerics
            | '>='        //greater or equal on numerics

⟨nary⟩ ::= '#'          //boolred(0,1,n)
         | nor          //boolred(0,0,n)

⟨pattern_expr⟩ ::= ⟨pattern⟩ ':' ⟨expr⟩

⟨pattern⟩ ::= IDENT          //pattern identifier
            | CHAR           //pattern char
            | [ - ] INTEGER          //pattern integer
            | true           //pattern bool
            | false          //pattern bool
            | '_'            //pattern any

⟨struct_expr⟩ ::= ⟨expr⟩ '.' IDENT          //access to a member of a struct
                | '{' ⟨field_expr⟩ { ',' ⟨field_expr⟩ } '}'      //construct a struct

⟨field_expr⟩ ::= IDENT : ⟨expr⟩

⟨array_expr⟩ ::= ⟨expr⟩ ⟨index⟩      //access to (index+1)th member of an array expr
                | ⟨expr⟩ '^ ' INTEGER          //one way to build an array
                | '[' ⟨expr_list⟩ ']'          //another way to build an array
                | '(' ⟨expr⟩ '.' ⟨index⟩ { ⟨index⟩ } default ⟨expr⟩ ')'
                                                //dynamic projection
                | ⟨expr⟩ '[' ⟨expr⟩ '..' ⟨expr⟩ ']'          //array slice
                | '(' ⟨expr⟩ with ⟨label_index⟩ { ⟨label_index⟩ } '=' ⟨expr⟩ ')'
                                                //construct for a new array or struct

⟨index⟩ ::= '[' ⟨expr⟩ ']'

⟨label_index⟩ ::= '.' IDENT
                | ⟨index⟩

⟨apply_expr⟩ ::= ⟨operator⟩ ⟨expr_list⟩

⟨operator⟩ ::= ⟨prefix_op⟩
             | ⟨iterator_op⟩ '<<' ⟨operator⟩ ',' INTEGER '<<'

⟨prefix_op⟩ ::= IDENT
              | ⟨prefix_unop⟩
              | ⟨prefix_binop⟩

⟨prefix_unop⟩ ::= short$          //convert to short(signed 16 bits)
                | int$          //convert to int(signed 32 bits)
                | float$        //convert to float(32 bits)
```

| **real**$ //convert to real(64 bits)
| **not**$ //boolean negation
| +$ //unary plus
| -$ //unary minus

⟨*prefix_binop*⟩ ::= $+$ //addition
| $-$ //subtraction
| $*$ //multiplication
| $/$' //division real
| $**div**$ //division integer
| $**mod**$ //remainder
| $**and**$ //logical and
| $**or**$ //logical or
| $**xor**$ //logical exclusive or
| $=$ //equality between any type of values
| $<>$ //inequality between any type of values
| $<$ //lower on numerics
| $>$ //greater on numerics
| $<=$ //lower or equal on numerics
| $>=$ //greater or equal on numerics

⟨*iterator_op*⟩ ::= **map** //higher-order operator *map*
| **fill** //higher-order operator *fill*
| **red** //higher-order operator *red*
| **fillred** //higher-order operator *fillred*