The Lustre V6 Grammar Rules

Erwan Jahier, Pascal Raymond, Nicolas Halbwachs

Software Version: 6.101.27 (17-08-20)

We recall that grammar rules are given using an extended BNF notation, where non-terminals are written ⟨*like_this*⟩ and terminals "**`like that`**". All non-terminals (should) have pdf internal links to ease the reading.

- One-line comments start with `--` and stop at the the end of the line.
- Multi-line comments start with '`(*`' and end at the next following '`*)`' ('`/*`' and '`*/`' also work). Multi-line comments cannot be nested.
- *TK_IDENT* stands for identifier: `[_a-zA-Z][_a-zA-Z0-9]*`
- *TK_LONGIDENT* stands for pointed (or long) identifier, that is, two identifiers separated by a double colon: *TK_IDENT*`::`*TKIDENT*

Ebnf group *ProgramRules*

⟨*program*⟩	::=	{ ⟨*Include*⟩ } (⟨*PackBody*⟩ | ⟨*PackList*⟩)
⟨*Include*⟩	::=	**`include "<string>"`**
⟨*PackBody*⟩	::=	⟨*OneDecl*⟩ { ⟨*OneDecl*⟩ }
⟨*OneDecl*⟩	::=	⟨*ConstDecl*⟩ | ⟨*TypeDecl*⟩ | ⟨*ExtNodeDecl*⟩ | ⟨*NodeDecl*⟩

Ebnf group *PackageRules*

⟨*PackList*⟩	::=	⟨*OnePack*⟩ { ⟨*OnePack*⟩ }
⟨*OnePack*⟩	::=	⟨*ModelDecl*⟩ | ⟨*PackDecl*⟩ | ⟨*PackEq*⟩
⟨*PackDecl*⟩	::=	**`package`** ⟨*Lv6Id*⟩ ⟨*Uses*⟩ ⟨*Provides*⟩ **`body`** ⟨*PackBody*⟩ **`end`**
⟨*Uses*⟩	::=	[**`uses`** ⟨*Lv6Id*⟩ { **`,`** ⟨*Lv6Id*⟩ } **`;`**]
⟨*Eq_or_Is*⟩	::=	**`=`**
	|	**`is`**
⟨*PackEq*⟩	::=	**`package`** ⟨*Lv6Id*⟩ ⟨*Eq_or_Is*⟩ ⟨*Lv6Id*⟩ **`(`** ⟨*ByNameStaticArgList*⟩ **`)`** **`;`**

Ebnf group *ModelRules*

⟨*Provides*⟩	::=	[**`provides`** ⟨*Provide*⟩ **`;`** { ⟨*Provide*⟩ **`;`** }]
⟨*Provide*⟩	::=	**`const`** ⟨*Lv6Id*⟩ **`:`** ⟨*Type*⟩ [**`=`** ⟨*Expression*⟩]
	|	**`unsafe node`** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*Params*⟩ **`returns`** ⟨*Params*⟩
	|	**`node`** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*Params*⟩ **`returns`** ⟨*Params*⟩
	|	**`unsafe function`** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*Params*⟩ **`returns`** ⟨*Params*⟩
	|	**`function`** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*Params*⟩ **`returns`** ⟨*Params*⟩
	|	**`type`** ⟨*OneTypeDecl*⟩

⟨*ModelDecl*⟩ ::= **model** ⟨*Lv6Id*⟩ ⟨*Uses*⟩ **needs** ⟨*StaticParamList*⟩ **;** ⟨*Provides*⟩ **body** ⟨*PackBody*⟩ **end**

Ebnf group *ConstRules*

Ebnf group *IdentVIRules*

⟨*Lv6IdRef*⟩ ::= ⟨*TK_IDENT*⟩
| ⟨*TK_LONGIDENT*⟩

Ebnf group *IdentRules*

⟨*Lv6Id*⟩ ::= ⟨*TK_IDENT*⟩ ⟨*Pragma*⟩
⟨*Pragma*⟩ ::= { **%** ⟨*TK_IDENT*⟩ **:** ⟨*TK_IDENT*⟩ **%** }

Ebnf group *NodesRules*

⟨*TypedLv6IdsList*⟩ ::= ⟨*TypedLv6Ids*⟩ { **;** ⟨*TypedLv6Ids*⟩ }
⟨*TypedLv6Ids*⟩ ::= ⟨*Lv6Id*⟩ { **,** ⟨*Lv6Id*⟩ } **:** ⟨*Type*⟩
⟨*TypedValuedLv6Ids*⟩ ::= ⟨*TypedValuedLv6Id*⟩ { **;** ⟨*TypedValuedLv6Id*⟩ }
⟨*TypedValuedLv6Id*⟩ ::= ⟨*Lv6Id*⟩ (**:** ⟨*Type*⟩ | **,** ⟨*Lv6Id*⟩ { **,** ⟨*Lv6Id*⟩ } **:** ⟨*Type*⟩ | **:** ⟨*Type*⟩ **=** ⟨*Expression*⟩)
⟨*NodeDecl*⟩ ::= ⟨*LocalNode*⟩
⟨*LocalNode*⟩ ::= **node** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*Params*⟩ **returns** ⟨*Params*⟩ [**;**] ⟨*LocalDecls*⟩ ⟨*Body*⟩ (**.** | [**;**])
| **function** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*Params*⟩ **returns** ⟨*Params*⟩ [**;**] ⟨*LocalDecls*⟩ ⟨*Body*⟩ (**.** | [**;**])
| **node** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*NodeProfileOpt*⟩ **=** ⟨*EffectiveNode*⟩ [**;**]
| **function** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*NodeProfileOpt*⟩ **=** ⟨*EffectiveNode*⟩ [**;**]
| **unsafe node** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*Params*⟩ **returns** ⟨*Params*⟩ [**;**] ⟨*LocalDecls*⟩ ⟨*Body*⟩ (**.** | [**;**])
| **unsafe function** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*Params*⟩ **returns** ⟨*Params*⟩ [**;**] ⟨*LocalDecls*⟩ ⟨*Body*⟩ (**.** | [**;**])
| **unsafe node** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*NodeProfileOpt*⟩ **=** ⟨*EffectiveNode*⟩ [**;**]

	|	**unsafe function** ⟨*Lv6Id*⟩ ⟨*StaticParams*⟩ ⟨*NodeProfileOpt*⟩ **=** ⟨*EffectiveNode*⟩ [**;**]
⟨*NodeProfileOpt*⟩	::=	[⟨*Params*⟩ **returns** ⟨*Params*⟩]
⟨*Params*⟩	::=	**(** [⟨*VarDeclList*⟩ [**;**]] **)**
⟨*LocalDecls*⟩	::=	[⟨*LocalDeclList*⟩]
⟨*LocalDeclList*⟩	::=	⟨*OneLocalDecl*⟩ { ⟨*OneLocalDecl*⟩ }
⟨*OneLocalDecl*⟩	::=	⟨*LocalVars*⟩
	|	⟨*LocalConsts*⟩
⟨*LocalConsts*⟩	::=	**const** ⟨*ConstDeclList*⟩
⟨*LocalVars*⟩	::=	**var** ⟨*VarDeclList*⟩ **;**
⟨*VarDeclList*⟩	::=	⟨*VarDecl*⟩ { **;** ⟨*VarDecl*⟩ }
⟨*VarDecl*⟩	::=	⟨*TypedLv6Ids*⟩
	|	⟨*TypedLv6Ids*⟩ **when** ⟨*ClockExpr*⟩
	|	**(** ⟨*TypedLv6IdsList*⟩ **) when** ⟨*ClockExpr*⟩

Ebnf group *ConstantDeclRules*

⟨*ConstDecl*⟩	::=	**const** ⟨*ConstDeclList*⟩
⟨*ConstDeclList*⟩	::=	⟨*OneConstDecl*⟩ **;** { ⟨*OneConstDecl*⟩ **;** }
⟨*OneConstDecl*⟩	::=	⟨*Lv6Id*⟩ (**:** ⟨*Type*⟩ | **,** ⟨*Lv6Id*⟩ { **,** ⟨*Lv6Id*⟩ } **:** ⟨*Type*⟩ | **:** ⟨*Type*⟩ **=** ⟨*Expression*⟩ | **=** ⟨*Expression*⟩)

Ebnf group *TypeDeclRules*

⟨*TypeDecl*⟩	::=	**type** ⟨*TypeDeclList*⟩
⟨*TypeDeclList*⟩	::=	⟨*OneTypeDecl*⟩ **;** { ⟨*OneTypeDecl*⟩ **;** }
⟨*OneTypeDecl*⟩	::=	⟨*Lv6Id*⟩ [**=** (⟨*Type*⟩ | **enum {** ⟨*Lv6Id*⟩ { **,** ⟨*Lv6Id*⟩ } **}** | [**struct**] **{** ⟨*TypedValuedLv6Ids*⟩ [**;**] **}**)]

Ebnf group *SimpleTypeRules*

⟨*Type*⟩	::=	(**bool** | **int** | **real** | ⟨*Lv6IdRef*⟩) { **^** ⟨*Expression*⟩ }

Ebnf group *ExtNodesRules*

⟨*ExtNodeDecl*⟩	::=	(**extern function** | **unsafe extern function** | **extern node** | **unsafe extern node**) ⟨*Lv6Id*⟩ ⟨*Params*⟩ **returns** ⟨*Params*⟩ [**;**]

Ebnf group *StaticRules*

```
⟨StaticParams⟩          ::= [ << ⟨StaticParamList⟩ >> ]
⟨StaticParamList⟩       ::= ⟨StaticParam⟩ { ; ⟨StaticParam⟩ }
⟨StaticParam⟩           ::= type ⟨Lv6Id⟩
                          | const ⟨Lv6Id⟩ : ⟨Type⟩
                          | node ⟨Lv6Id⟩ ⟨Params⟩ returns ⟨Params⟩
                          | function ⟨Lv6Id⟩ ⟨Params⟩ returns ⟨Params⟩
                          | unsafe node ⟨Lv6Id⟩ ⟨Params⟩ returns ⟨Params⟩
                          | unsafe function ⟨Lv6Id⟩ ⟨Params⟩ returns ⟨Params⟩
⟨EffectiveNode⟩         ::= ⟨Lv6IdRef⟩ [ << ⟨StaticArgList⟩ >> ]
⟨StaticArgList⟩         ::= ⟨StaticArg⟩ { ( , | ; ) ⟨StaticArg⟩ }
⟨StaticArg⟩             ::= type ⟨Type⟩
                          | const ⟨Expression⟩
                          | node ⟨EffectiveNode⟩
                          | function ⟨EffectiveNode⟩
                          | ⟨PredefOp⟩
                          | ⟨SimpleExp⟩
                          | ⟨SurelyType⟩
                          | ⟨SurelyNode⟩
⟨ByNameStaticArgList⟩   ::= ⟨ByNameStaticArg⟩ { ( , | ; ) ⟨ByNameStaticArg⟩ }
⟨ByNameStaticArg⟩       ::= type ⟨Lv6Id⟩ = ⟨Type⟩
                          | const ⟨Lv6Id⟩ = ⟨Expression⟩
                          | node ⟨Lv6Id⟩ = ⟨EffectiveNode⟩
                          | function ⟨Lv6Id⟩ = ⟨EffectiveNode⟩
                          | ⟨Lv6Id⟩ = ⟨PredefOp⟩
                          | ⟨Lv6Id⟩ = ⟨SimpleExp⟩
                          | ⟨Lv6Id⟩ = ⟨SurelyType⟩
                          | ⟨Lv6Id⟩ = ⟨SurelyNode⟩
⟨SurelyNode⟩            ::= ⟨Lv6IdRef⟩ << ⟨StaticArgList⟩ >>
⟨SurelyType⟩            ::= ( bool | int | real ) { ^ ⟨Expression⟩ }
⟨SimpleExp⟩             ::= ⟨Constant⟩
                          | ⟨Lv6IdRef⟩
                          | ⟨SimpleTuple⟩
                          | not ⟨SimpleExp⟩
                          | - ⟨SimpleExp⟩
                          | ⟨SimpleExp⟩ and ⟨SimpleExp⟩
                          | ⟨SimpleExp⟩ or ⟨SimpleExp⟩
                          | ⟨SimpleExp⟩ xor ⟨SimpleExp⟩
                          | ⟨SimpleExp⟩ => ⟨SimpleExp⟩
                          | ⟨SimpleExp⟩ = ⟨SimpleExp⟩
                          | ⟨SimpleExp⟩ <> ⟨SimpleExp⟩
```

	|	⟨*SimpleExp*⟩ `<` ⟨*SimpleExp*⟩
	|	⟨*SimpleExp*⟩ `<=` ⟨*SimpleExp*⟩
	|	⟨*SimpleExp*⟩ `>` ⟨*SimpleExp*⟩
	|	⟨*SimpleExp*⟩ `>=` ⟨*SimpleExp*⟩
	|	⟨*SimpleExp*⟩ `div` ⟨*SimpleExp*⟩
	|	⟨*SimpleExp*⟩ `mod` ⟨*SimpleExp*⟩
	|	⟨*SimpleExp*⟩ `-` ⟨*SimpleExp*⟩
	|	⟨*SimpleExp*⟩ `+` ⟨*SimpleExp*⟩
	|	⟨*SimpleExp*⟩ `/` ⟨*SimpleExp*⟩
	|	⟨*SimpleExp*⟩ `*` ⟨*SimpleExp*⟩
	|	`if` ⟨*SimpleExp*⟩ `then` ⟨*SimpleExp*⟩ `else` ⟨*SimpleExp*⟩
⟨*SimpleTuple*⟩	::=	[`(` ⟨*SimpleExpList*⟩ `)`]
⟨*SimpleExpList*⟩	::=	⟨*SimpleExp*⟩ { `,` ⟨*SimpleExp*⟩ }

Ebnf group *BodyRules*

⟨*Body*⟩	::=	`let` [⟨*EquationList*⟩] `tel`
⟨*EquationList*⟩	::=	⟨*Equation*⟩ { ⟨*Equation*⟩ }
⟨*Equation*⟩	::=	(`assert` | ⟨*Left*⟩ `=`) ⟨*Expression*⟩ `;`

Ebnf group *LeftRules*

⟨*Left*⟩	::=	⟨*LeftItemList*⟩
	|	`(` ⟨*LeftItemList*⟩ `)`
⟨*LeftItemList*⟩	::=	⟨*LeftItem*⟩ { `,` ⟨*LeftItem*⟩ }
⟨*LeftItem*⟩	::=	⟨*Lv6Id*⟩
	|	⟨*FieldLeftItem*⟩
	|	⟨*TableLeftItem*⟩
⟨*FieldLeftItem*⟩	::=	⟨*LeftItem*⟩ `.` ⟨*Lv6Id*⟩
⟨*TableLeftItem*⟩	::=	⟨*LeftItem*⟩ `[` (⟨*Expression*⟩ | ⟨*Select*⟩) `]`
⟨*Select*⟩	::=	⟨*Expression*⟩ `..` ⟨*Expression*⟩ ⟨*Step*⟩
⟨*Step*⟩	::=	[`step` ⟨*Expression*⟩]

Ebnf group *ExpressionRules*

⟨*Expression*⟩	::=	⟨*Constant*⟩
	|	⟨*Lv6IdRef*⟩
	|	`not` ⟨*Expression*⟩
	|	`-` ⟨*Expression*⟩
	|	`pre` ⟨*Expression*⟩

```
                        | current ⟨Expression⟩
                        | int ⟨Expression⟩
                        | real ⟨Expression⟩
                        | ⟨Expression⟩ when ⟨ClockExpr⟩
                        | ⟨Expression⟩ fby ⟨Expression⟩
                        | ⟨Expression⟩ -> ⟨Expression⟩
                        | ⟨Expression⟩ and ⟨Expression⟩
                        | ⟨Expression⟩ or ⟨Expression⟩
                        | ⟨Expression⟩ xor ⟨Expression⟩
                        | ⟨Expression⟩ => ⟨Expression⟩
                        | ⟨Expression⟩ = ⟨Expression⟩
                        | ⟨Expression⟩ <> ⟨Expression⟩
                        | ⟨Expression⟩ < ⟨Expression⟩
                        | ⟨Expression⟩ <= ⟨Expression⟩
                        | ⟨Expression⟩ > ⟨Expression⟩
                        | ⟨Expression⟩ >= ⟨Expression⟩
                        | ⟨Expression⟩ div ⟨Expression⟩
                        | ⟨Expression⟩ mod ⟨Expression⟩
                        | ⟨Expression⟩ - ⟨Expression⟩
                        | ⟨Expression⟩ + ⟨Expression⟩
                        | ⟨Expression⟩ / ⟨Expression⟩
                        | ⟨Expression⟩ * ⟨Expression⟩
                        | if ⟨Expression⟩ then ⟨Expression⟩ else ⟨Expression⟩
                        | with ⟨Expression⟩ then ⟨Expression⟩ else ⟨Expression⟩
                        | # ( ⟨ExpressionList⟩ )
                        | nor ( ⟨ExpressionList⟩ )
                        | ⟨CallByPosExpression⟩
                        | [ ⟨ExpressionList⟩ ]
                        | ⟨Expression⟩ ^ ⟨Expression⟩
                        | ⟨Expression⟩ | ⟨Expression⟩
                        | ⟨Expression⟩ [ ⟨Expression⟩ ]
                        | ⟨Expression⟩ [ ⟨Select⟩ ]
                        | ⟨Expression⟩ . ⟨Lv6Id⟩
                        | ⟨CallByNameExpression⟩
                        | ( ⟨ExpressionList⟩ )
                        | merge ⟨Lv6Id⟩ ⟨MergeCaseList⟩
⟨ExpressionList⟩      ::= [ ⟨Expression⟩ ] { , ⟨Expression⟩ }
⟨ClockExpr⟩           ::= ⟨Lv6IdRef⟩ ( ⟨Lv6Id⟩ )
                        | ⟨Lv6Id⟩
                        | not ⟨Lv6Id⟩
                        | not ( ⟨Lv6Id⟩ )
⟨CallByPosExpression⟩ ::= ⟨EffectiveNode⟩ ( ⟨ExpressionList⟩ )
```

Ebnf group *MergeRules*

⟨*MergeCaseList*⟩ ::= [⟨*MergeCase*⟩] { ⟨*MergeCase*⟩ }
⟨*MergeCase*⟩ ::= [`(` (⟨*Lv6IdRef*⟩ | `true` | `false`) `->` ⟨*Expression*⟩ `)`]

Ebnf group *PredefRules*

⟨*PredefOp*⟩ ::= `not` | `fby` | `pre` | `current` | `->` | `and` | `or` | `xor` | `=>` | `=` | `<>` | `<` | `<=` | `>` | `>=` | `div` | `mod` | `-` | `+` | `/` | `*` | `if`

Ebnf group *ExpressionByNamesRules*

⟨*CallByNameExpression*⟩ ::= [⟨*Lv6IdRef*⟩ `{` [[⟨*Lv6IdRef*⟩ `with`] ⟨*CallByNameParamList*⟩ [`;`]] `}`]
⟨*CallByNameParamList*⟩ ::= ⟨*CallByNameParam*⟩ { (`;` | `,`) ⟨*CallByNameParam*⟩ }
⟨*CallByNameParam*⟩ ::= ⟨*Lv6Id*⟩ `=` ⟨*Expression*⟩

Ebnf group *ConstantRules*

⟨*Constant*⟩ ::= `true` | `false` | ⟨*IntConst*⟩ | ⟨*RealConst*⟩