

The Language LAMA

BNF-converter

August 17, 2012

This document was automatically generated by the *BNF-Converter*. It was generated together with the lexer, the parser, and the abstract syntax module, which guarantees that the document matches with the implementation of the language (provided no hand-hacking has taken place).

The lexical structure of LAMA

Literals

Integer literals $\langle Int \rangle$ are nonempty sequences of digits.

Identifier literals are recognized by the regular expression $(\langle letter \rangle \mid \text{'_'})(\langle letter \rangle \mid \langle digit \rangle \mid \text{'_'})^*$

StateId literals are recognized by the regular expression $(\langle letter \rangle \mid \text{'_'})(\langle letter \rangle \mid \langle digit \rangle \mid \text{'_'})^* \text{'"}$

Reserved words and symbols

The set of reserved words is the set of terminals appearing in the grammar. Those reserved words that consist of non-letter characters are called symbols, and they are treated in a different way from those that are similar to identifiers. The lexer follows rules familiar from languages like Haskell, C, and Java, including longest match and spacing conventions.

The reserved words used in LAMA are the following:

and	assertion	automaton
bool	constants	default
definition	div	edge
enum	false	initial
int	invariant	ite
let	local	location
match	mod	node
nodes	not	or
project	real	returns
sint	state	tel
transition	true	typedef
uint	use	xor

The symbols used in LAMA are the following:

```

;    =    {
}    ,    ^
(    #    )
[    ]    -
/    :    .
-    =>    <
>    <=    >=
+    *

```

Comments

Single-line comments begin with `--`.

There are no multiple-line comments in the grammar.

The syntactic structure of LAMA

Non-terminals are enclosed between \langle and \rangle . The symbols $::=$ (production), $|$ (union) and ϵ (empty rule) belong to the BNF notation. All other symbols are terminals.

$$\begin{aligned}
 \langle Program \rangle & ::= \langle TypeDefs \rangle \langle ConstantDefs \rangle \\
 & \quad \langle Declarations \rangle \langle Flow \rangle \langle Initial \rangle \\
 & \quad \langle Assertion \rangle \langle Invariant \rangle \\
 \langle TypeDefs \rangle & ::= \epsilon \\
 & \quad | \quad \text{typedef } \langle ListTypeDef \rangle \\
 \langle ListTypeDef \rangle & ::= \langle TypeDef \rangle ; \\
 & \quad | \quad \langle TypeDef \rangle ; \langle ListTypeDef \rangle
 \end{aligned}$$

$$\begin{aligned}
\langle \text{TypeDef} \rangle &::= \text{enum } \langle \text{Identifier} \rangle = \{ \langle \text{ListEnumConstr} \rangle \} \\
\langle \text{EnumConstr} \rangle &::= \langle \text{Identifier} \rangle \\
\langle \text{ListEnumConstr} \rangle &::= \langle \text{EnumConstr} \rangle \\
&\quad | \quad \langle \text{EnumConstr} \rangle , \langle \text{ListEnumConstr} \rangle \\
\langle \text{Type} \rangle &::= \langle \text{BaseType} \rangle \\
&\quad | \quad \langle \text{Identifier} \rangle \\
&\quad | \quad \langle \text{BaseType} \rangle \sim \langle \text{Natural} \rangle \\
&\quad | \quad (\# \langle \text{ListType} \rangle) \\
\langle \text{ListType} \rangle &::= \langle \text{Type} \rangle \\
&\quad | \quad \langle \text{Type} \rangle \langle \text{ListType} \rangle \\
\langle \text{BaseType} \rangle &::= \text{bool} \\
&\quad | \quad \text{int} \\
&\quad | \quad \text{real} \\
&\quad | \quad \text{ sint } [\langle \text{Natural} \rangle] \\
&\quad | \quad \text{ uint } [\langle \text{Natural} \rangle] \\
\langle \text{ConstantDefs} \rangle &::= \epsilon \\
&\quad | \quad \text{constants } \langle \text{ListConstantDef} \rangle \\
\langle \text{ListConstantDef} \rangle &::= \langle \text{ConstantDef} \rangle ; \\
&\quad | \quad \langle \text{ConstantDef} \rangle ; \langle \text{ListConstantDef} \rangle \\
\langle \text{ConstantDef} \rangle &::= \langle \text{Identifier} \rangle = \langle \text{Constant} \rangle \\
\langle \text{Natural} \rangle &::= \langle \text{Integer} \rangle \\
\langle \text{IntegerConst} \rangle &::= \langle \text{Integer} \rangle \\
&\quad | \quad (- \langle \text{Integer} \rangle) \\
\langle \text{Constant} \rangle &::= \langle \text{BoolV} \rangle \\
&\quad | \quad \langle \text{IntegerConst} \rangle \\
&\quad | \quad \langle \text{IntegerConst} \rangle / \langle \text{IntegerConst} \rangle \\
&\quad | \quad \text{ sint } [\langle \text{Natural} \rangle] (\langle \text{IntegerConst} \rangle) \\
&\quad | \quad \text{ uint } [\langle \text{Natural} \rangle] (\langle \text{Natural} \rangle) \\
\langle \text{BoolV} \rangle &::= \text{true} \\
&\quad | \quad \text{false} \\
\langle \text{Initial} \rangle &::= \epsilon \\
&\quad | \quad \text{initial } \langle \text{ListStateInit} \rangle ; \\
\langle \text{Assertion} \rangle &::= \epsilon \\
&\quad | \quad \text{assertion } \langle \text{Expr} \rangle ;
\end{aligned}$$

$$\begin{aligned}
\langle \text{Invariant} \rangle &::= \epsilon \\
&| \quad \text{invariant } \langle \text{Expr} \rangle ; \\
\langle \text{ListStateInit} \rangle &::= \langle \text{StateInit} \rangle \\
&| \quad \langle \text{StateInit} \rangle , \langle \text{ListStateInit} \rangle \\
\langle \text{StateInit} \rangle &::= \langle \text{Identifier} \rangle = \langle \text{ConstExpr} \rangle \\
\langle \text{ConstExpr} \rangle &::= \langle \text{Expr} \rangle \\
\langle \text{ListIdentifier} \rangle &::= \langle \text{Identifier} \rangle \\
&| \quad \langle \text{Identifier} \rangle , \langle \text{ListIdentifier} \rangle \\
\langle \text{TypedVars} \rangle &::= \langle \text{ListIdentifier} \rangle : \langle \text{Type} \rangle \\
\langle \text{ListTypedVars} \rangle &::= \langle \text{TypedVars} \rangle \\
&| \quad \langle \text{TypedVars} \rangle ; \langle \text{ListTypedVars} \rangle \\
\langle \text{MaybeTypedVars} \rangle &::= \epsilon \\
&| \quad \langle \text{ListTypedVars} \rangle \\
\langle \text{Node} \rangle &::= \text{node } \langle \text{Identifier} \rangle (\langle \text{MaybeTypedVars} \rangle) \text{ returns } (\langle \text{ListTypedVars} \rangle) ; \\
&\quad \text{let} \\
&\quad \quad \langle \text{Declarations} \rangle \\
&\quad \quad \langle \text{Flow} \rangle \\
&\quad \quad \langle \text{ControlStructure} \rangle \\
&\quad \quad \langle \text{Initial} \rangle \\
&\quad \quad \langle \text{Assertion} \rangle \\
&\quad \text{tel} \\
\langle \text{ListNode} \rangle &::= \langle \text{Node} \rangle \\
&| \quad \langle \text{Node} \rangle \langle \text{ListNode} \rangle \\
\langle \text{Declarations} \rangle &::= \langle \text{NodeDecls} \rangle \langle \text{LocalDecls} \rangle \langle \text{StateDecls} \rangle \\
\langle \text{VarDecls} \rangle &::= \langle \text{TypedVars} \rangle ; \\
&| \quad \langle \text{TypedVars} \rangle ; \langle \text{VarDecls} \rangle \\
\langle \text{NodeDecls} \rangle &::= \epsilon \\
&| \quad \text{nodes } \langle \text{ListNode} \rangle \\
\langle \text{LocalDecls} \rangle &::= \epsilon \\
&| \quad \text{local } \langle \text{VarDecls} \rangle \\
\langle \text{StateDecls} \rangle &::= \epsilon \\
&| \quad \text{state } \langle \text{VarDecls} \rangle
\end{aligned}$$

$$\begin{aligned}
\langle \text{Flow} \rangle &::= \langle \text{LocalDefinitions} \rangle \langle \text{Transitions} \rangle \\
\langle \text{LocalDefinitions} \rangle &::= \epsilon \\
&| \quad \text{definition } \langle \text{ListInstantDefinition} \rangle \\
\langle \text{Transitions} \rangle &::= \epsilon \\
&| \quad \text{transition } \langle \text{ListTransition} \rangle \\
\langle \text{ListInstantDefinition} \rangle &::= \langle \text{InstantDefinition} \rangle ; \\
&| \quad \langle \text{InstantDefinition} \rangle ; \langle \text{ListInstantDefinition} \rangle \\
\langle \text{ListTransition} \rangle &::= \langle \text{Transition} \rangle ; \\
&| \quad \langle \text{Transition} \rangle ; \langle \text{ListTransition} \rangle \\
\langle \text{InstantDefinition} \rangle &::= \langle \text{Identifier} \rangle = \langle \text{Expr} \rangle \\
&| \quad \langle \text{Identifier} \rangle = (\text{use } \langle \text{Identifier} \rangle \langle \text{ListExpr} \rangle) \\
\langle \text{Transition} \rangle &::= \langle \text{StateId} \rangle = \langle \text{Expr} \rangle \\
\langle \text{ControlStructure} \rangle &::= \langle \text{ListAutomaton} \rangle \\
\langle \text{Automaton} \rangle &::= \text{automaton let } \langle \text{ListLocation} \rangle \langle \text{InitialLocation} \rangle \langle \text{ListEdge} \rangle \langle \text{Defaults} \rangle \text{ tel} \\
\langle \text{Location} \rangle &::= \text{location } \langle \text{Identifier} \rangle \text{ let } \langle \text{Flow} \rangle \text{ tel} \\
\langle \text{InitialLocation} \rangle &::= \text{initial } \langle \text{Identifier} \rangle ; \\
\langle \text{Edge} \rangle &::= \text{edge } (\langle \text{Identifier} \rangle , \langle \text{Identifier} \rangle) : \langle \text{Expr} \rangle ; \\
\langle \text{ListLocation} \rangle &::= \langle \text{Location} \rangle \\
&| \quad \langle \text{Location} \rangle \langle \text{ListLocation} \rangle \\
\langle \text{ListEdge} \rangle &::= \langle \text{Edge} \rangle \\
&| \quad \langle \text{Edge} \rangle \langle \text{ListEdge} \rangle \\
\langle \text{ListAutomaton} \rangle &::= \epsilon \\
&| \quad \langle \text{Automaton} \rangle \langle \text{ListAutomaton} \rangle \\
\langle \text{Defaults} \rangle &::= \epsilon \\
&| \quad \text{default } \langle \text{ListDefault} \rangle ; \\
\langle \text{ListDefault} \rangle &::= \langle \text{Default} \rangle \\
&| \quad \langle \text{Default} \rangle , \langle \text{ListDefault} \rangle \\
\langle \text{Default} \rangle &::= \langle \text{Identifier} \rangle = \langle \text{Expr} \rangle
\end{aligned}$$

$$\begin{aligned}
\langle Atom \rangle & ::= \langle Constant \rangle \\
& \quad | \quad \langle Identifier \rangle \\
\langle Expr \rangle & ::= \langle Atom \rangle \\
& \quad | \quad (\langle UnOp \rangle \langle Expr \rangle) \\
& \quad | \quad (\langle BinOp \rangle \langle Expr \rangle \langle Expr \rangle) \\
& \quad | \quad (\langle TernOp \rangle \langle Expr \rangle \langle Expr \rangle \langle Expr \rangle) \\
& \quad | \quad (\# \langle ListExpr \rangle) \\
& \quad | \quad (\text{project } \langle Identifier \rangle \langle Natural \rangle) \\
& \quad | \quad (\text{match } \langle Expr \rangle \{ \langle ListPattern \rangle \}) \\
\langle ListExpr \rangle & ::= \epsilon \\
& \quad | \quad \langle Expr \rangle \langle ListExpr \rangle \\
\langle ListPattern \rangle & ::= \langle Pattern \rangle \\
& \quad | \quad \langle Pattern \rangle , \langle ListPattern \rangle \\
\langle Pattern \rangle & ::= \langle PatHead \rangle . \langle Expr \rangle \\
\langle PatHead \rangle & ::= \langle EnumConstr \rangle \\
& \quad | \quad - \\
\langle List2Id \rangle & ::= \langle Identifier \rangle \langle Identifier \rangle \\
& \quad | \quad \langle Identifier \rangle \langle List2Id \rangle \\
\langle UnOp \rangle & ::= \text{not} \\
\langle BinOp \rangle & ::= \text{or} \\
& \quad | \quad \text{and} \\
& \quad | \quad \text{xor} \\
& \quad | \quad => \\
& \quad | \quad = \\
& \quad | \quad < \\
& \quad | \quad > \\
& \quad | \quad <= \\
& \quad | \quad >= \\
& \quad | \quad + \\
& \quad | \quad - \\
& \quad | \quad * \\
& \quad | \quad / \\
& \quad | \quad \text{div} \\
& \quad | \quad \text{mod} \\
\langle TernOp \rangle & ::= \text{ite}
\end{aligned}$$