The grammar of the AltaRica language and its syntactic tree Version X.XX

A. Griffault, G. Point, A. Vincent

March 15, 2006

Contents

This document presents the grammar of the AltaRica language. Non-terminals are written using an *italic* font. A TYPEWRITER font and small capitals are used for terminals. The *opt* subscript is added to optional terminals or non-terminals.

This following example describes the two production rules for the non-terminal *definition-list*. *definition* is another non-terminal. The semi-colon "**;**" is an optional terminal.

definition-list ::= *definition-list* $;_{opt}$ *definition*
::= *definition*

In this document we also describe the syntactic tree which should be produced by a valid parser of the AltaRica grammar. The following schema (Fig. 1) gives a graphical representation of a syntactic tree. A node of the tree is represented by a box containing a label (e.g. EXAMPLE) and two points (•).

Labels written using an *italic* font (e.g. *abstract-child3*) are not actual labels; they are used for the sake of clarity in place of a set of actual labels. Some labels are followed by two numbers (e.g. CHILD1 1/2) which are respectively the section and the page number of the definition of that node.

The left point in the box represents the pointer to the child of the node. The children of a node are chained using the right point.

The right-hand side of the figure 1 shows the ANSI-C structure used to store the nodes. An extra-field *value* (not shown on schemas) is used to store the value associated to some terminals: IDENTIFIER which the value is a character string and INTEGER which the value is an unsigned integer.

The NULL pointer (terminating any node chain) is represented by a hatched half-circle. Dots depict a subtree which is not represented. Dotted arrows depict optional links; for example, on the figure 1 the node labelled by CHILD1 might be chained with NULL or with a node labelled by CHILD2.

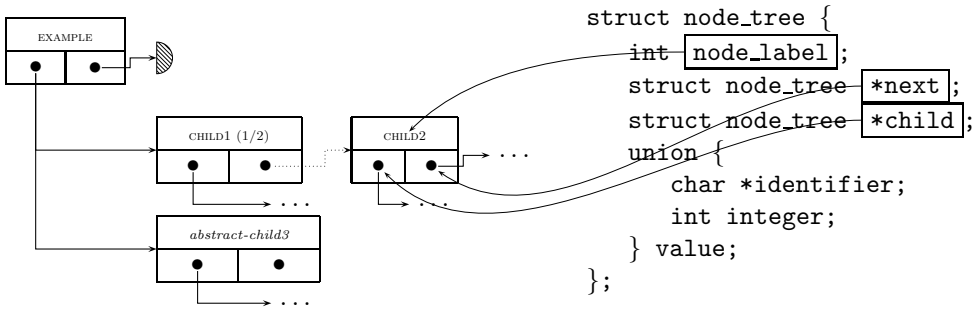


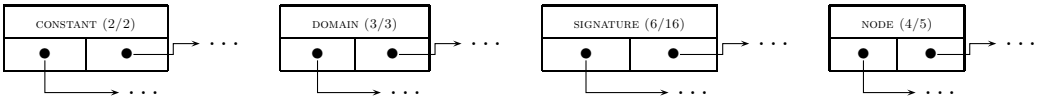
Figure 1:

1 AltaRica description

altarica-description ::= *definition-list*$_{opt}$

definition-list ::= *definition-list* $;_{opt}$ *definition*
::= *definition*

definition ::= *constant-definition*
::= *domain-definition*
::= *sort-declaration*
::= *signature-declaration*
::= *node-definition*



2 Definition of constants

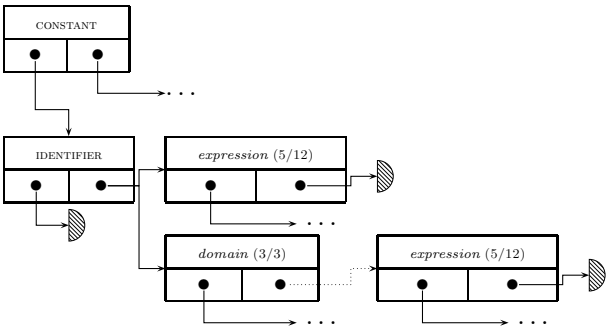
constant-definition ::= *cst-definition*

cst-definition ::= `CONST` *constant-name* `=` *constant-expression*
::= `CONST` *constant-name* `:` *domain* `=` *constant-expression*
::= `CONST` *constant-name* `:` *domain*

constant-name ::= *identifier*

constant-expression ::= *expression*

identifier ::= *RegExp(*`[a-zA-Z_][a-zA-Z_0-9]*`*)*



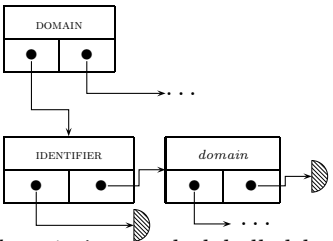
If the constant is declared using a domain specification then the integer field of the root node (i.e. labelled with CONSTANT) is set to 1.

3 Definitions of domains

domain-definition ::= `DOMAIN` *domain-name* `=` *domain*

domain-name ::= *identifier*

domain ::= *range-domain*
::= *enumeration-domain*
::= *predefined-domain*
::= *domain-name*
::= *structure-domain*
::= *array-domain*

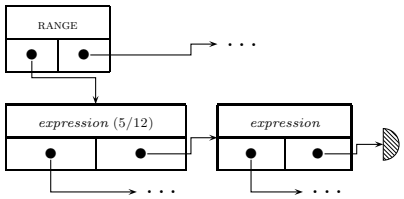


domain is a node labelled by one of the following labels: IDENTIFIER, RANGE (3.1/3), SYMBOL_SET (3.2/4), BOOLEANS (3.3/4), INTEGERS (3.3/4), STRUCTURE (3.4/4) or ARRAY_DOMAIN (3.5/4).

3.1 Integer ranges

range-domain ::= [*numerical-constant-expression* ,
numerical-constant-expression]

numerical-constant-expression ::= *constant-expression*

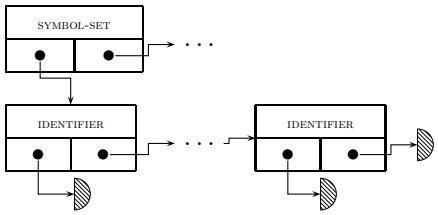


3.2 List of symbolic values (enumerations)

symbol-set-domain ::= { *symbol-list* }

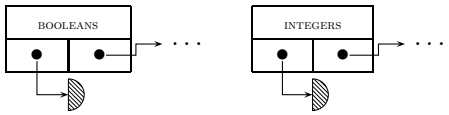
symbol-list ::= *symbol-list* , *symbol*
::= *symbol*

symbol ::= *identifier*



3.3 Predefined domains

predefined-domain ::= `BOOL`
::= `INTEGER`



3.4 Structured domains

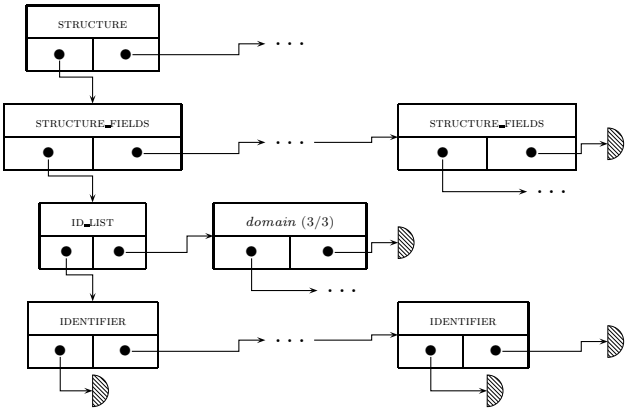
structure-domain ::= `STRUCT` *structure-fields* `;`$_{opt}$ `TCURTS`

structure-fields ::= *structure-fields* ; *field-list*
::= *field-list*

field-list ::= *id-list* : *domain*

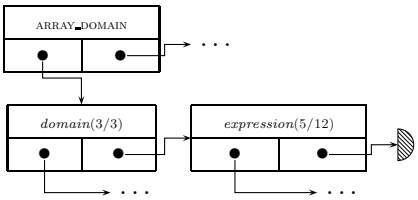
id-list ::= *list-of-identifiers*

list-of-identifiers ::= *list-of-identifiers* , *identifier*
::= *identifier*



3.5 Array domains

array-domain ::= *domain* `[` *numerical-constant-expression* `]`



4 AltaRica nodes

node-definition ::= `NODE` *node-name attributes node-field-list* `EDON`

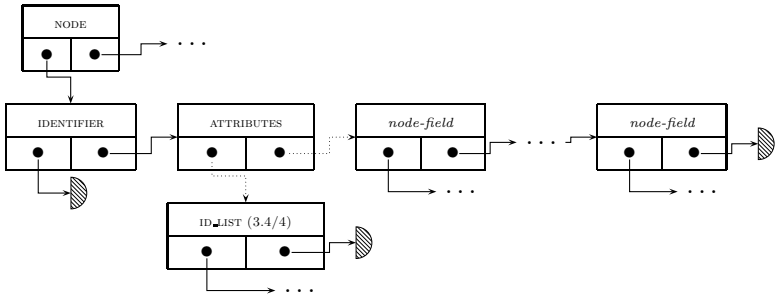
node-name ::= *identifier*

attributes ::= `:` *attribute-list*
::= ϵ

attribute-list ::= *id-list*

node-field-list ::= *node-field-list node-field*
::= ϵ

node-field ::= *parameters-declaration*
::= *variables-declaration*
::= *events-declaration*
::= *transitions-definition*
::= *assertions-definition*
::= *subnodes-declaration*
::= *vectors-definition*
::= *init-declaration*
::= *parameter-set-declaration*
::= *external-directives-declaration*



node-field is a node labelled by one of the following labels: PARAMETERS_DECL (4.1/6), VARIABLES_DECL (4.2/6), EVENTS_DECL (4.3/7), SUBNODES_DECL (4.4/8), ASSERTIONS_DEF (4.5/8), TRANSITIONS_DEF (4.6/9), SYNCHRONIZATION_DEF (4.7/10), INIT_DECL (4.8/11) and PARAM_SET_DECL (4.9/11). *external_directives* are ignored.

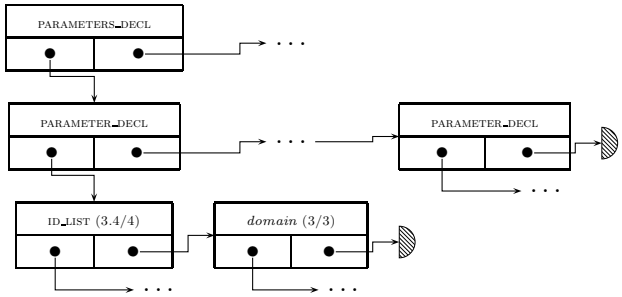
4.1 Declaration of parameters

parameters-declaration ::= **PARAM** *parameters-decl-list* **;**$_{opt}$

parameters-decl-list ::= *parameters-decl-list* **;** *parameter-decl*
::= *parameter-decl*

parameter-decl ::= *parameter-name-list* **:** *domain*

parameter-name-list ::= *id-list*



4.2 Declaration of flows and states

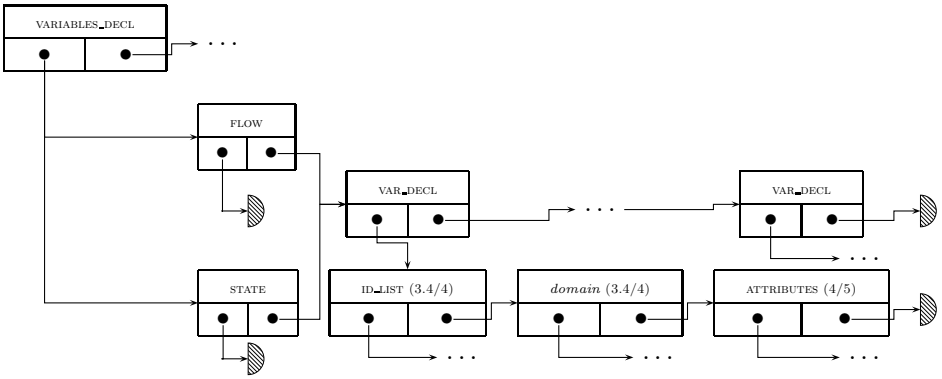
variables-declaration ::= *variable-class variable-decl-list* **;**$_{opt}$

variable-class ::= **FLOW**
::= **STATE**

variable-decl-list ::= *variable-decl-list* **;** *variable-decl*
::= *variable-decl*

variable-decl ::= *variable-name-list* **:** *domain attributes*

variable-name-list ::= *id-list*



4.3 Declaration of events

events-declaration ::= **EVENT** *events-decl-list* **;**$_{opt}$

events-decl-list ::= *events-decl-list* **;** *event-decl*
::= *event-decl*

event-decl ::= *event-dag-list attributes*

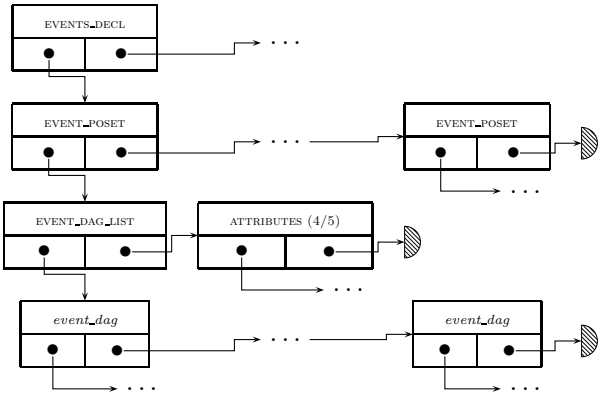
event-dag-list ::= *event-dag-list-rec*

event-dag-list-rec ::= *event-dag-list-rec* **,** *event-dag*
::= *event-dag*

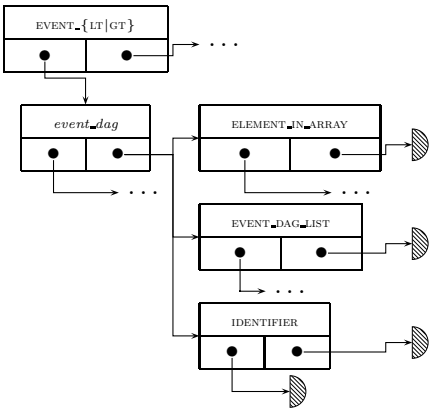
event-dag ::= *event-dag* **<** *event-set*
::= *event-dag* **>** *event-set*
::= *event-set*

event-set ::= { *event-dag-list* }
::= *event-name*

event-name ::= *ident*



event_dag is one of the following labels: IDENTIFIER, ELEMENT_IN_ARRAY (4.7/11), EVENT_LT, EVENT_GT or EVENT_DAG_LIST. EVENT_LT and EVENT_GT have the same structure:



4.4 Declaration of subnodes

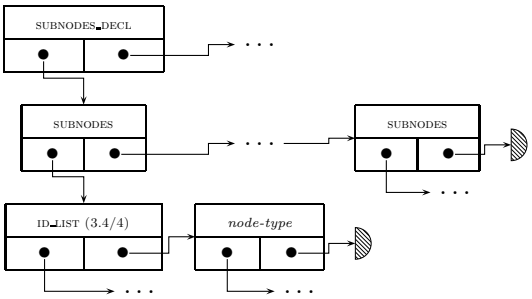
subnodes-declaration ::= `SUB` *subnodes-decl-list* `;`$_{opt}$

subnodes-decl-list ::= *subnodes-decl-list* `;` *subnodes-decl*
::= *subnodes-decl*

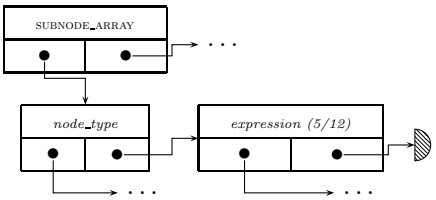
subnodes-decl ::= *subnode-name-list* `:` *node-type*

subnode-name-list ::= *id-list*

node-type ::= *node-type* `[` *numerical-constant-expression* `]`
::= *node-name*



node_type is a node labelled by IDENTIFIER or by SUBNODE_ARRAY.

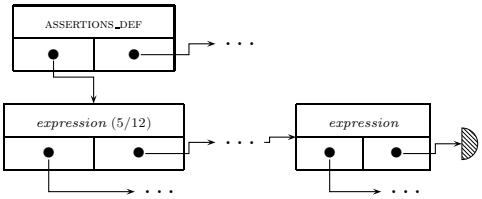


4.5 Definition of assertions

assertions-definition ::= `ASSERT` *assertion-list* `;`$_{opt}$

assertion-list ::= *assertion-list* `;` *assertion*
::= *assertion*

assertion ::= *boolean-expr*



4.6 Definition of transitions

transitions-definition ::= `TRANS` *transition-list* `;`$_{opt}$

transition-list ::= *transition-list* `;` *transition*
::= *transition*

transition ::= *boolean-expr transition-target-list*

transition-target-list ::= *transition-target-list transition-target*
::= *transition-target*

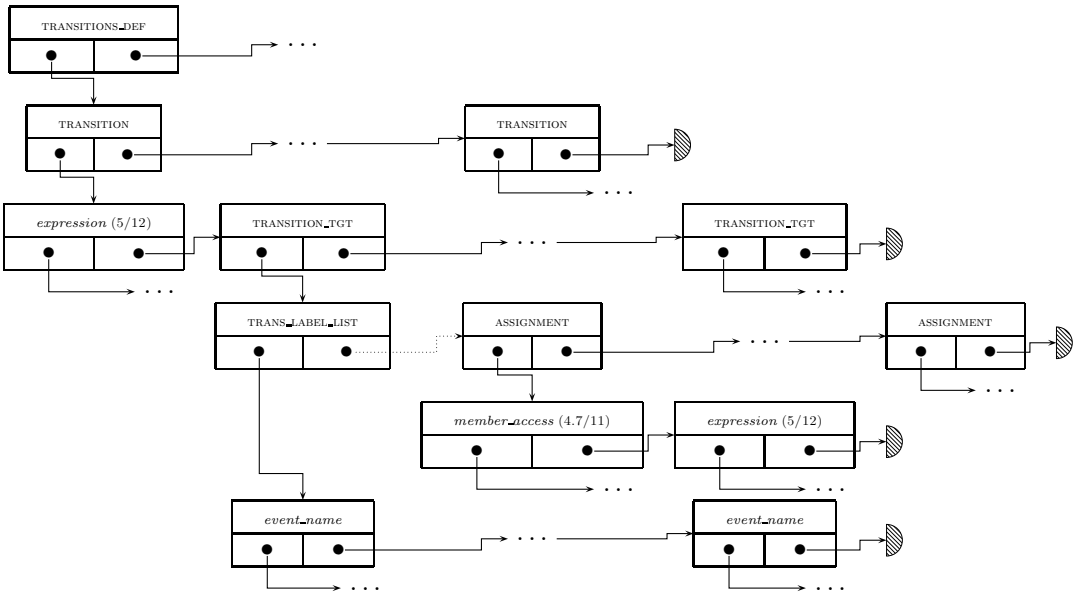
transition-target ::= `|-` *trans-label-list* `->` assignment-list$_{opt}$

trans-label-list ::= *event-name-list*

event-name-list ::= *event-name-list* `,` *event-name*
::= *event-name*

assignment-list ::= *assignment-list* `,` *assignment*
::= *assignment*

assignment ::= *member-access* `:=` *expression*



event_name denotes a node labelled either by IDENTIFIER or by ELEMENT_IN_ARRAY (4.7/11).

4.7 Broadcast synchronization vectors

vectors-definition ::= `SYNC` *vector-list* `;`$_{opt}$

vector-list ::= *vector-list* `;` *vector*
::= *vector*

vector ::= `<` *broadcast-list* `>` *constraint-on-vector broadcast-kind*

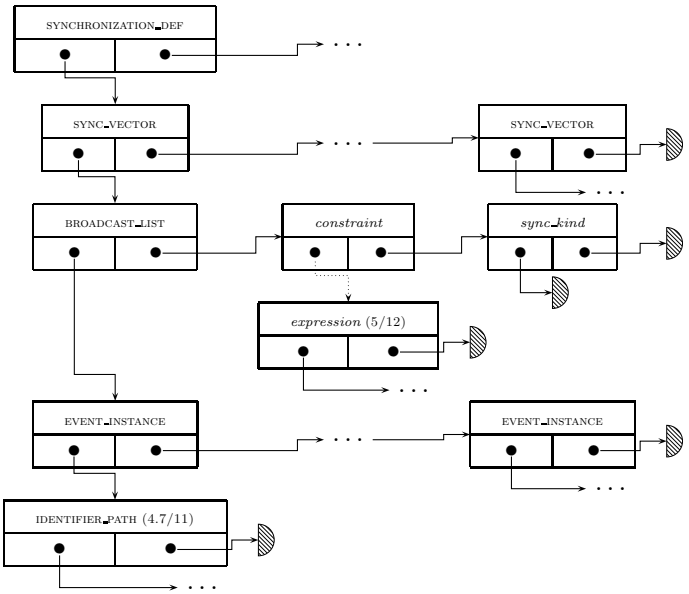
broadcast-list ::= *broadcast-list* `,` *broadcast*
::= *broadcast*

broadcast ::= *event-instance-identifier* `?`
::= `?` *event-instance-identifier*
::= *event-instance-identifier*

event-instance-identifier ::= *identifier-path*

constraint-on-vector ::= `<` *numerical-constant-expression*
::= `<=` *numerical-constant-expression*
::= `>` *numerical-constant-expression*
::= `>=` *numerical-constant-expression*
::= `=` *numerical-constant-expression*
::= ϵ

broadcast-kind ::= `MIN`
::= `MAX`
::= ϵ



sync_kind is either SYNC_MIN or SYNC_MAX. *constraint* is one of the following labels: SYNC_CONSTRAINT_{LT|LEQ|GT|GEQ|EQ|NONE}. In the case of SYNC_CONSTRAINT_NONE, the child node is NULL. When an event is marked, the corresponding node labelled by EVENT_INSTANCE is also valued with the integer 1.

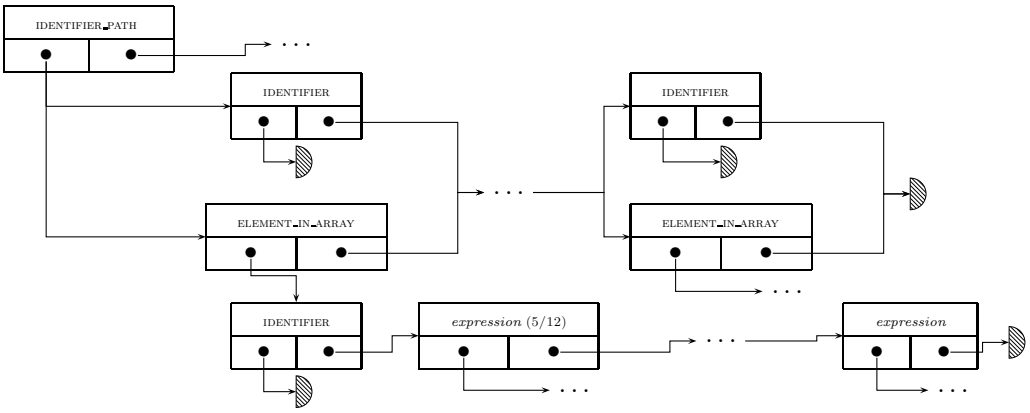
identifier-path ::= *dot-separated-identifier-list*

dot-separated-identifier-list ::= *dot-separated-identifier-list* `.` *ident*
::= *ident*

ident ::= *identifier*
::= *identifier position-in-array*

position-in-array ::= *list-of-array-position*

list-of-array-position ::= *list-of-array-position* `[` *numerical-constant-expression* `]`
::= `[` *numerical-constant-expression* `]`

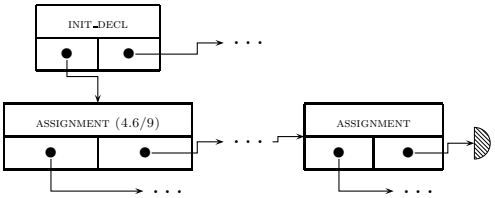


4.8 Initialization of state variables

init-declaration ::= `INIT` *init-assignment-list* `;`$_{opt}$

init-assignment-list ::= *init-assignment-list* `,` *init-assignment*
::= *init-assignment*

init-assignment ::= *assignment*

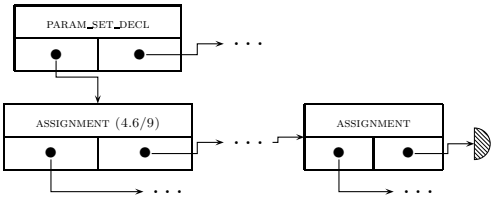


4.9 Parameter settings

parameter-set-declaration ::= `PARAM_SET` *param-assignment-list* `;`$_{opt}$

param-assignment-list ::= *param-assignment-list* `,` *param-assignment*
::= *param-assignment*

param-assignment ::= *assignment*



4.10 External directives

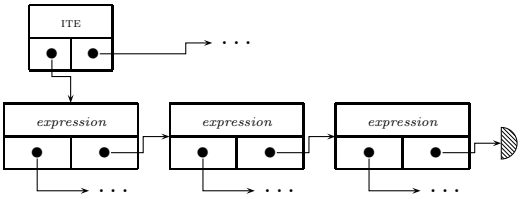
external-directives-declaration ::= `EXTERN ;`$_{opt}$

5 AltaRica expressions

expression ::= *conditional-expr*

conditional-expr ::= *if-then-else-expr*
::= *case-expr*
::= *disjunctive-expr*

if-then-else-expr ::= `IF` *expression* `THEN` *expression* `ELSE` *expression*
::= `(` expression `?` *expression* `:` *expression* `)`



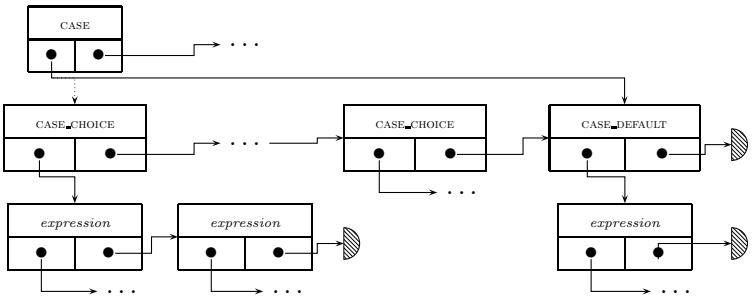
case-expr ::= `CASE` { *case-choice-list* }

case-choice-list ::= *case-conditional-choice-list* `,` *case-default*
::= *case-default*

case-conditional-choice-list ::= *case-conditional-choice-list* `,` *case-conditional-choice*
::= *case-conditional-choice*

case-default ::= `ELSE` *expression*

case-conditional-choice ::= *boolean-expr* `:` *expression*



boolean-expr ::= *expression*

disjunctive-expr ::= *disjunctive-expr* `OR` *conjunctive-expr*
::= *disjunctive-expr* `|` *conjunctive-expr*
::= *conjunctive-expr*

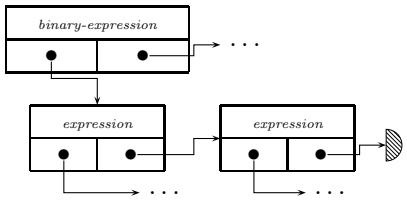
conjunctive-expr ::= *conjunctive-expr* `AND` *logical-comparison-expr*
::= *conjunctive-expr* `&` *logical-comparison-expr*
::= *logical-comparison-expr*

logical-comparison-expr ::= *logical-comparison-expr* `=` *arithmetic-comparison-expr*
::= *logical-comparison-expr* `!=` *arithmetic-comparison-expr*
::= *logical-comparison-expr* `=>` *arithmetic-comparison-expr*
::= *arithmetic-comparison-expr*

arithmetic-comparison-expr ::= *arithmetic-comparison-expr* `<` *additive-expr*
::= *arithmetic-comparison-expr* `>` *additive-expr*
::= *arithmetic-comparison-expr* `<=` *additive-expr*
::= *arithmetic-comparison-expr* `>=` *additive-expr*
::= *additive-expr*

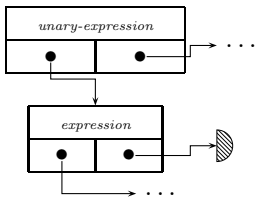
additive-expr ::= *additive-expr* `+` *multiplicative-expr*
::= *additive-expr* `-` *multiplicative-expr*
::= *multiplicative-expr*

multiplicative-expr ::= *multiplicative-expr* `*` *unary-expr*
::= *multiplicative-expr* `/` *unary-expr*
::= *multiplicative-expr* `MOD` *unary-expr*
::= *unary-expr*



binary-expression can be one of the following label: OR, AND, EQ, NEQ, IMPLY, LT, LEQ, GT, GEQ, ADD, SUB, MUL, DIV or MOD.

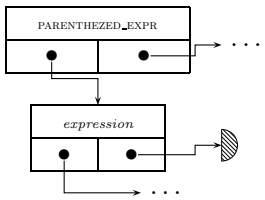
unary-expr ::= `-` *unary-expr*
::= `~` *unary-expr*
::= `NOT` *unary-expr*
::= *atomic-expr*



unary-expression is either the label NEG or NOT.

atomic-expr ::= *parenthezed-expr*
::= *member-access*
::= *min-max-expr*
::= *unsigned-integer*
::= *boolean-constant*
::= *quantified-expr*
::= *function-call-expr*
::= *constant-structure*
::= *constant-array*

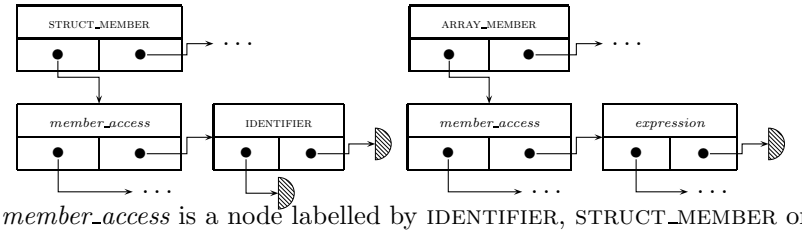
parenthezed-expr ::= `(` *expression* `)`



member-access ::= *struct-member*
::= *array-member*
::= *identifier*

struct-member ::= *member-access* `.` *identifier*

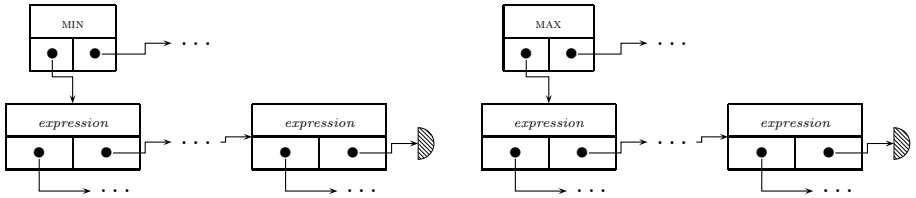
array-member ::= *member-access* `[` *numerical-constant-expression* `]`



member_access is a node labelled by IDENTIFIER, STRUCT_MEMBER or ARRAY_MEMBER.

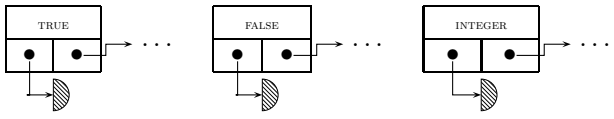
min-max-expr ::= `MIN` `(` *comma-separated-expr-list* `)`
::= `MAX` `(` *comma-separated-expr-list* `)`

comma-separated-expr-list ::= *comma-separated-expr-list* `,` *expression*
::= *expression*



boolean-constant ::= `TRUE`
::= `FALSE`

unsigned-integer ::= *RegExp(*`[1-9][0-9]*)|0`*)*



quantified-expr ::= `<` *quantified-variables-decl* `>` *unary-expr*
::= `[` *quantified-variables-decl* `]` *unary-expr*

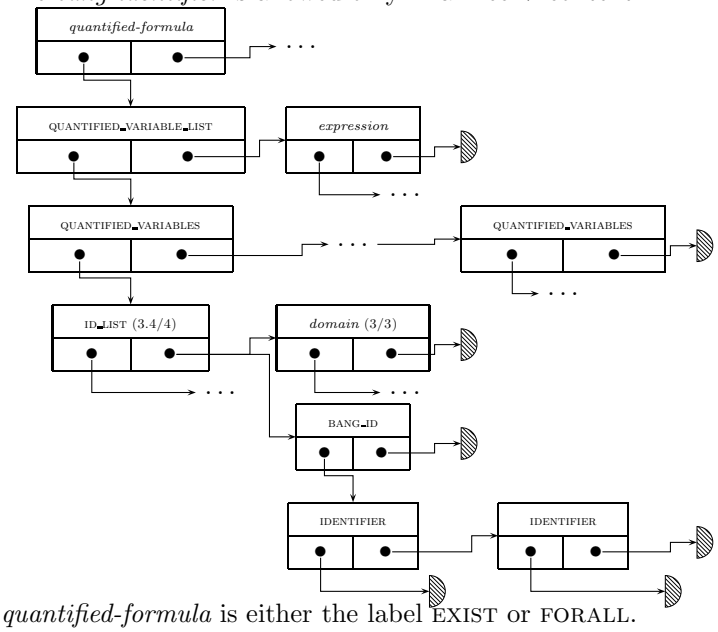
quantified-variables-decl ::= *quantified-variables-decl-list* $;_{opt}$

quantified-variables-decl-list ::= *quantified-variables-decl-list* `;` *quantified-variables*
::= *quantified-variables*

quantified-variables ::= *id-list* `:` *domain*
::= *id-list* `:` *bang-identifier*

bang-identifier ::= *identifier* `!` *identifier*

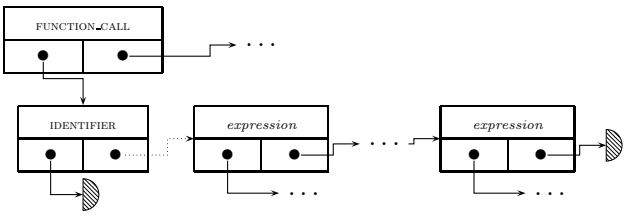
The *bang-identifier* is allowed only in a Mec V context.



quantified-formula is either the label EXIST or FORALL.

function-call-expr ::= *identifier* `(` $function\text{-}parameters_{opt}$ `)`

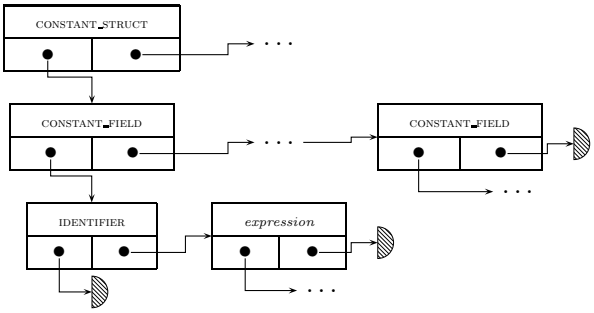
function-parameters ::= *comma-separated-expr-list*



constant-structure ::= `{` *const-field-assignment-list* `}`

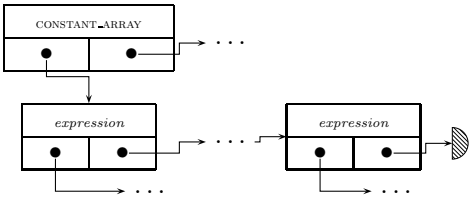
const-field-assignment-list ::= *const-field-assignment-list* `,` *const-field-assignment*
::= *const-field-assignment*

const-field-assignment ::= `.` *identifier* `=` *constant-expression*



constant-array ::= { *constant-expression-list* }

constant-expression-list ::= *constant-expression-list* , *constant-expression*
::= *constant-expression*



6 Abstract type definitions

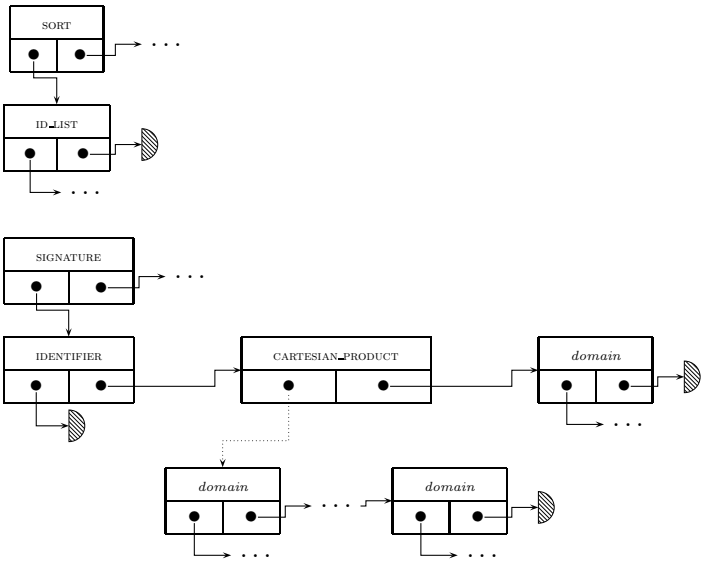
sort-declaration ::= `SORT` *id-list*

signature-declaration ::= `SIG` *operator-name* `:` *cartesian-product* `->` *domain*

operator-name ::= *identifier*

cartesian-product ::= *cartesian-product-list*
::= ϵ

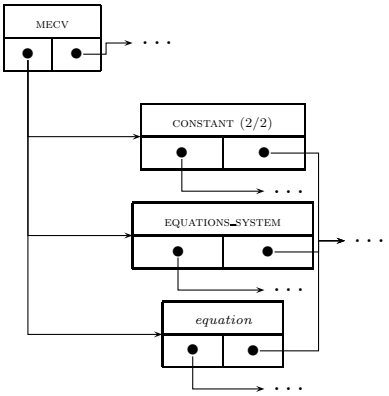
cartesian-product-list ::= *cartesian-product-list* `*` *domain*
::= *domain*



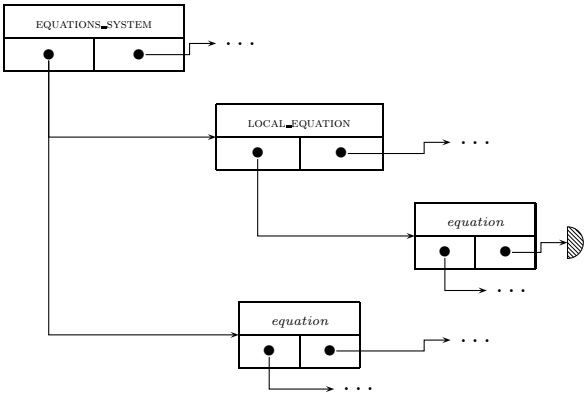
A Appendix: Syntax of Mec V specifications

mecv-specification ::= *mecv-definition mecv-specification*
::= *mecv-definition*

mecv-definition ::= *cst-definition* ;
::= `BEGIN` *equations-system* `END`
::= *equation* ;



equations-system ::= `LOCAL` *equation* ; *equations-system*
::= *equation* ; *equations-system*
::= ϵ



equation ::= *identifier* `(` *eq-parameters-list* `)` *fixpoint-op boolean-expr*

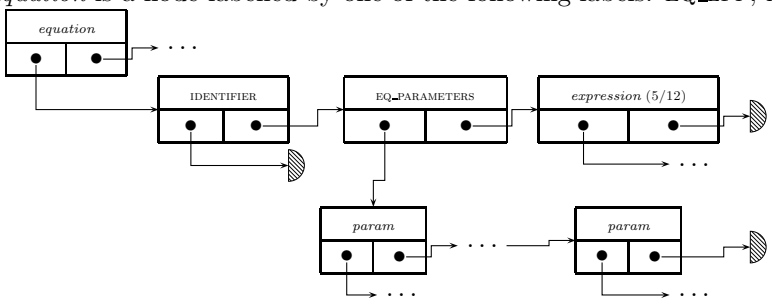
fixpoint-op ::= `+=`
::= `+` *unsigned-integer* `=`
::= `-=`
::= `-` *unsigned-integer* `=`
::= `:=`

eq-parameters-list ::= *eq-parameters*

eq-parameters ::= *identifier*
::= *typed-identifier*
::= *identifier* `,` *eq-parameters*
::= *typed-identifier* `,` *eq-parameters*

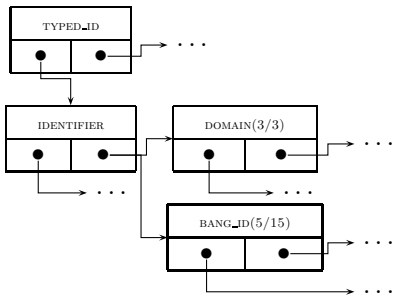
typed-identifier ::= *identifier* `:` *domain*
::= *identifier* `:` *bang-identifier*

equation is a node labelled by one of the following labels: EQ_LFP, EQ_GFP, or EQ_DEF.



If the label of an *equation* node is LFP or GFP then its integer field is set to 0 in the case of `+=` and `-=` equations or the fied is set to i in the case of `+`i`=` and `-`i`=` equations.

param is a node labelled by one of the following labels: IDENTIFIER or TYPED_ID.



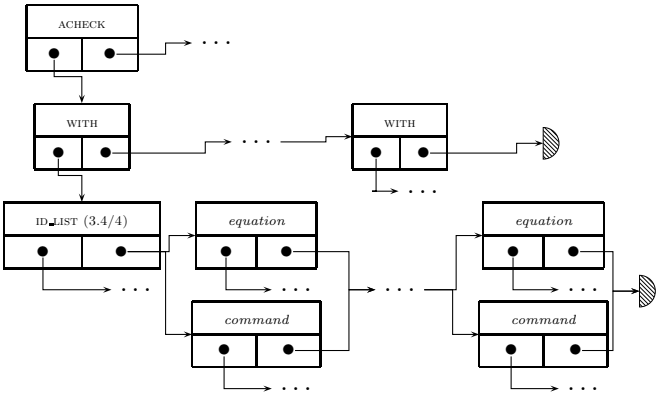
B Appendix: Syntax of Acheck specifications and commands

acheck-specification ::= *with-block acheck-specification*
::= *with-block*

with-block ::= `WITH` *id-list* `DO` *equation-or-command-list* `DONE`

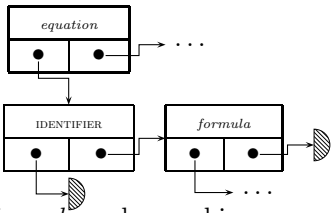
equation-or-command-list ::= *equation-or-command* `;` *equation-or-command-list*
::= *equation-or-command* `;`

equation-or-command ::= *equation*
::= *redirected-command*



equation ::= *identifier* `:=` *state-or-transition-formula*
::= *identifier* `+=` *state-or-transition-formula*
::= *identifier* `-=` *state-or-transition-formula*

equation denotes a node labelled by one of the following label: EQ_LFP, EQ_GFP, or EQ_DEF.



formula nodes are binary or unary operations, an IDENTIFIER node or atomic (state or transition) formulas.

state-or-transition-formula ::= *st-or-expression*

state-formula ::= *st-or-expression*

transition-formula ::= *st-or-expression*

st-or-expression ::= *st-or-expression* `OR` *st-and-expression*
::= *st-or-expression* `|` *st-and-expression*
::= *st-and-expression*

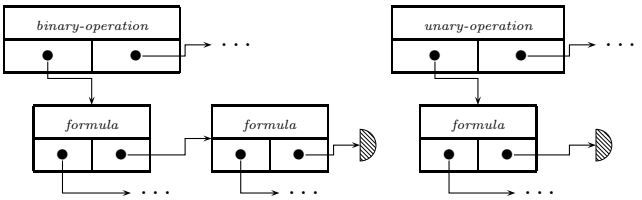
st-and-expression ::= *st-and-expression* `AND` *st-sub-expression*
::= *st-and-expression* `&` *st-sub-expression*

st-sub-expression ::= *st-sub-expression* `-` *st-unary-expression*
::= *st-unary-expression*

st-unary-expression ::= `~` *st-unary-expression*
::= `NOT` st-unary-expression
::= *st-atomic-expression*

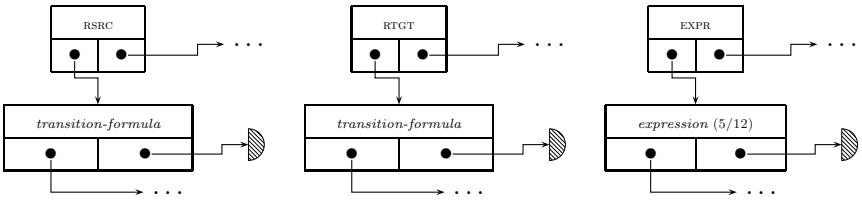
st-atomic-expression ::= `(` *state-or-transition-formula* `)`
::= *identifier*
::= *atomic-state-expression*
::= *atomic-transition-expression*

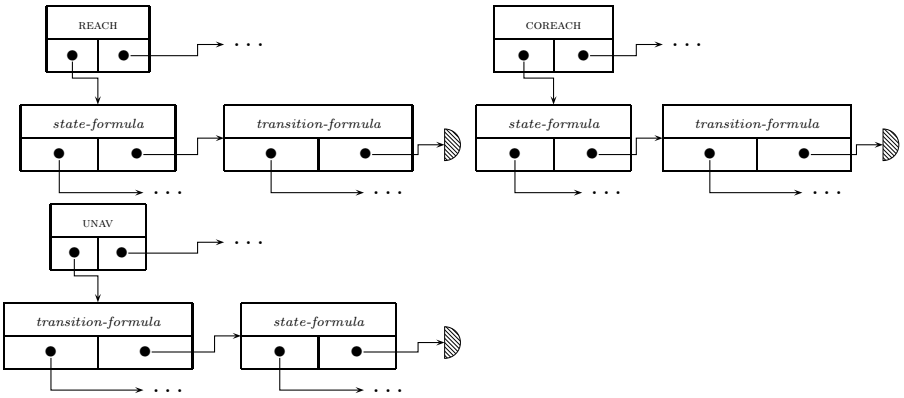
binary-operation are nodes labelled with OR, AND or SUB. *unary-operation* are nodes labelled with NOT or PARENTHEZED_EXPR.



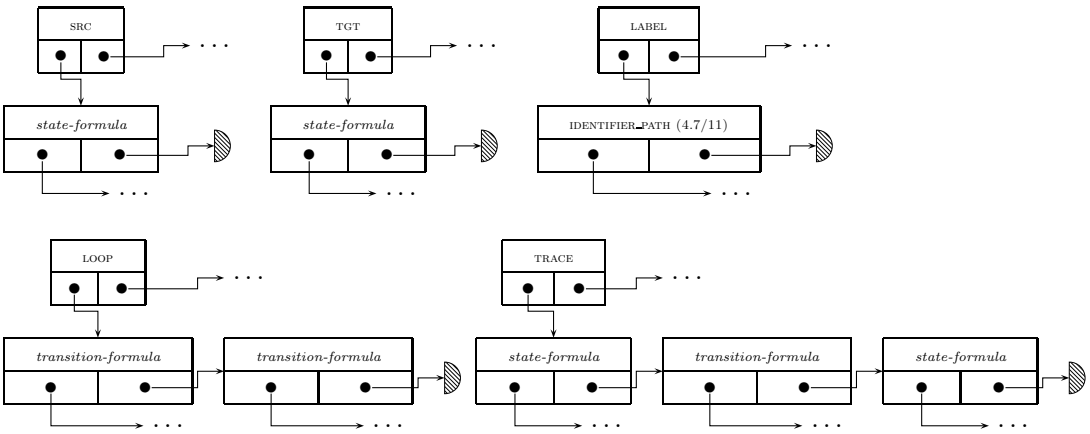
atomic-state-expression ::= `RTGT` `(` *transition-formula* `)`
::= `RSRC` `(` *transition-formula* `)`
::= `REACH` `(` *state-formula* `,` *transition-formula* `)`
::= `COREACH` `(` *state-formula* `,` *transition-formula* `)`
::= `[` *boolean-expr* `]`
::= `UNAV` `(` *transition-formula* `,` *state-formula* `)`

A *state-formula* (resp. *transition-formula*) is a *formula* with leaves labelled by RTGT, RSRC, REACH, COREACH, EXPR or UNAV (resp. SRC, TGT, LOOP, TRACE or LABEL).

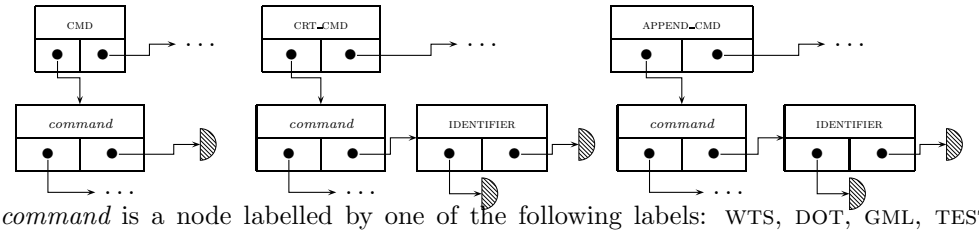




atomic-transition-expression ::= `TGT` (*state-formula*)
::= `SRC` (*state-formula*)
::= `LOOP` (*transition-formula* , *transition-formula*)
::= `TRACE` (*state-formula* , *transition-formula* , *state-formula*)
::= `LABEL` *identifier-path*



redirected-command ::= *command*
::= *command* `>` *identifier*
::= *command* `>>` *identifier*



command is a node labelled by one of the following labels: WTS, DOT, GML, TEST, SHOW, QUOT, PROJECT.

command ::= `WTS` (*state-formula* , *transition-formula*)
::= `DOT` (*state-formula* , *transition-formula*)
::= `GML` (*state-formula* , *transition-formula*)
::= `TEST` (*state-or-transition-formula* , *unsigned-integer*)
::= `SHOW` (*id-list*)
::= `QUOT` ( )
::= `PROJECT` (*state-formula* , *transition-formula* , *identifier* , *boolean-constant* , *identifier-path*)
::= `PROJECT` (*state-formula* , *transition-formula* , *identifier* , *boolean-constant*)