```
(*  EBNF-style grammar of Checker  *)

(*  proof is the main symbol  *)

proof ::= { prove | permanent-definition |
            local-assumptions | permanent-assumptions }

prove    ::= 'PROVE' [ label":" ] formula
             'FROM' references)

permanent-definition   ::=  'DEFINE' define-predicate |
                            'DEFINE' define-function-by-predicate |
                            'DEFINE' define-function-by-term

local-assumptions     ::= 'ASSUME' assumptions 'IN' subproof 'END'

permanent-assumptions ::= 'ASSUME' declarations-list


subproof ::= { prove | local-assumptions | permanent-definition }



declarations-list                ::=  declaration { ","declaration }

declaration                      ::=  function-declaration  |
                                      predicate-declaration |
                                      formula-declaration


function-declaration             ::= 'FUNCTION' identifier ":" arity

predicate-declaration            ::= 'PREDICATE' identifier ":" arity

formula-declaration              ::= 'FORMULA' [ label":"] formula


assumptions       ::= assumption { "," assumption }

assumption        ::= declaration |
                      assume-defined-predicate |
                      assume-defined-function


define-predicate             ::= 'PREDICATE' identifier 'BY'
                                  [ label":" ]
                                  lambda-bound-vars formula

define-function-by-predicate ::= 'FUNCTION' identifier 'INDIRECTLY BY'
                                  [ label":" ]
                                  lambda-bound-vars formula
                                  'EXISTENCE'  references
                                  'UNIQUENESS' references

define-function-by-term      ::= 'FUNCTION' identifier 'BY'
                                  lambda-bound-vars [ label":"] term


assume-defined-predicate             ::= 'PREDICATE' identifier ':='
                                         lambda-bound-vars
                                         [ label":"] formula

assume-defined-function              ::= 'FUNCTION' identifier ':='
                                         lambda-bound-vars
                                         [ label":"] term


reference       ::=   "-" natural-number |
                      label [ "+" natural-number | "-" natural-number ] |
                      reference "[" instantiations "]" |
                      'MP' '(' reference "," reference ')' |
                      '(' reference ')'

references      ::=   [ reference { ',' reference } ]
```

```
instantiation   ::=   identifier ':=' 'FUNCTION'  lambda-bound-vars term |
                      identifier ':=' 'PREDICATE' lambda-bound-vars formula

instantiations  ::=   instantiation { "," instantiation}


lambda-bound-vars ::= [ LAMBDA identifier { "," identifier} ]

identifier        ::= ? string of alphanumerical characters and "_"
                        not starting from a digit ?

label             ::= identifier

arity             ::= ? natural number ?



(*  General formulas with possibly schematic quantifications.         *)
(*  In Checker we only handle schematic formulas where all schematic  *)
(*  quantifications occur in prefix.                                  *)



(*  As usual /\ has highest precedence, \/ -lower than /\,            *)
(*  -> and <-> the lowest - so conflicts must be solved               *)
(*  by parentheses                                                    *)
formula  ::= formula '/\' formula | formula '\/' formula  |
             formula '->' formula | formula '<->' formula |
             basic-formula

basic-formula                    ::=  quantified-formula  |
                                      "!" formula         |
                                      atomic-formula

quantified-formula               ::=  existentially-quantified-formula |
                                      universally-quantified-formula

existentially-quantified-formula ::=  "<" boundvar {"," boundvar} ">" formula

universally-quantified-formula   ::=  "[" boundvar {"," boundvar} "]" formula

(*  In boundvar predicate-declaration, and function-declaration          *)
(*  where arity > 0 are schematic variables, function-declaration        *)
(*  where arity = 0 and identifier are first order variables.            *)

boundvar  ::=  function-declaration |  predicate-declaration|  identifier


atomic-formula                   ::=  "(" formula ")" | 'FALSE' | 'TRUE' |
                                      atom | term "=" term

term                             ::= identifier [ "(" term-list ")" ]

atom                             ::= identifier [ "(" term-list ")" ]

term-list                        ::= term {"," term}
```