Specification Language Semantics

Fig. 1 summarizes the syntax of the Vigor specification language.

- Import subspec (l.3): Include the file (add the '.py' extension to the module name) as if it were a part of current specification (like C preprocessor `#include`)
- Import state variable (l.4): Declare that the present specification uses a state variable, that is declared in the NF state declaration. The variable that enters the context will have the same name and type, specified in the NF declaration. The variable will be initialized to the value at the beginning of the packet processing iteration. The variable is mutable, and the final mutated value will be compared against the computed value at the end of the iteration.
- Constant declaration (l.7): Declare the constant with the given value (like C preprocessor `#define`).
- If statement (ll.14-16): A regular Python `if` statement with optional `else` branch. If the condition holds it executes the bloc that immediately follows, otherwise the block that follows the `else` keyword, if present. The block is identified by its indentation.
- Assignment (l.17): Creates or updates a variable on the left hand side and assigns it the value from right hand side of '='.
- Pop header (l.18): Parses a header of the protocol that is specified as the first element, and stores it in the variable on the left hand side of '='.
- Assert (l.20): Evaluates an expression and checks if it is equivalent to `True`
- Return (l.21): Accepts a tuple of two lists - "interfaces", "headers". The first list "interfaces" enumerates all the interfaces where the NF has to send a packet. The second list "headers" is a stack of headers (from the outermost, to the innermost protocol) in the outcast packet. A header is set using the constructor syntax (l.49)
- Pass (l.23): "Immediately succeeds" the verification. Used to discard the behaviors and circumstances, not interesting for the property. Forces the verification engine to finish verification and report success.
- Expressions (ll.25-42): The standard set of arithmetic and boolean expressions, comparisons, lists, and a special ellipsis symbol "..." used to designate ignored parts in assertions. For example, when used as a value of a field in a constructor, it means that the outcast packet may have any value at that place.
- Call (ll.46-47): A function or a method call. Only the predefined functions can be called. The function name can not be indirect.
- Attribute (l.48): Some objects, such as packet headers and state have a predefined set of attributes, accessible through the "." syntax.
- Constructor (ll.49-51): Currently defined only for packet headers, a constructor of an object, initializing all its fields. Constructors assign fields as listed in the parentheses using the "=" signs (ll.52-53). The constructor accepts an optional argument — an existing object of the same type (l.50). It will copy the values of all the fields from the given object and override them with the values provided by explicit assignment arguments.

```
<spec>                    ::= <imports> <constant-decls> <block>
<imports                  ::= <import> <imports> | ""
<import>                  ::= "import" <python-symbol> "\n"
                            | "from" <python-symbol> "import" <symbols> "\n"
<symbols>                 ::= <python-symbol> | <python-symbol> "," <symbols>
<constant-decls>          ::= <constant-decl> | <constant-decl> <constant-decl>
<constant-decl>           ::= <python-symbol> "=" <expression> "\n"
<block>                   ::= <statements>
<statements>              ::= <tatement> | <tatement> <statements>
<statement>               ::= <if-statement> | <if-else-statement>
                            | <assignment-statement> | <pop-header-statement>
                            | <assert-statement> | <return-statement>
                            | <pass-statement>
<if-statement>            ::= "if" <expression> ":" "\n" "    " <block>
<if-else-statement>       ::= "if" <expression> ":" "\n" "    " <block> "\n"
                              "else: "\n" "    " <block>
<assignment-statement>    ::= <python-symbol> "=" <expression> "\n"
<pop-header-statement>    ::= <python-symbol> "=" "pop_header(" <python-symbol>
                                  "," "on_mismatch=" <expression> ")" "\n"
<assert-statement>        ::= "assert" <expression> "\n"
<return-statement>        ::= "return (" <list-expression> ","
                                          <list-expression> ")" "\n"
<pass-statement>          ::= "pass" "\n"
<expressions>             ::= <expression> | <expression> <expressions>
<expression>              ::= "(" <expression> ")"
                            | <aexpression> | <bexpression> | <term> | "..."
                            | <list-expression>
<list-expression>         ::= "[" <expressions> "]"
<aexpression>             ::= <aexpression> "/" <aexpression>
                            > <aexpression> "*" <aexpression>
                            > <aexpression> "-" <aexpression>
                            > <aexpression> "+" <aexpression>
<bexpression>             ::= <aexpression> "<" <aexpression>
                            | <aexpression> "<=" <aexpression>
                            | <aexpression> ">" <aexpression>
                            | <aexpression> ">=" <aexpression>
                            | <aexpression> "==" <aexpression>
                            | <aexpression> "!=" <aexpression>
                            | <aexpression> "<>" <aexpression>
                            > <bexpression> "and" <bexpression>
                            | <bexpression> "or" <bexpression>
                            > <bexpression> "not" <bexpression>
<term>                    ::= <python-symbol> | <literal>
                            | <call> | <attribute> | <constructor>
<literal>                 ::= NUMBER | True | False
<call>                    ::= <python-symbol> "(" <expressions> ")"
                            | <attribute> "(" <expressions> ")"
<attribute>               ::= <python-symbol> "." <python-symbol>
<constructor>             ::= <python-symbol> "(" <initializers> ")"
                            | <python-symbol> "(" <python-symbol> ","
                                                  <initializers> ")"
<initializers>            ::= <initializer> <initializers> | ""
<initializer>             ::= <python-symbol> "=" <expression>
```

Figure 1: BNF grammar of Vigor specification language