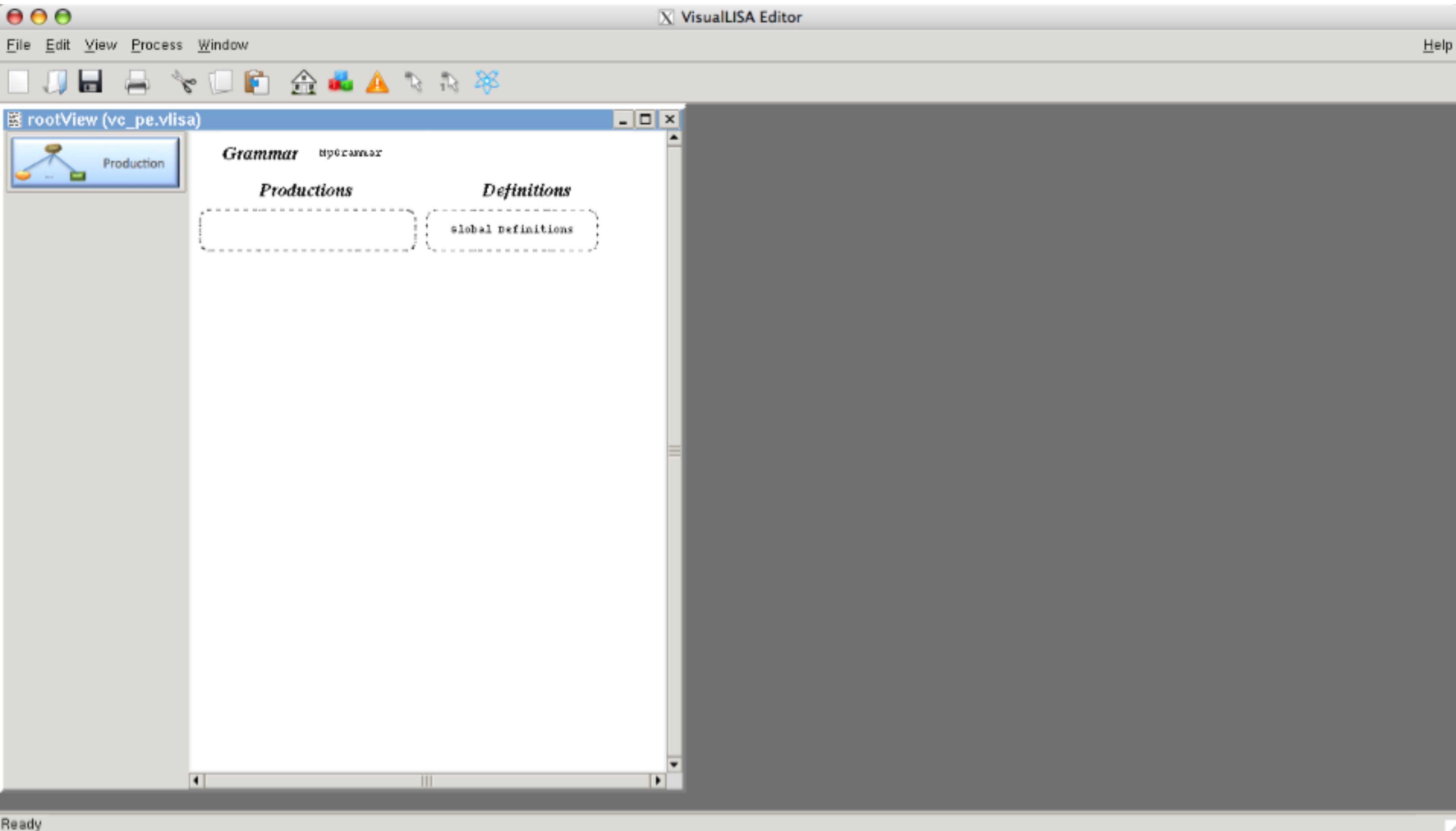
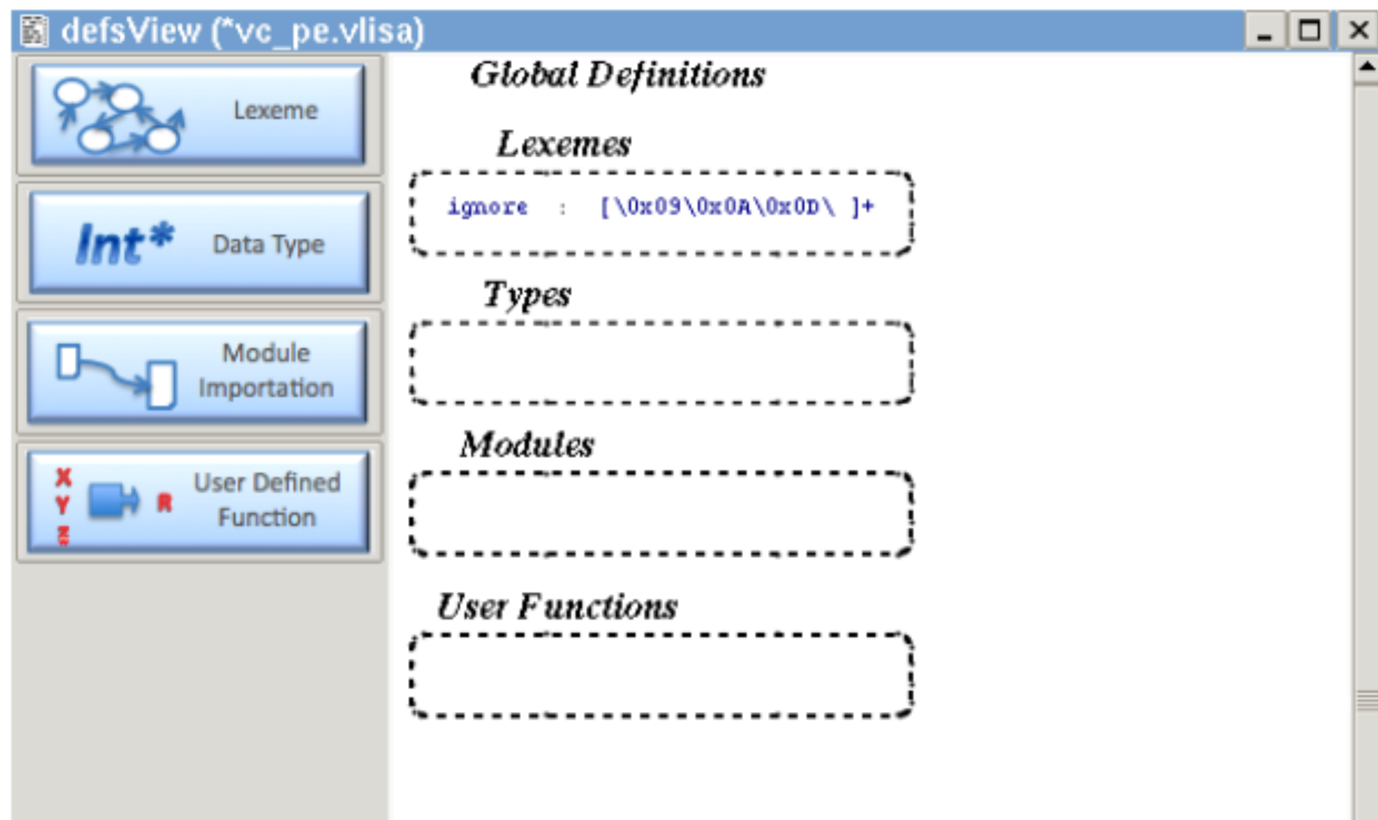
VisualLISA

A Quick and Simple Demonstration

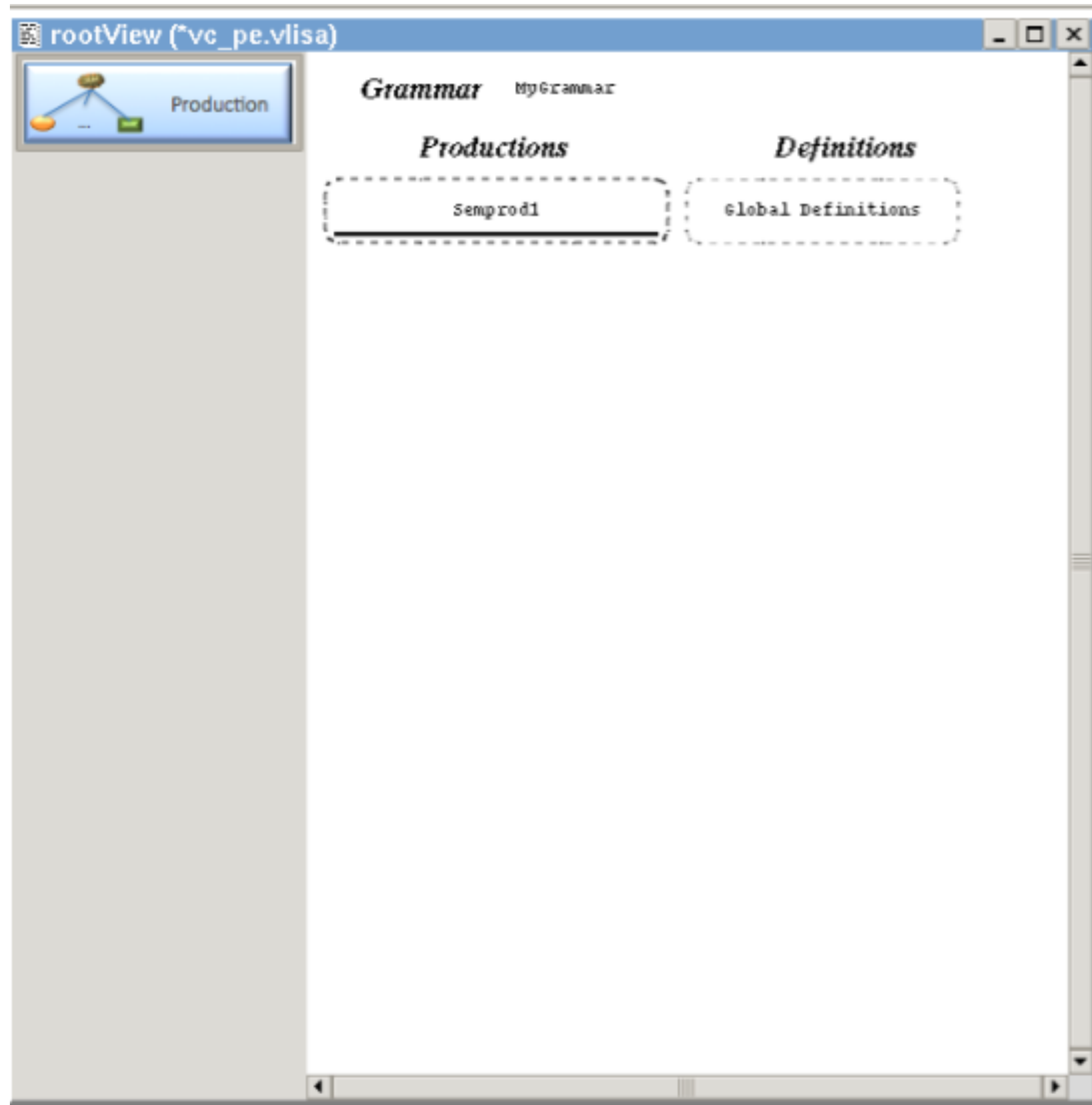
The default aspect of an attribute grammar specification on VisualLISA.
In this view we can add a new production to the grammar.



Global definitions for the grammar. Here we can add a new Lexeme or data type, import a new package or other kind of module, and define functions.

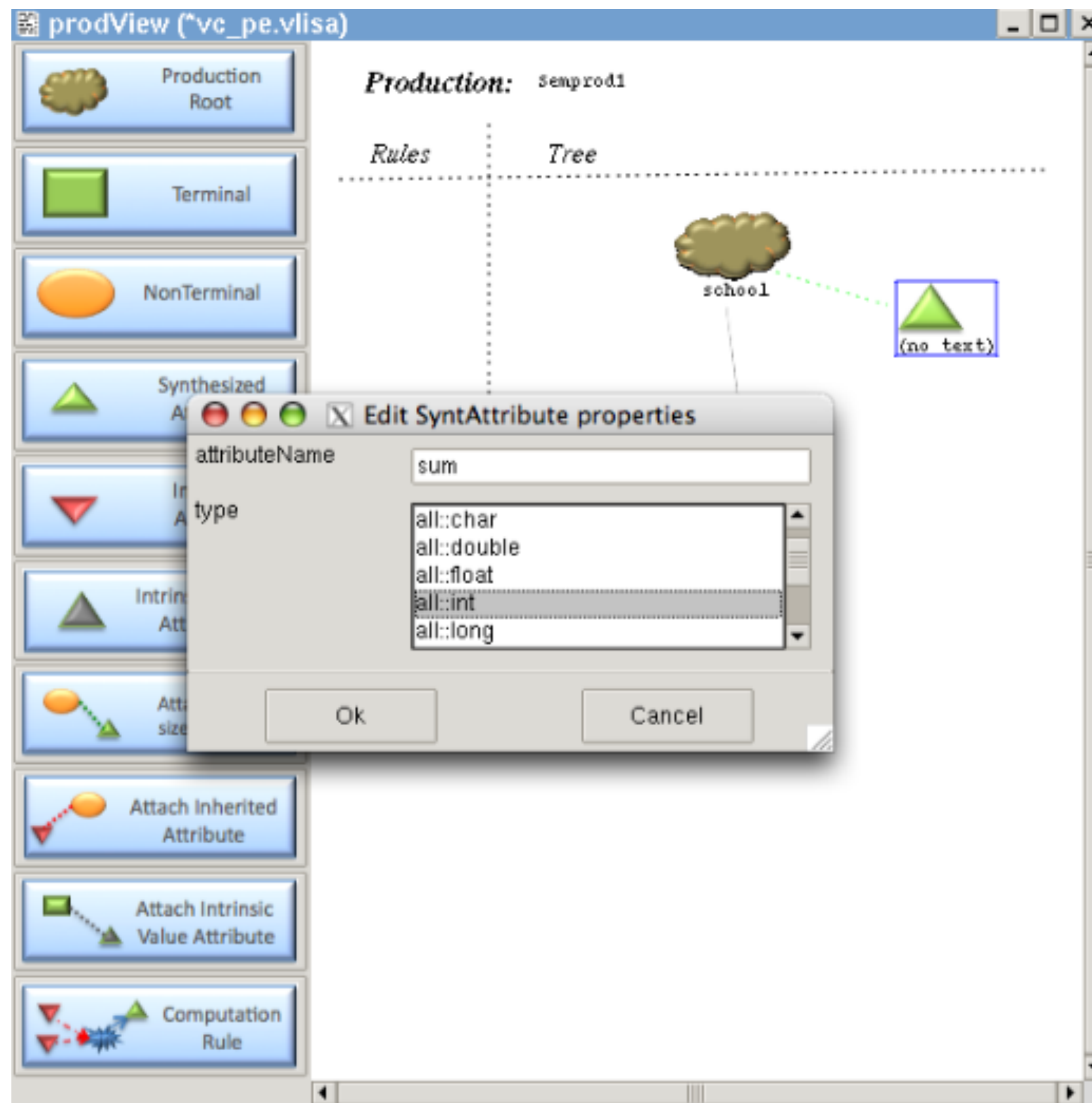


Adding a new Production. The addition of any symbol in the specification is done by clicking on the respective dock button, and click again on the drawing area.

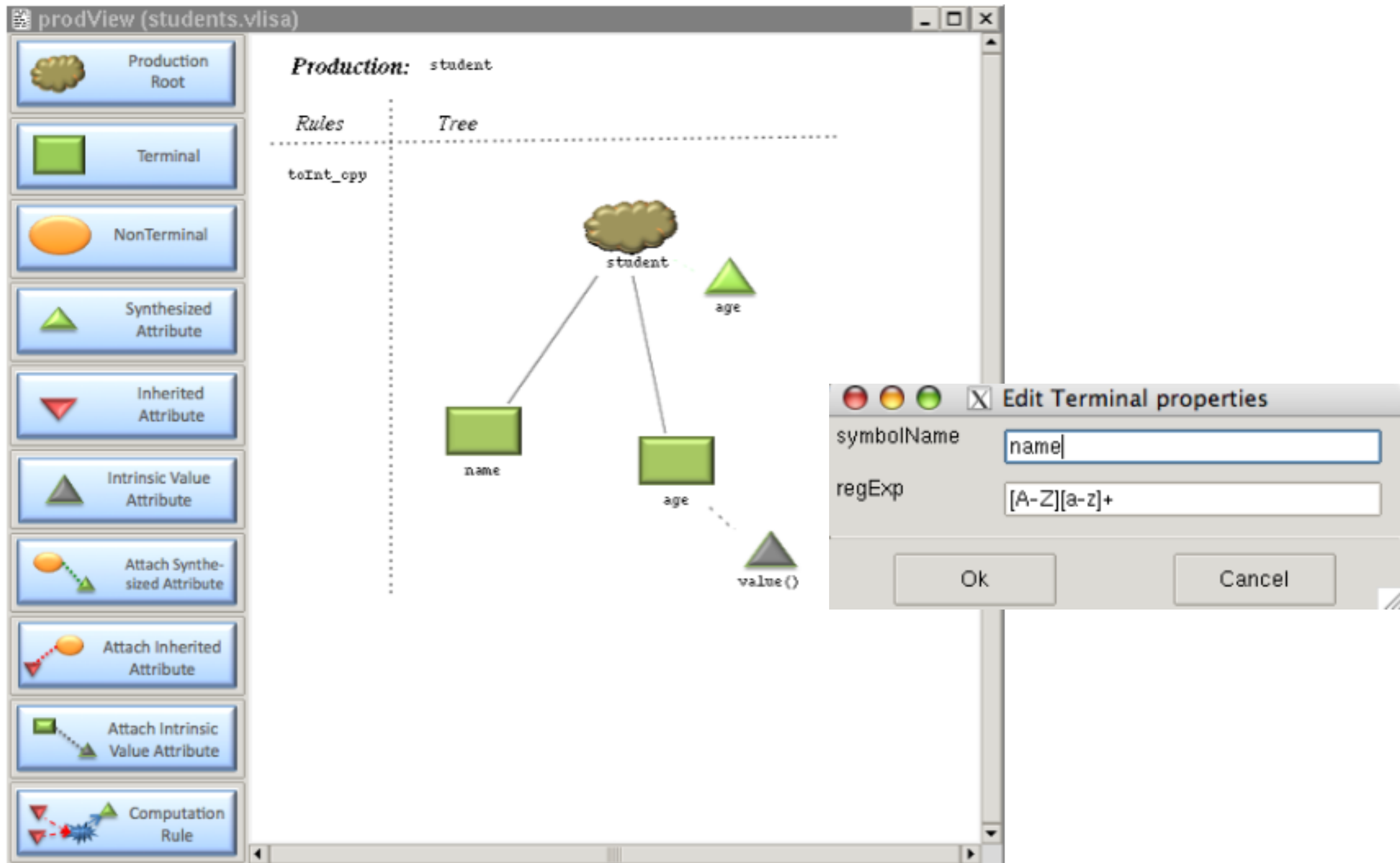


Building the production tree: The root of the production is a brown cloud. The Nonterminal are orange ovals and the Terminals are green rectangles. The tree is drawn automatically.

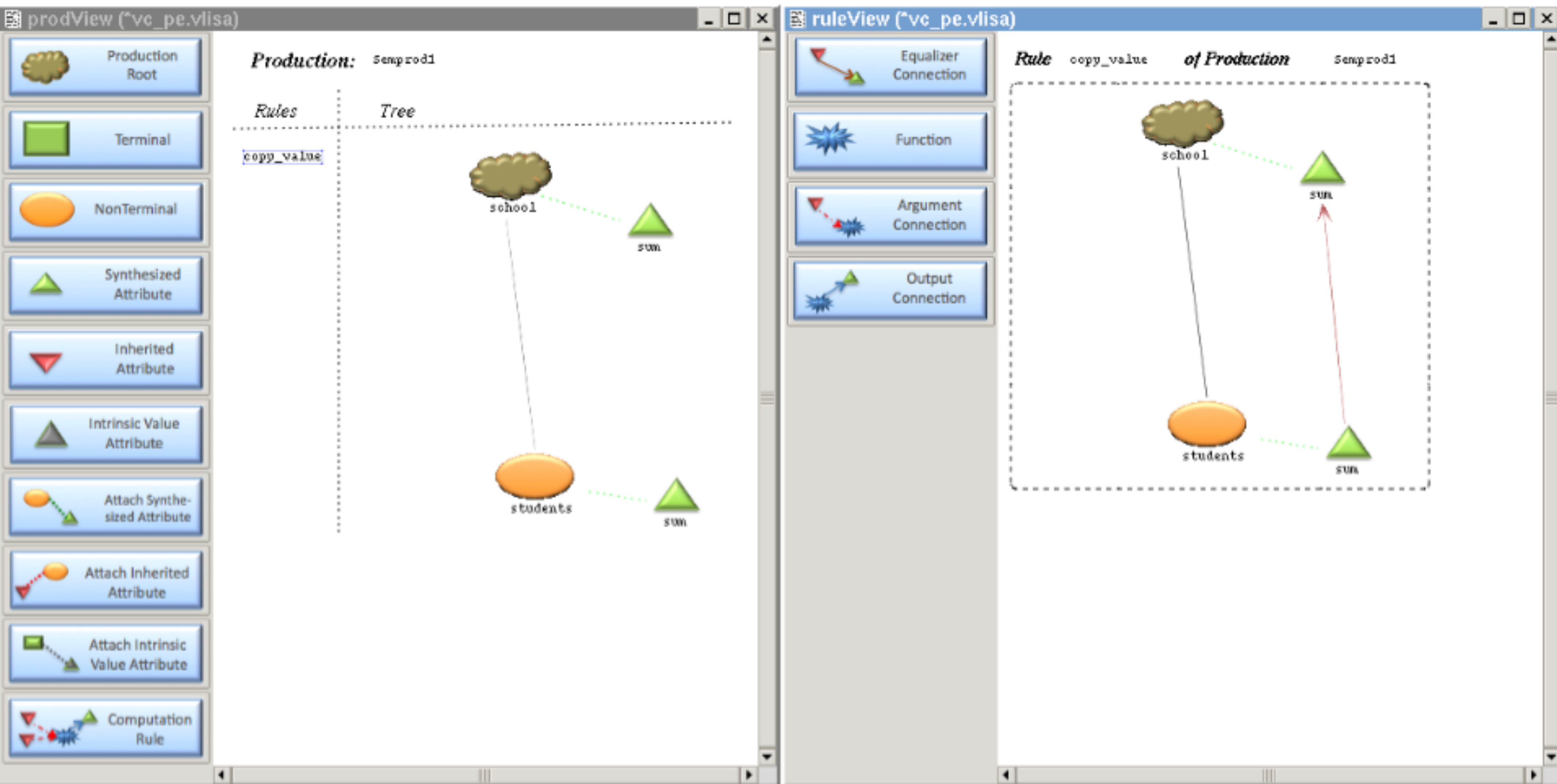
Adding an attribute to a symbol.
It is necessary to attach the attribute to the symbol (manually) and fill in the attribute’s form (attributeName and type)



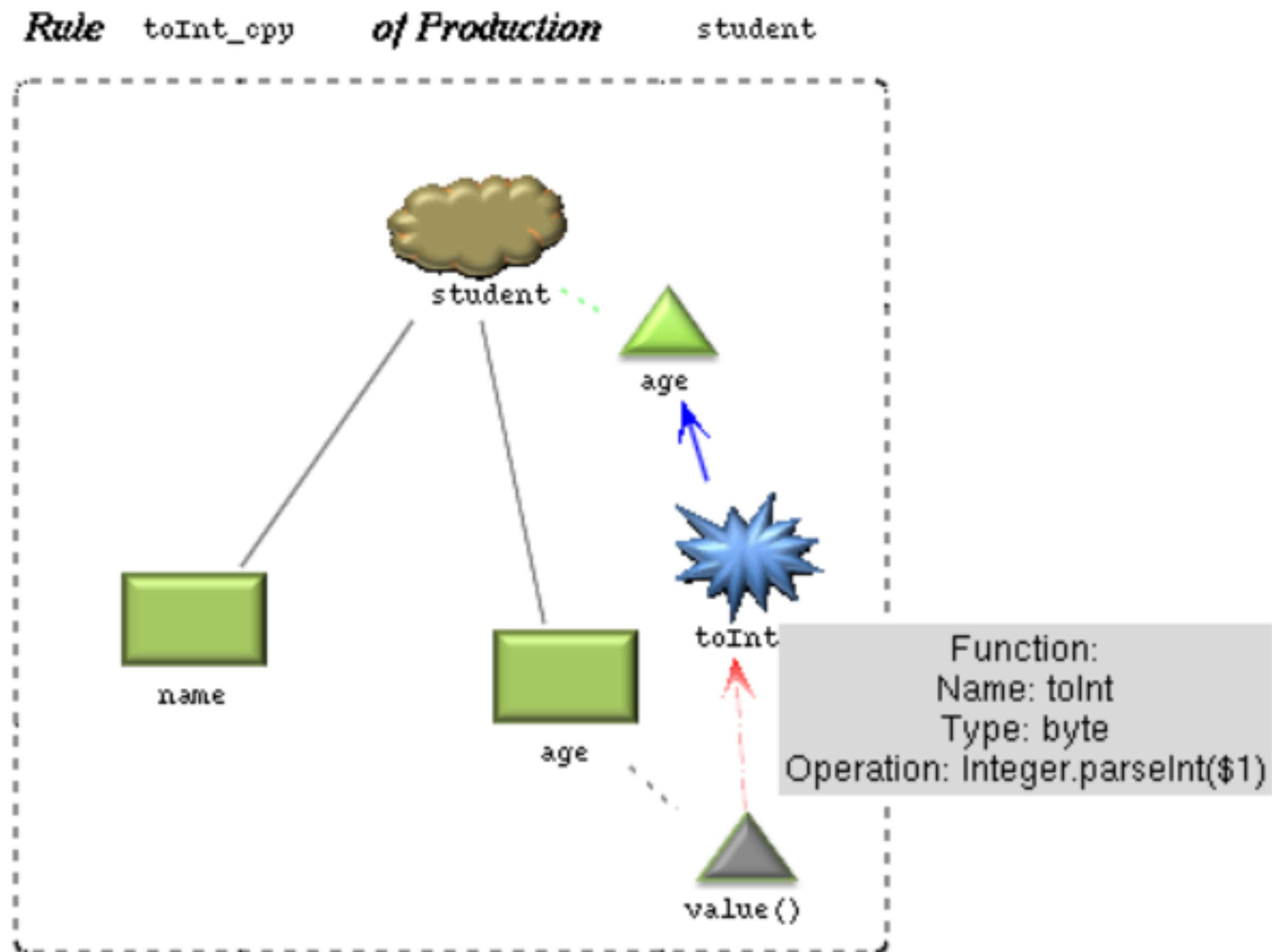
In this picture is possible to see the usage of terminals and the their intrinsic attributes.
A terminal has a name (mandatory) and a reg. exp. (optional).



Creating a computation rule. The base template of one production tree P is used for every Computation rules which belong to P.
In this example, we just copy the value of an attribute to another.



Other computation rule example:
In this case a function 'toInt', of type 'byte', is used to assign 'toInt(value)' to the attribute 'age', of type ‘int’.
The semantic analyzer, associated with the graphical editor, will detect and signal an error (type mismatch).
A Function has always one and only one output (blue arrow), but may have 0 or more arguments (red arrow).



Semantic errors detected on the grammar specification:
In the example below, the checker complains about the mismatched types of the function and the attribute which is the target of the function's output.

Each error line is a link to the context where the error occurs.

Check view (*students.vlisa)

ERROR: rules_Semprod '::obj1958119': Return type of Function 'toInt' and type of Attribute 'age' must match!

```
language schoolGra {

   lexicon{
      Name     [A-Z][a-z]+
      Age     [0-9]+
      ignore     [\0x09\0x0A\0x0D\ ]+
   }

   attributes
      int  SCHOOL.sum;
      int  STUDENTS.sum;
      int  STUDENT.age;


   rule school {
      SCHOOL ::=  STUDENTS compute {
         SCHOOL.sum = STUDENTS.sum;
      };
   }

   rule students_1 {
      STUDENTS ::=  STUDENT STUDENTS compute {
         STUDENTS.sum = STUDENT.age + STUDENTS[1].sum;
      };
   }

   rule students_2 {
      STUDENTS ::=  STUDENT compute {
         STUDENTS.sum = STUDENT.age;
      };
   }

   rule student {
      STUDENT ::=  #Name #Age compute {
         STUDENT.age = Integer.parseInt(#Age.value());
      };
   }
```

The direct translation of the Visual Grammar into a LISA Specification is one of the possible Code Generation features.

The translation into XML is another possibility.
The next XML example is not complete.
For the sake of space it was splitted into three parts

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
   <attributeGrammar name="schoolGra">
      <symbols>
         <terminals>
            <terminal id="name">[A-Z][a-z]+</terminal>
            <terminal id="age">[0-9]+</terminal>
            <terminal id="ignore">[\0x09\0x0A\0x0D\ ]+</terminal>
         </terminals>
         <nonterminals>
            <nonterminal id="school" />
            <nonterminal id="students" />
            <nonterminal id="student" />
         </nonterminals>
         <start nt="school">
      </symbols>
      <attributesDecl>
         <declaration symbol="school">
            <attribute id="sum" type="int" class="SyntAttribute" />
         </declaration>
         <declaration symbol="students">
            <attribute id="sum" type="int" class="SyntAttribute" />
         </declaration>
         <declaration symbol="student">
            <attribute id="age" type="int" class="SyntAttribute" />
         </declaration>
         <declaration symbol="age">
            <attribute id="value()" type="String" class="IntrinsicValueAttribute" />
         </declaration>
      </attributesDecl>
      <semanticProds>
         <semanticProd name="school">
            <lhs nt="school" />
            <rhs>
               <element symbol="students" />
            </rhs>
            <computations>
               <computation name="copy_value">
                  <assignedAttribute att="sum" assocSymbol="school" position="0" />
```

```
        <operation>
            <argument att="sum" assocSymbol="students" position="1" />
            <modus> $1 </modus>
        </operation>
      </computation>
    </computations>
  </semanticProd>
  <semanticProd name="students_1">
    <lhs nt="students" />
    <rhs>
      <element symbol="student" />
      <element symbol="students" />
    </rhs>
    <computations>
      <computation name="getTheSum">
        <assignedAttribute att="sum" assocSymbol="students" position="" />
        <operation>
          <argument att="age" assocSymbol="student" position="1" />
          <argument att="sum" assocSymbol="students" position="2" />
          <modus> $1 + $2 </modus>
        </operation>
      </computation>
    </computations>
  </semanticProd>
  <semanticProd name="students_2">
    <lhs nt="students" />
    <rhs>
      <element symbol="student" />
    </rhs>
    <computations>
      <computation name="copy_value">
        <assignedAttribute att="sum" assocSymbol="students" position="0" />
        <operation>
          <argument att="age" assocSymbol="student" position="1" />
          <modus> $1 </modus>
        </operation>
      </computation>
    </computations>
```

```
    </semanticProd>
    <semanticProd name="student">
      <lhs nt="student" />
      <rhs>
        <element symbol="name" />
        <element symbol="age" />
      </rhs>
      <computations>
        <computation name="toInt_cpy">
          <assignedAttribute att="age" assocSymbol="student" position="" />
          <operation>
            <argument att="value()" assocSymbol="age" position="2" />
            <modus> Integer.parseInt($1) </modus>
          </operation>
        </computation>
      </computations>
    </semanticProd>
  </semanticProds>

  <functions>

  </functions
</attributeGrammar>
```

The XML Dialect proposed is intended to have a universal structure, being possible to keep the specification of any attribute grammar different from LISA.

The End

VisualLISA