CP Visualizer Format

February 15, 2010

Helmut Simonis

email: h.simonis@4c.ucc.ie
homepage: http://4c.ucc.ie/~hsimonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Outline

Contents

Abstract

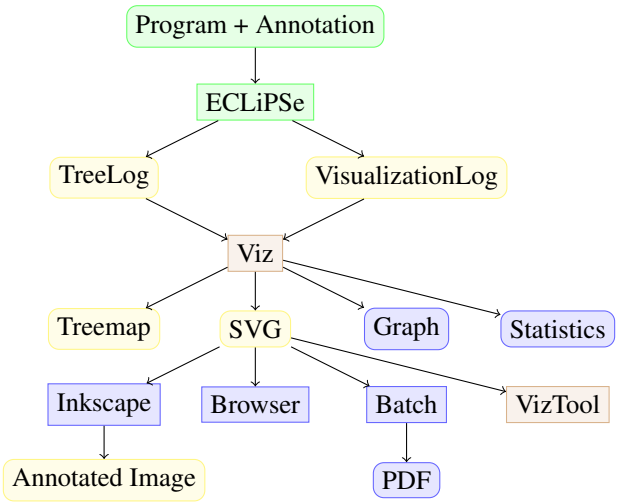
In this paper we describe a generic visualizer tool for finite domain constraint solvers, based on two data formats. The search tree visualizer allows to generate diagrams of the search tree of a finite domain constraint program. The constraint and variable visualizer displays information about the state of variables and constraints at different points of the computation. Both formats are using XML, and are described with their schemas in this document.

1 Introduction

Figure 1 shows the overall system architecture, for the ECLiPSe version of the tool. An annotated constraint program is run through the ECLiPSe system, and produces two log files, one for the search tree, the other for the constraints and variables. These log files are run through the Viz tool to generate visualization files in different formats. Most common are SVG based drawings, but treemaps, graphs and statistics can also be generated. The SVG files can be processed in different ways. They can be displayed in modern Web browsers, or can be edited with the inkscape SVG editor, for example to add annotations. inkscape can also be used in a batch process to convert the SVG drawings into other formats, in particular pdf for inclusion in LaTeX documents. Finally, the SVG files can be fed into the VizTool, an interactive SVG viewer linking multiple search tree and constraint visualizations.

To use the visualizer for another constraint system, the annotations in the source program need to be adjusted, and the log files must be generated during the execution of the tool.

Figure 1: Architecture



This document describes the intermediate log formats for the search tree and constraint visualizer tools. These are XML files, each defined according to a schema detailed in the appendix. The document is structured as follows: In the next section we show an example annotated constraint program (in ECLiPSe), to show how the logging steps interact with the constraint model. We then describe the details of the search tree visualizer in section 3. We give an example of the log format, discuss relevant points of the schema, and show some example output. In section 4, we repeat those steps for the constraint and variable visualizers. In section 5, we detail the interface of the viz program, and in section 6 we describe the interactive viz tool.

2 Example Program

Figure 2 shows the annotated ECLiPSe program to solve the SEND+MORE=MONEY puzzle, which highlights all required steps to interface the log generation with a constraint model. Before any constraint propagation is done, we create the visualization tool calling `create_visualization`, producing an opaque handle *Handle*. This handle gives us access to the visualization tool throughout the program, until we close the visualization

with close_visualization. We add a visualizer to the tool with the add_visualizer call, defining the type of the item to be visualized (here: vector) and the display format we want to use (here: expanded). When setting up constraints, we can explicitly call draw_visualizer to create a new snapshot of the model at this point. In this puzzle problem, we can name the variables with user-friendly labels, this is done with the name_variables call. Before starting the search, we call root to indicate that we are at the root of the search tree. The search routine uses a special version of the indomain assignment routine to automatically handle the updating of the logs at each search step. We describe the predicate tree_indomain below. Once a solution is found, we call the solution predicate to indicate that the search is finished and no further assignments will be done. At the end of the program, we close the visualizer with the close_visualization call, which will close the log files correctly, so that they can be processed with the viz tool.

Figure 2: Annotated SendMoreMoney Program

```
sendmory(L,Output,IgnoreFixed):-
  L=[S,E,N,D,M,O,R,Y],
  L :: 0..9,
  create_visualization([output:Output,
                        ignore_fixed:IgnoreFixed,
                        width:8,
                        height:10],Handle),
  add_visualizer(Handle,
                 vector(L),
                 [display:expanded]),
  alldifferent(L),draw_visualization(Handle),
  S #\= 0,draw_visualization(Handle),
  M #\= 0,draw_visualization(Handle),
  1000*S + 100*E + 10*N + D +
  1000*M + 100*O + 10*R + E #=
  10000*M + 1000*O + 100*N + 10*E + Y,
  name_variables(Handle,L,
                 ['S','E','N','D','M','O','R','Y'],
                 Pairs),
  root(Handle),
  search(Pairs,1,input_order,
         tree_indomain(Handle,_),
         complete,[]),
  solution(Handle),
  close_visualization(Handle).
```

Note that not all the calls are required in every scenario, in particular we can skip the draw_visualization calls and the naming of the variables.

Figure 3 shows the internal operation of a visualizer aware indomain predicate. Depending on the value of the IgnoreFixed option, we may decide not to show any assignment steps which deal with already instantiated variables. The tree_indomain_generic routine gets the values in the domain into a list, possibly reordering the values depending on the assignment type Type and then iterates over the possible values in try_value. For each of the possible values V, it tries to set the variable X to this value. If this works, then it creates a TRY search node, indicating the assignment of the value V as the focus when drawing the visualization. If the assignment fails, it creates a FAIL tree node, and marks the failed assignment in the visualization, before forcing a fail to backtrack over the assignment. In the recursive call, we skip the first value, and test the remaining entries in the list of possible values.

The parameter Term of the indomain routine is not just a domain variable, but a term with multiple arguments. We extract the correct fields with multiple arg calls, using some parameters in the visualization structure. Ordinary ECLiPSe users should not have to understand the details of the tree_indomain_generic implementation, in most cases it is enough just to use a packaged search routine. We presented the details at this point to explain when the different nodes types of the search log are generated, and which state of the execution is captured

Figure 3: Prolog Based Indomain Version

```
tree_indomain_generic(Term,Handle,Handle,Type):-
  Handle = visualization{ignore_fixed:IgnoreFixed,
                         var_arg:VarArg,
                         name_arg:NameArg,
                         focus_arg:FocusArg},
  arg(VarArg,Term,X),
  ((integer(X),IgnoreFixed = yes) ->
      true
  ;
      arg(NameArg,Term,Name),
      arg(FocusArg,Term,Focus),
      get_domain_as_list(X,L),
      get_domain_size(X,Size),
      reorganize_domain(X,L,Type,K),
      try_value(Handle,X,K,Name,Size,Focus)
  ).

try_value(Handle,X,[V|_],Name,Size,Focus):-
  ((X = V, true) ->
      try(Handle,Name,Size,V),
      focus_option(Focus,FocusOption),
      draw_visualization(Handle,FocusOption)
  ;
      failure(Handle,Name,Size,V),
      fail_option(Focus,V,FailOption),
      draw_visualization(Handle,FailOption),
      fail
  ).
try_value(Handle,X,[_|R],Name,Size,Focus):-
  try_value(Handle,X,R,Name,Size,Focus).
```

in the visualization logs.

There are four different search tree log node entries:

root There is a single *root* node with *id* 0, at the very top of the search tree. The corresponding visualizer snapshot is taken before the first assignment is attempted. It typically will show the effect of the constraint setup, e.g. domain reductions which have been performed while stating the constraints. The *root* node is generated by a call to the `root` predicate.

try A *try* node is generated whenever the assignment of a value to a variable succeeded. The visualizer snapshot shows the state after that assignment, and highlights the assigned variable.

fail A *fail* node is generated whenever the assignment of a value to a variable failed. The visualizer snapshot shows the state *before* that assignment, as this is the last fix-point that can be displayed. The visualizer also shows the failed assignment choice itself, but not any further constraint propagation which led to the failure. In particular, it does not show which domain was wiped out, or which constraint detected the failure.

succ Whenever a complete assignment which satisfies all constraints has been found, a *succ* node is generated. It marks the last choice as leading to a success. The visualization snapshot shows the state after the last assignment, i.e. there should be no variables left. The *succ* nodes are generated by a call to the `solution` predicate.

3 Search Tree Visualizer

In this section we describe the log format and the visualizer for the search tree. This can be used as a stand-alone tool to understand the search behavior of the program, without any variable or constraint visualizations.

3.1 Example

Figure 4 shows an example of the log format for the search tree. It is the search tree created when running the SEND+MORE=MONEY program above. The top-level element is *tree*, which contains a sequence of node elements of the four possible types. The first node must be a *root* node, the order of the following entries depends on the search. The log may contain zero, one or more *succ* nodes, the last node often is a *succ* node, but, as shown here, this is not always the case, depending on where the search is stopped. In this example we have explored all possible choice points, unfortunately this is often only possible for quite small problems.

Note how the *tree* element refers to the schema for the search tree log, given in the `tree.xsd` file. Using the schema is not required, but can help XML tools to validate a file based on the given structural information.

3.2 Schema

The `tree.xsd` schema is detailed in the appendix. Figure 5 shows a high-level structural reference. The *tree* element has a single *version* attribute (currently "1.0"), the *succ* and *root* node elements have a single *id* attribute, while *try* and *fail* elements have five attributes. They are

id This is a non-negative integer which serves as a reference to the node. Nodes should be numbered consecutively, starting with the *root* node with *id* 0.

parent This is a reference to the parent of the current node. This is a non-negative integer.

name This string holds the name of the selected variable. Often, this will be just an index in a list or vector rather than a specific name string. In our example file (figure 4) we have named the variables explicitly.

size This non-negative integer is the size of the domain of the selected variable before the assignment. This information can be helpful to understand the size of the unexplored part of the search tree.

value This is the integer value which is assigned to the variable at this node.

All attributes are required. A valid log file must have at least a single *root* node. Note that the log file is a flat structure, all nodes are children of the single *tree* element, i.e. the nodes are not nested according to the tree structure. The structure itself is maintained through the *id* and *parent* attributes.

Figure 4: Example Input Format

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!- Helmut Simonis (University College Cork) ->
<tree version="1.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="tree.xsd">
  <root id="0"/>
  <try id="1" parent="0" name="S" size="1" value="9"/>
  <fail id="2" parent="1" name="E" size="4" value="4"/>
  <try id="3" parent="1" name="E" size="4" value="5"/>
  <try id="4" parent="3" name="N" size="1" value="6"/>
  <try id="5" parent="4" name="D" size="1" value="7"/>
  <try id="6" parent="5" name="M" size="1" value="1"/>
  <try id="7" parent="6" name="O" size="1" value="0"/>
  <try id="8" parent="7" name="R" size="1" value="8"/>
  <try id="9" parent="8" name="Y" size="1" value="2"/>
  <succ id="9"/>
  <fail id="10" parent="1" name="E" size="4" value="6"/>
  <fail id="11" parent="1" name="E" size="4" value="7"/>
</tree>
```

3.3 Output

Figure 6 shows the resulting tree picture generated from the log file in figure 4. *Try* nodes are visualized as a node labelled with the variable name and a downward link labelled with the selected value. If multiple values for a variable are explored in a choice point, then the variable node is shared. *Fail* nodes are displayed as small red circles, while *succ* nodes are displayed as slightly larger, green circles. The *root* node is usually not displayed. Considering the links in upward direction shows the parent relation.

As the search tree grows in size, displaying the complete, expanded search tree becomes unwieldy. There are different options to compress failed sub-trees, or to only show some of the nodes in the tree. The latter option is also used to generate animations of the search progress, where a separate picture is drawn for each search node. Shown in sequence, they give an impression of the search progress.

4 Constraint and Variable Visualizers

In this section we describe the log format and visualizers for domain variables and constraints. While they can be used stand-alone, they normally refer to search tree nodes, so that a proper understanding of the behavior must integrate both logs.

4.1 Example

Figure 7 shows an excerpt of the log file generated from the example program in figure 2. An ellipsis (...) marks parts which have been removed. The top level element is *visualization*, which contains a non-empty set of *visualizer* elements followed by a set of *state* elements. The *visualizer* elements describe the different visualizers which should be shown in the visualization, the *state* elements describe execution states where snapshots for the variables and constraints are taken. Inside each *state* element, there are *visualizer_state* elements, one for each visualizer. The states refer to the current search tree node with an attribute *tree_node*. This attribute has value -1 if the snapshot is taken outside the search, otherwise it is the *id* of the current node. The *visualizer_state* element refers to its *visualizer* through its *id* attribute.

4.1.1 Basic Types

Inside each *visualizer_state* element we describe the state of its variables. This can be one of four types:

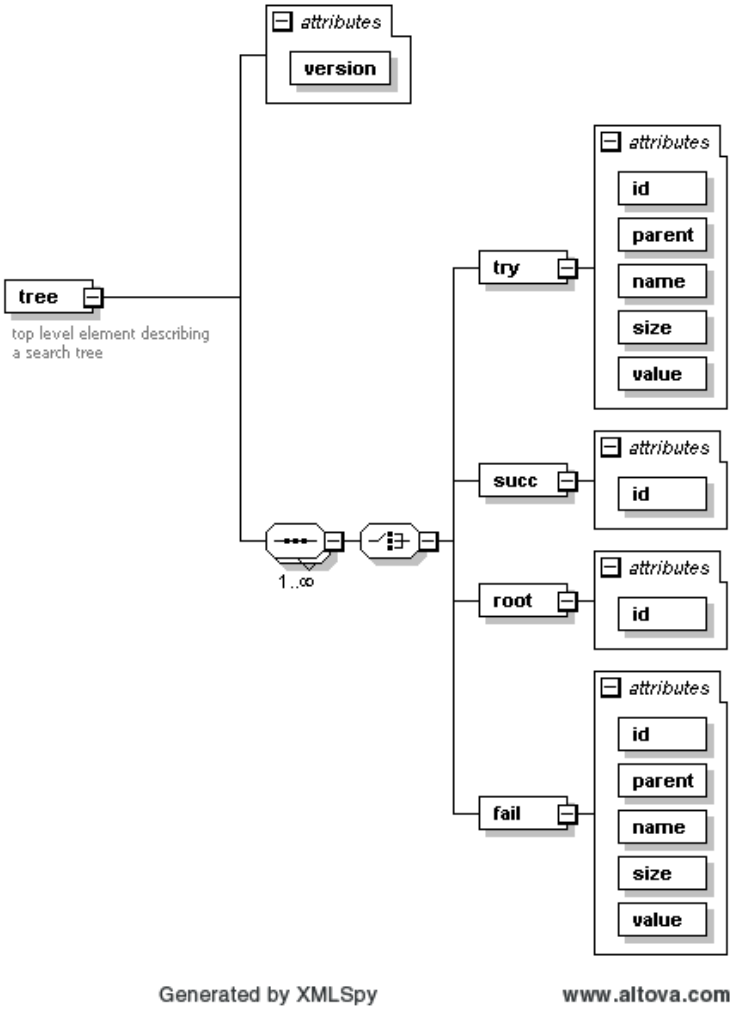


Figure 5: Search Tree Schema

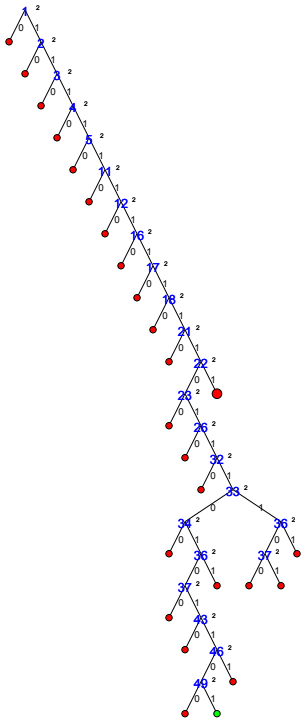


Figure 6: Example Search Tree Output

Figure 7: Visualization Log Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<visualization version="1.0"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation="visualization.xsd">
<visualizer id="1" type="vector" display="expanded"
           x="0" y="0" width="8" height="10" min="0"  max="9" />
  <state id="1" tree_node="-1">
    <visualizer_state id="1">
      <dvar index="1" domain="0 .. 9"/>
      <dvar index="2" domain="0 .. 9"/>
      <dvar index="3" domain="0 .. 9"/>
      <dvar index="4" domain="0 .. 9"/>
      <dvar index="5" domain="0 .. 9"/>
      <dvar index="6" domain="0 .. 9"/>
      <dvar index="7" domain="0 .. 9"/>
      <dvar index="8" domain="0 .. 9"/>
    </visualizer_state>
  </state>
...
  <state id="5" tree_node="1">
    <visualizer_state id="1">
      <integer index="1" value="9"/>
      <dvar index="2" domain="4 .. 7"/>
      <dvar index="3" domain="5 .. 8"/>
      <dvar index="4" domain="2 .. 8"/>
      <integer index="5" value="1"/>
      <integer index="6" value="0"/>
      <dvar index="7" domain="2 .. 8"/>
      <dvar index="8" domain="2 .. 8"/>
      <focus index="1"/>
    </visualizer_state>
  </state>
  <state id="6" tree_node="2">
    <visualizer_state id="1">
      <integer index="1" value="9"/>
      <dvar index="2" domain="4 .. 7"/>
      <dvar index="3" domain="5 .. 8"/>
      <dvar index="4" domain="2 .. 8"/>
      <integer index="5" value="1"/>
      <integer index="6" value="0"/>
      <dvar index="7" domain="2 .. 8"/>
      <dvar index="8" domain="2 .. 8"/>
      <failed index="2" value="4"/>
    </visualizer_state>
  </state>
...
  <state id="14" tree_node="9">
    <visualizer_state id="1">
      <integer index="1" value="9"/>
      <integer index="2" value="5"/>
      <integer index="3" value="6"/>
      <integer index="4" value="7"/>
      <integer index="5" value="1"/>
      <integer index="6" value="0"/>
      <integer index="7" value="8"/>
      <integer index="8" value="2"/>
    </visualizer_state>
  </state>
...
</visualization>
```

dvar This element describes an unassigned variable, its position in the visualizer (attribute *index*), and its domain (attribute *domain*). If the visualizer is a one dimensional collection, the *index* is an integer, for two dimensional collections, it is a sequence of two integers, etc. The domain can be given as an interval (`1 .. 9`) or as a white space separated list of integer values (`1 2 3 5 7`).

integer This element describes an assigned variable. The *index* attribute plays the same role as for *dvar* elements.

svar This element describes a set variable, given by lower and upper bounds on the set elements.

sinteger This is the basic type for a finite integer set.

other The element can be used by more complex global constraints to contain information which is not of one of the other basic types. An example would be atomic constraint types ($<$, $\leq$, $\neq$) used by some constraints.

focus This element describes a value assignment which should be highlighted in the visualizer. This is used in states which correspond to *try* search nodes. Only a single entry may be given.

failed This element describes a failed assignment. It marks the element which should be highlighted in the visualizer. This is used in states which correspond to *fail* search nodes. Only a single entry may be given, it can not be used in the same `visualizer_state` with a *focus* element.

4.1.2 Structured Types

If we want to visualize more complex global constraints, we can use additional structured types.

argument A number of (named or numbered) arguments can be given for a visualizer. Each argument may contain a single basic type, or a sequence of basic types, or more structured collection and tuple types. Arguments can only appear as direct children of `visualizer_state` elements, they can not be nested.

collection A collection is used to bind a sequence of other elements together. All elements inside a collection must have the same type, and can be other collections, tuples or basic types.

tuple A tuple provides fields for multiple elements which are accessed by name. They do not need to have the same type, and can be collections, other tuples or basic types.

Figure 8 shows an excerpt of a log file for a cumulative constraint. The constraint has two arguments, the first is a sequence of tuples, the second a single limit value. Each tuple corresponds to a task, the fields can be accessed with named indices *start*, *dur* and *res*.

4.2 Schema

Figure 9 shows the high-level structure of the `visualization.xsd` schema, which is described in more detail in the appendix. The top level *visualization* element contains *visualizer* and *state* elements. Insides each *state* element we have a sequence of *visualizer_state* elements, which in turn contain elements for variables and constants as well as focus and failure information. The schema contains the structured elements for arguments, collections and tuples, but does not enforce all constraints on them, e.g. it does not know that all members of a collection should have them same type.

4.3 Output

Figure 10 shows the output of the vector visualization in expanded form for a single state of execution. A vector is a one-dimensional sequence of variables, the expanded display shows each possible value in the domains as a separate field. Color coding is used to show which values have been assigned at this step (red), which values have been removed from the domain (blue) and which values remain in the domains (green). The display shown does not use the optional labels to save space in the display.

Figure 8: Example Log for Cumulative Constraint

```
</visualizer_state>
<visualizer_state id="6" >
<argument index="tasks" >
<tuple index="1" >
<dvar index="start" domain="1 .. 8" />
<integer index="dur" value="1" />
<integer index="res" value="1" />
</tuple>
<tuple index="2" >
<dvar index="start" domain="1 .. 8" />
<integer index="dur" value="1" />
<integer index="res" value="1" />
</tuple>
<tuple index="3" >
<dvar index="start" domain="1 .. 8" />
<integer index="dur" value="1" />
<integer index="res" value="1" />
</tuple>
<tuple index="4" >
<dvar index="start" domain="1 .. 8" />
<integer index="dur" value="1" />
<integer index="res" value="1" />
</tuple>
<tuple index="5" >
<dvar index="start" domain="1 .. 8" />
<integer index="dur" value="1" />
<integer index="res" value="1" />
</tuple>
<tuple index="6" >
<dvar index="start" domain="1 .. 8" />
<integer index="dur" value="1" />
<integer index="res" value="1" />
</tuple>
<tuple index="7" >
<dvar index="start" domain="1 .. 8" />
<integer index="dur" value="1" />
<integer index="res" value="1" />
</tuple>
<tuple index="8" >
<dvar index="start" domain="1 .. 8" />
<integer index="dur" value="1" />
<integer index="res" value="1" />
</tuple>
</argument>
<argument index="limit" >
<integer index="1" value="1" />
</argument>
</visualizer_state>
```

5 Viz Program

The viz program is a command line tool which converts the log files into one or multiple drawings of some format. It is called with three arguments, all XML files. The first is a configuration file, which defines the tools that should be used to visualize the input data. The format of the configuration is detailed in the appendix, an example file is shown in figure 11.

More detailed documentation of the Viz program is given in its Javadoc description.

6 VizTool Program

The VizTool program is an interactive tool to display the visualizations along a time-line and to animate movement through the search. Figure 13 shows a snapshot. There is a time-line at the top, and two main panes, on the left for the search tree, on the right for the constraint visualization.

7 Conclusion

In this paper we have described the format for a generic CP visualizer, developed initially for an on-line constraint programming course using ECLiPSe. It uses XML based log files to collect information about the search, the

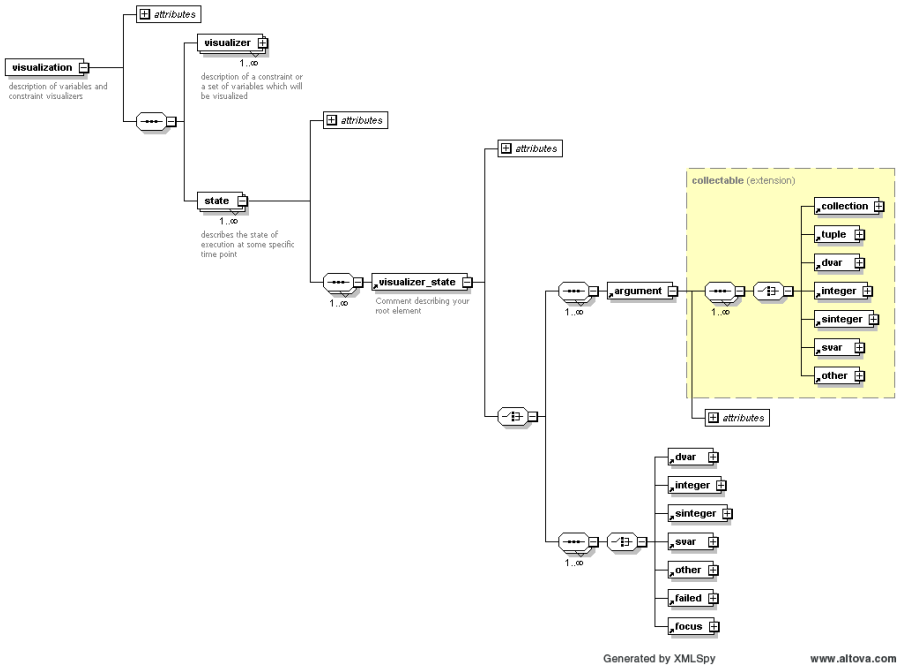


Figure 9: Visualization Log Schema

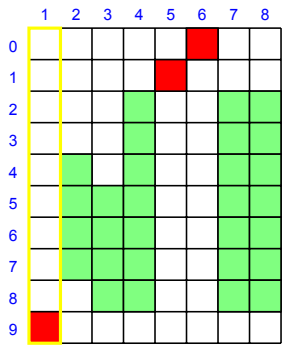


Figure 10: Example Visualization Output

Figure 11: Configuration Sample File

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2010 (http://www.altova.com)-->
<configuration version="1.0" directory="examples/mix/RESULT"
               xsi:noNamespaceSchemaLocation="configuration.xsd"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<tool show="tree" type="layout" display="expanded" repeat="all"
             width="700" height="700" fileroot="tree"  />
<tool show="viz" type="layout" display="compact" repeat="final"
             width="900" height="900" fileroot="viz"  />
</configuration>
```

variables and the constraints of the problem, and produces diagrams that can be used interactively, or processed for inclusion in documents and web sites.

The log formats are described using XML schemas, and Java tools to process and display the information are available under a Modzilla-type license for use with any constraint programming system.

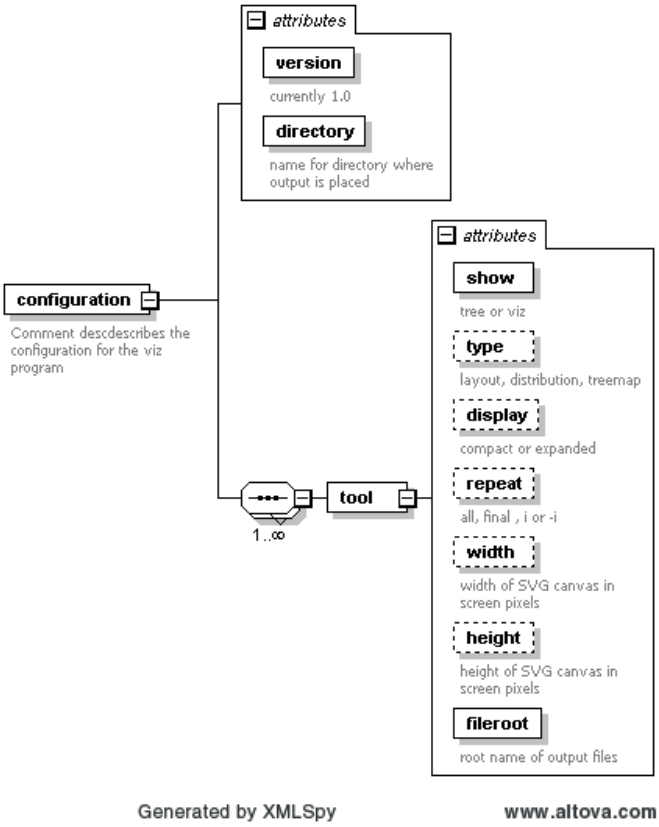
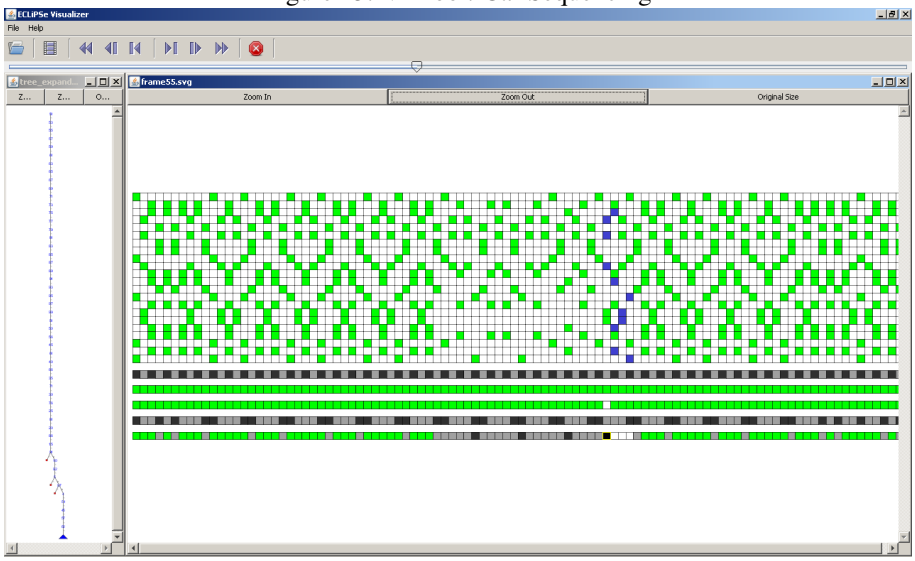


Figure 12: Configuration Schema

Figure 13: VizTool: Car Sequencing

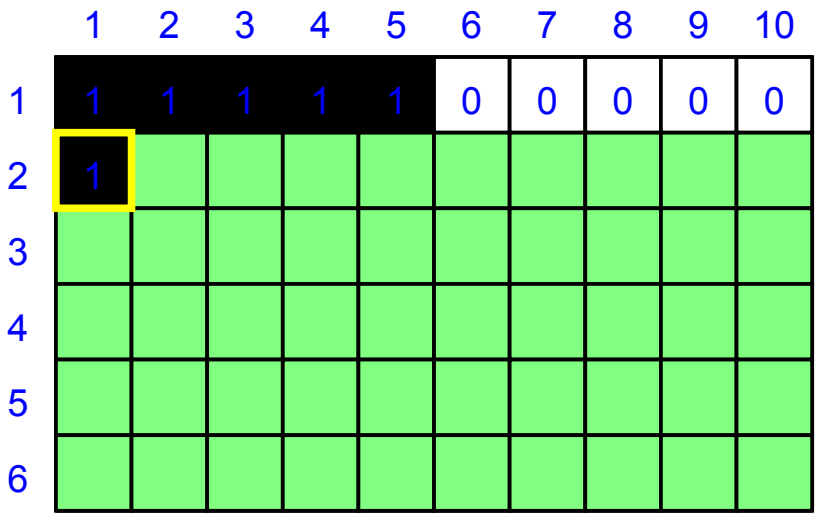


A Example Programs

The following pages show some snapshots from the visualization of different example programs written in ECLiPSe, generated by the Viz program as SVG output and converted into PDF documents through inkscape. We used the inkscapec extension to run inkscape through a Makefile.

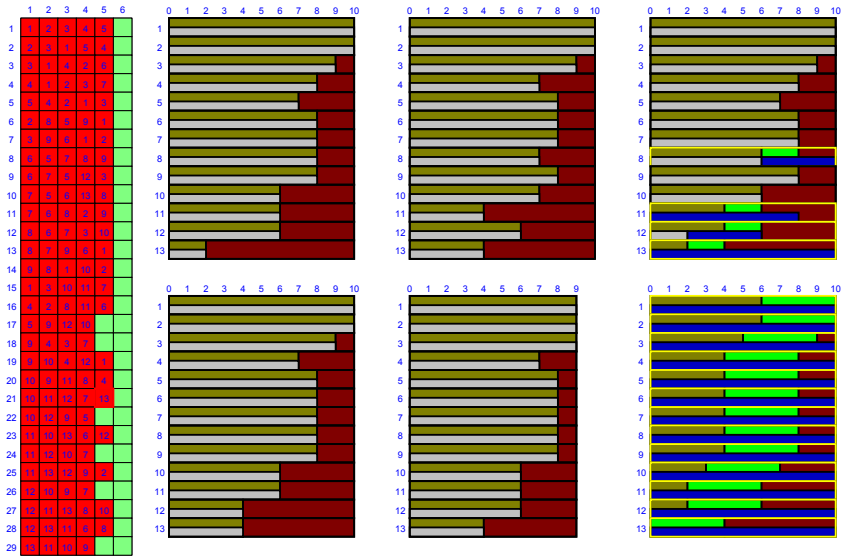
A.1 Bibd

Figure 14: Balanced Incomplete Block Design



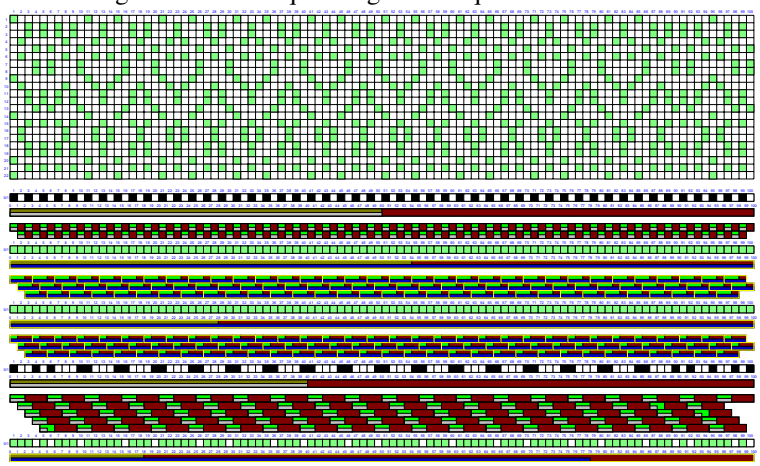
A.2 Bin

Figure 15: Bin Packing Version of Party



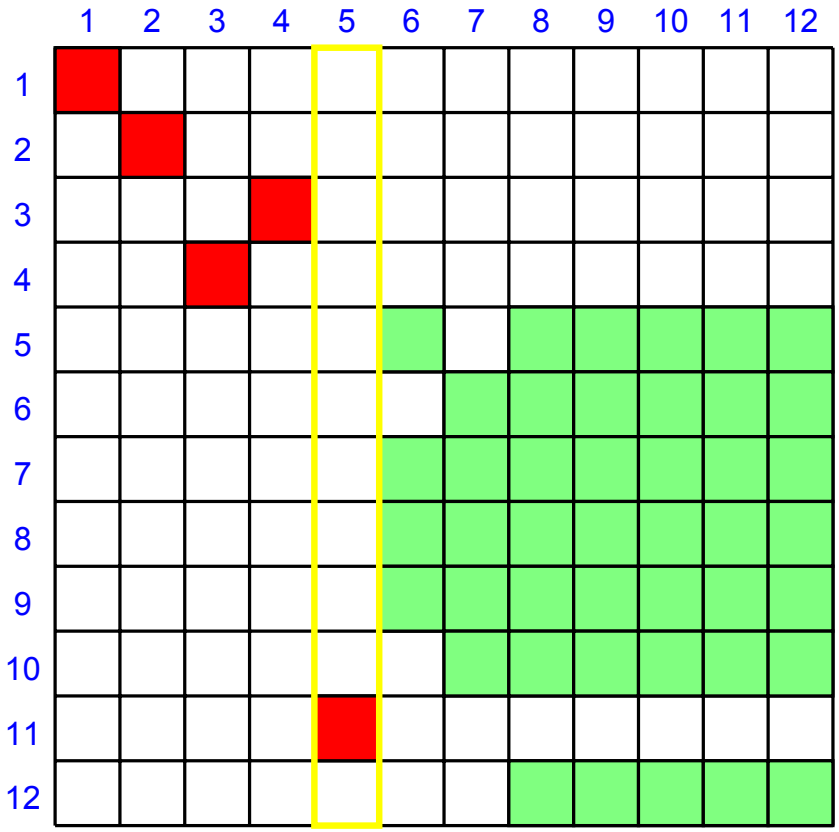
A.3 Car

Figure 16: Car Sequencing with Sequence Constraints



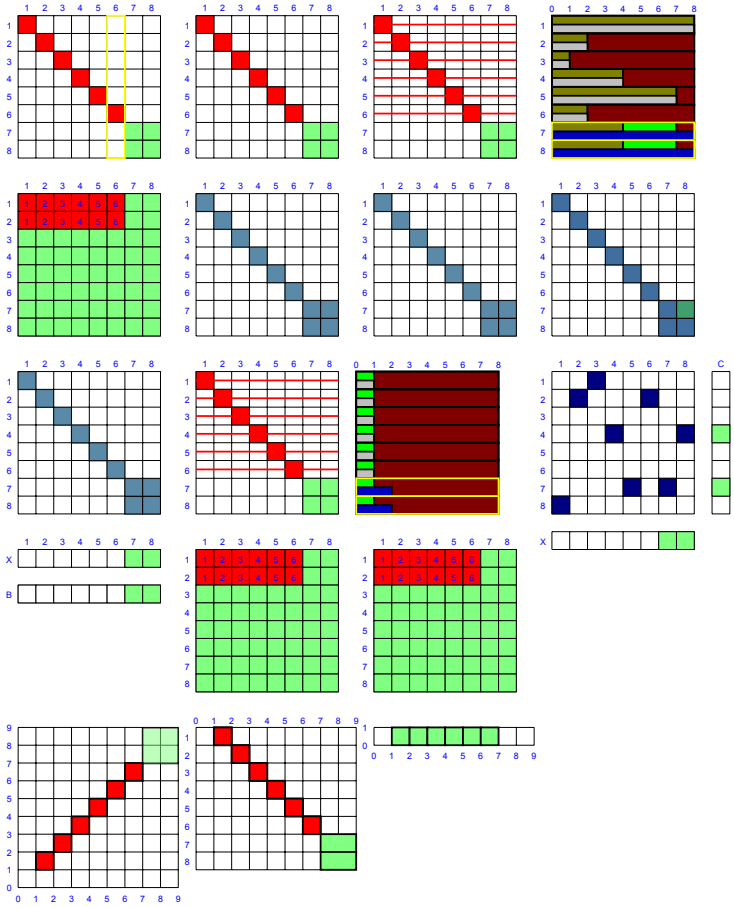
A.4 Costas

Figure 17: Costas Array



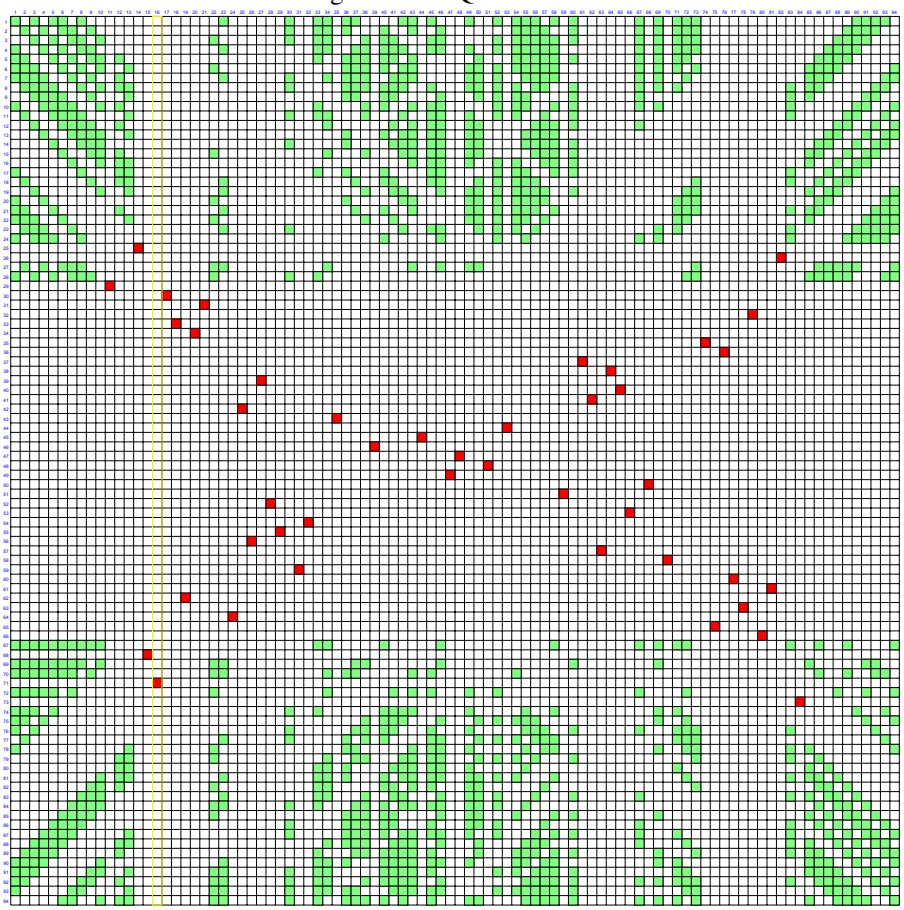
A.5 Mix

Figure 18: Visualization Demo



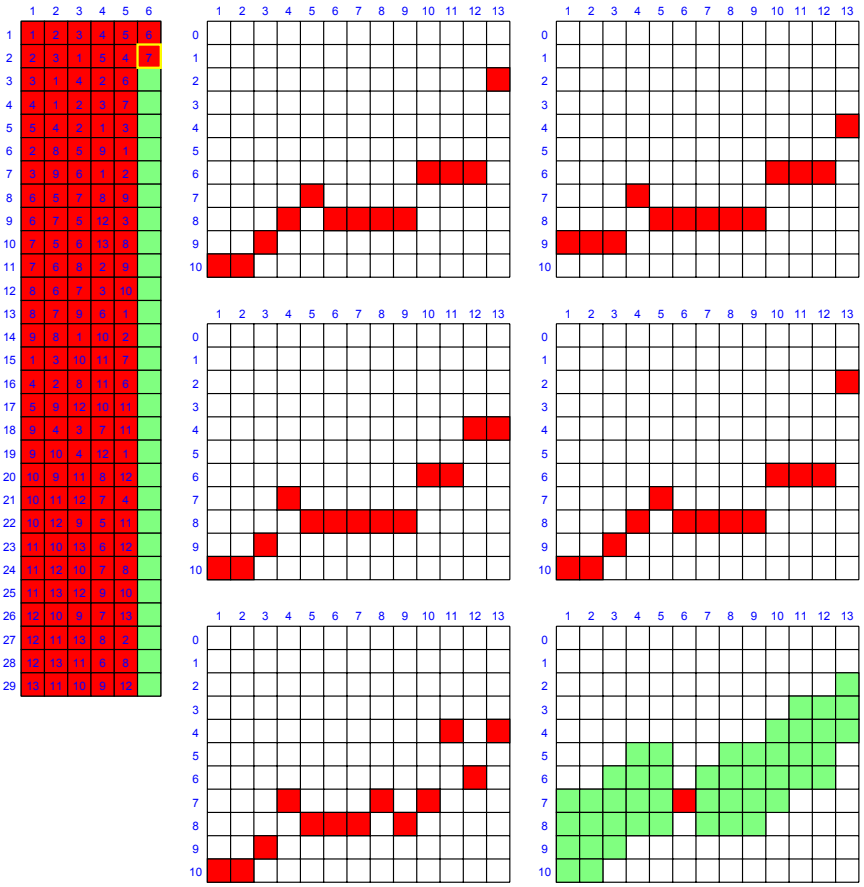
A.6 Nqueen

Figure 19: N-Queen Puzzle



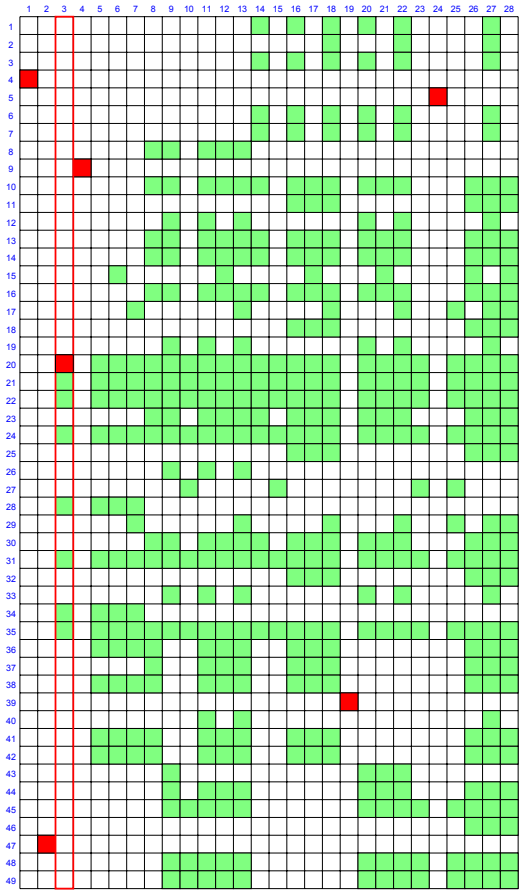
A.7 Party

Figure 20: Progressive Party Problem



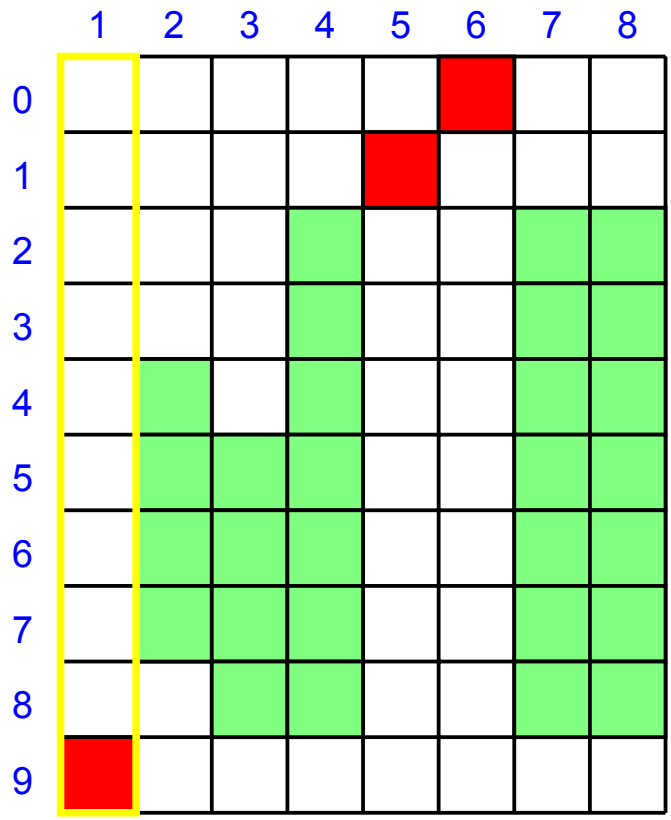
A.8 Rooms

Figure 21: Rooms Puzzle



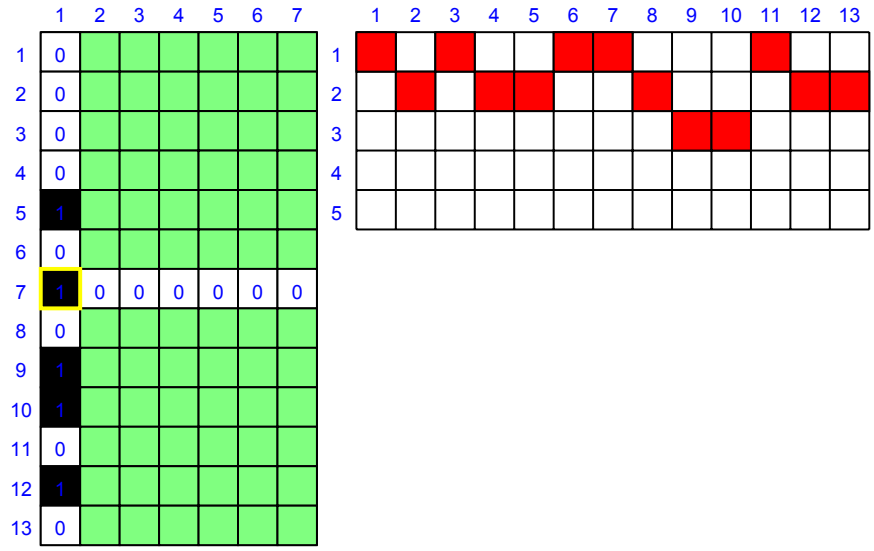
A.9 Sendmore

Figure 22: SEND+MORE=MONEY Puzzle



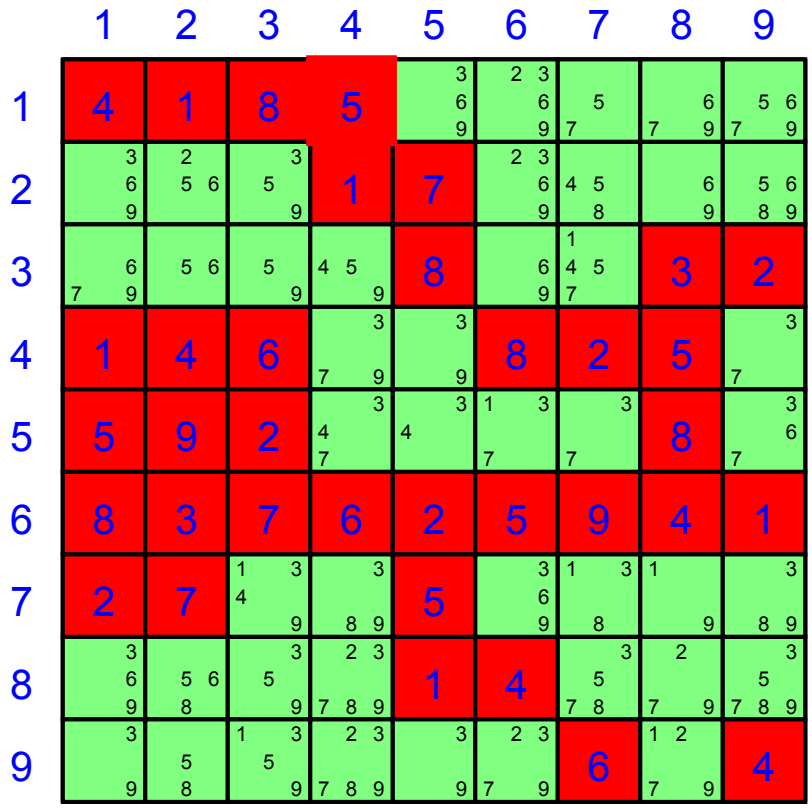
A.10 Sonet

Figure 23: Sonet Network Design



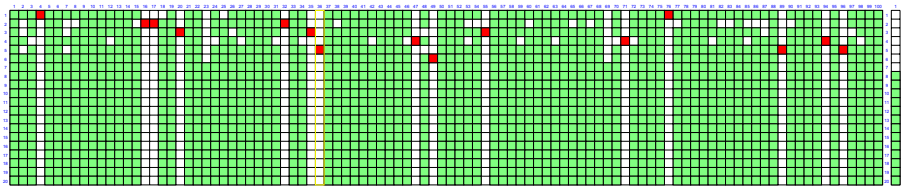
A.11 Sudoku

Figure 24: Sudoku



A.12 Wave

Figure 25: Routing and Wavelength Assignment

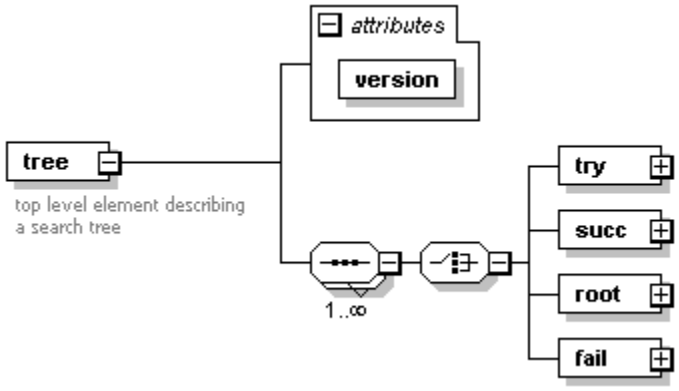


Schema **tree.xsd**

schema location: **..\..\sendmore\FULL\tree.xsd**
attribute form default: **unqualified**
element form default: **qualified**

Elements
tree

element **tree**

diagram	
properties	content complex
children	**try succ root fail**
attributes	Name: version; Type: **xs:string**; Use: required; Default: ; Fixed: ; annotation:
annotation	documentation top level element describing a search tree
source	

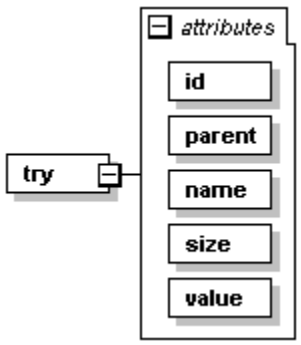
```
<xs:element name="tree">
 <xs:annotation>
  <xs:documentation>top level element describing a search tree</xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence maxOccurs="unbounded">
   <xs:choice>
    <xs:element name="try">
     <xs:complexType>
      <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
      <xs:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="size" type="xs:nonNegativeInteger" use="required"/>
      <xs:attribute name="value" type="xs:integer" use="required"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="succ">
     <xs:complexType>
      <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
     </xs:complexType>
    </xs:element>
```

```
      <xs:element name="root">
        <xs:complexType>
          <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="fail">
        <xs:complexType>
          <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
          <xs:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>
          <xs:attribute name="name" type="xs:string" use="required"/>
          <xs:attribute name="size" type="xs:nonNegativeInteger" use="required"/>
          <xs:attribute name="value" type="xs:integer" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="version" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>
```

attribute **tree/@version**

type	**xs:string**
properties	isRef 0 use required
source	`<xs:attribute name="version" type="xs:string" use="required"/>`

element **tree/try**

diagram	
properties	isRef 0 content complex

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xs:nonNegativeInteger**	required			
parent	**xs:nonNegativeInteger**	required			
name	**xs:string**	required			
size	**xs:nonNegativeInteger**	required			
value	**xs:integer**	required			

source

```
<xs:element name="try">
 <xs:complexType>
  <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>
```

	<xs:attribute name="name" type="xs:string" use="required"/> <xs:attribute name="size" type="xs:nonNegativeInteger" use="required"/> <xs:attribute name="value" type="xs:integer" use="required"/> </xs:complexType> </xs:element>

attribute **tree/try/@id**

type	**xs:nonNegativeInteger**
properties	isRef 0 use required
source	<xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>

attribute **tree/try/@parent**

type	**xs:nonNegativeInteger**
properties	isRef 0 use required
source	<xs:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>

attribute **tree/try/@name**

type	**xs:string**
properties	isRef 0 use required
source	<xs:attribute name="name" type="xs:string" use="required"/>

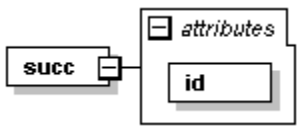
attribute **tree/try/@size**

type	**xs:nonNegativeInteger**
properties	isRef 0 use required
source	<xs:attribute name="size" type="xs:nonNegativeInteger" use="required"/>

attribute **tree/try/@value**

type	**xs:integer**
properties	isRef 0 use required
source	<xs:attribute name="value" type="xs:integer" use="required"/>

element **tree/succ**

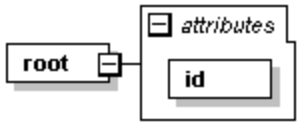
diagram	

properties	isRef 0 content complex
attributes	Name: id; Type: **xs:nonNegativeInteger**; Use: required; Default; Fixed; annotation
source	<xs:element name="succ"> <xs:complexType> <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/> </xs:complexType> </xs:element>

attribute **tree/succ/@id**

type	**xs:nonNegativeInteger**
properties	isRef 0 use required
source	<xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>

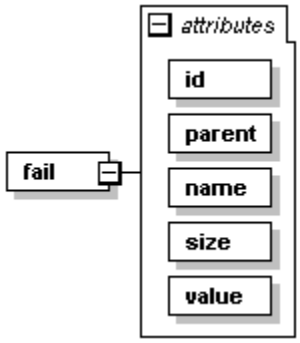
element **tree/root**

diagram	
properties	isRef 0 content complex
attributes	Name: id; Type: **xs:nonNegativeInteger**; Use: required; Default; Fixed; annotation
source	<xs:element name="root"> <xs:complexType> <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/> </xs:complexType> </xs:element>

attribute **tree/root/@id**

type	**xs:nonNegativeInteger**
properties	isRef 0 use required
source	<xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>

element **tree/fail**

diagram	
properties	isRef 0 content complex

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xs:nonNegativeInteger**	required			
parent	**xs:nonNegativeInteger**	required			
name	**xs:string**	required			
size	**xs:nonNegativeInteger**	required			
value	**xs:integer**	required			

source

```
<xs:element name="fail">
  <xs:complexType>
    <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
    <xs:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="size" type="xs:nonNegativeInteger" use="required"/>
    <xs:attribute name="value" type="xs:integer" use="required"/>
  </xs:complexType>
</xs:element>
```

attribute **tree/fail/@id**

type	**xs:nonNegativeInteger**
properties	isRef 0 use required
source	`<xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>`

attribute **tree/fail/@parent**

type	**xs:nonNegativeInteger**
properties	isRef 0 use required
source	`<xs:attribute name="parent" type="xs:nonNegativeInteger" use="required"/>`

attribute **tree/fail/@name**

type	**xs:string**
properties	isRef 0 use required
source	`<xs:attribute name="name" type="xs:string" use="required"/>`

attribute **tree/fail/@size**

type	**xs:nonNegativeInteger**
properties	isRef 0 use required
source	<xs:attribute name="size" type="xs:nonNegativeInteger" use="required"/>

attribute **tree/fail/@value**

type	**xs:integer**
properties	isRef 0 use required
source	<xs:attribute name="value" type="xs:integer" use="required"/>

Schema **visualization.xsd**

schema location: **..\..\..\courseware\documentation\visualization.xsd**
attribute form default: **unqualified**
element form default: **qualified**

Attributes	Elements	Complex types
group	**argument**	**collectable**
index	**collection**	**items**
	dvar	
	failed	
	focus	
	integer	
	other	
	sinteger	
	state	
	svar	
	tuple	
	visualization	
	visualizer	
	visualizer_state	

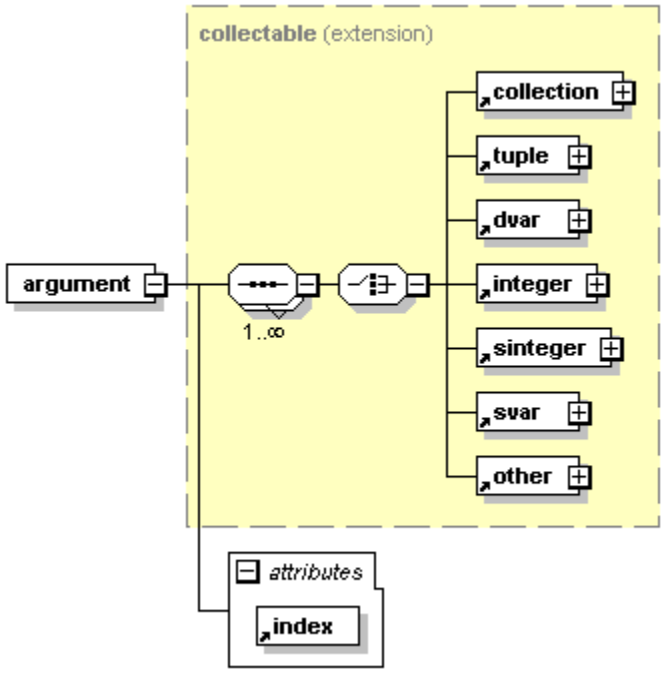
attribute **group**

type	**xs:string**
used by	elements **failed** **focus**
source	<xs:attribute name="group" type="xs:string"/>

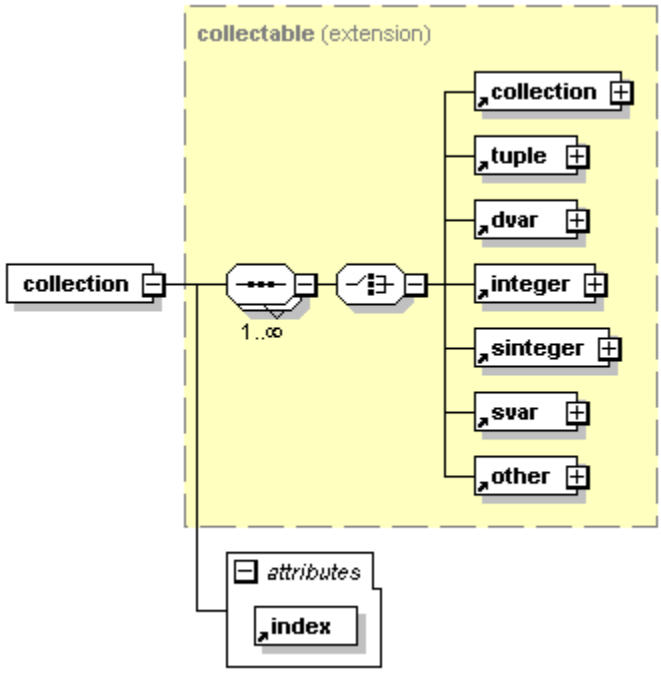
attribute **index**

type	**xs:string**
used by	elements **argument** **collection** **dvar** **failed** **focus** **integer** **other** **sinteger** **svar** **tuple**
source	<xs:attribute name="index" type="xs:string"/>

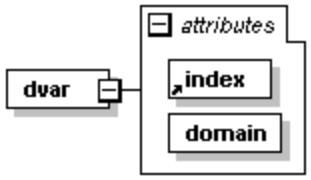
element **argument**

diagram	
type	extension of **collectable**
properties	content complex
children	**collection** **tuple** **dvar** **integer** **sinteger** **svar** **other**
used by	element **visualizer_state**
attributes	Name: index; Type: ; Use: required; Default: ; Fixed: ; annotation:
source	<xs:element name="argument"> <xs:complexType> <xs:complexContent> <xs:extension base="collectable"> <xs:attribute ref="index" use="required"/> </xs:extension> </xs:complexContent> </xs:complexType> </xs:element>

element **collection**

diagram	
type	extension of **collectable**
properties	content complex
children	**collection** **tuple** **dvar** **integer** **sinteger** **svar** **other**
used by	complexType **collectable**
attributes	Name Type Use Default Fixed annotation index required
source	<xs:element name="collection"> <xs:complexType> <xs:complexContent> <xs:extension base="collectable"> <xs:attribute ref="index" use="required"/> </xs:extension> </xs:complexContent> </xs:complexType> </xs:element>

element **dvar**

diagram	
properties	content complex
used by	element **visualizer_state** complexTypes **collectable** **items**

attributes	Name	Type	Use	Default	Fixed	annotation
	index		required			
	domain	**xs:string**	required			

source

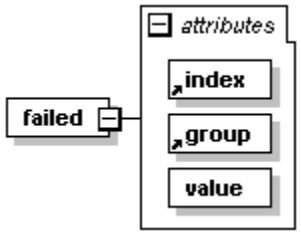
```
<xs:element name="dvar">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute name="domain" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

attribute **dvar/@domain**

type	**xs:string**
properties	isRef 0 use required
source	`<xs:attribute name="domain" type="xs:string" use="required"/>`

element **failed**

diagram



properties	content complex
used by	element **visualizer_state**

attributes	Name	Type	Use	Default	Fixed	annotation
	index		required			
	group		required			
	value	**xs:integer**	required			

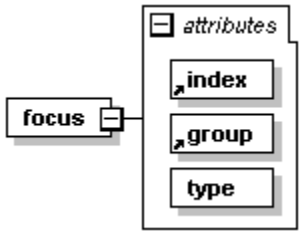
source

```
<xs:element name="failed">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute ref="group" use="required"/>
    <xs:attribute name="value" type="xs:integer" use="required"/>
  </xs:complexType>
</xs:element>
```

attribute **failed/@value**

type	**xs:integer**
properties	isRef 0 use required
source	`<xs:attribute name="value" type="xs:integer" use="required"/>`

element **focus**

diagram	
properties	content complex
used by	element **visualizer_state**

attributes

Name	Type	Use	Default	Fixed	annotation
index		required			
group		required			
type	**xs:string**	required			

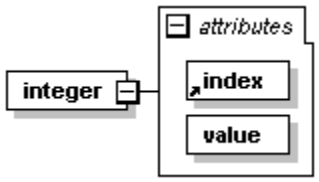
source

```
<xs:element name="focus">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute ref="group" use="required"/>
    <xs:attribute name="type" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

attribute **focus/@type**

type	**xs:string**
properties	isRef 0 use required
source	`<xs:attribute name="type" type="xs:string" use="required"/>`

element **integer**

diagram	
properties	content complex
used by	element **visualizer_state** complexTypes **collectable items**

attributes

Name	Type	Use	Default	Fixed	annotation
index		required			
value	**xs:integer**	required			

source

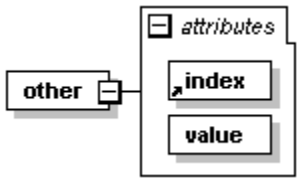
```
<xs:element name="integer">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute name="value" type="xs:integer" use="required"/>
  </xs:complexType>
```

	</xs:element>

attribute **integer/@value**

type	**xs:integer**
properties	isRef 0 use required
source	<xs:attribute name="value" type="xs:integer" use="required"/>

element **other**

diagram	
properties	content complex
used by	element **visualizer_state** complexTypes **collectable items**

attributes

Name	Type	Use	Default	Fixed	annotation
index		required			
value	**xs:string**	required			

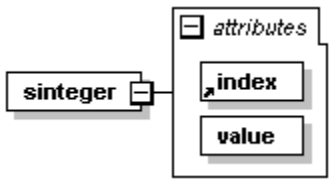
source

```
<xs:element name="other">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute name="value" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

attribute **other/@value**

type	**xs:string**
properties	isRef 0 use required
source	<xs:attribute name="value" type="xs:string" use="required"/>

element **sinteger**

diagram	
properties	content complex
used by	element **visualizer_state** complexTypes **collectable items**

attributes	Name	Type	Use	Default	Fixed	annotation
	index		required			
	value	**xs:string**	required			

source

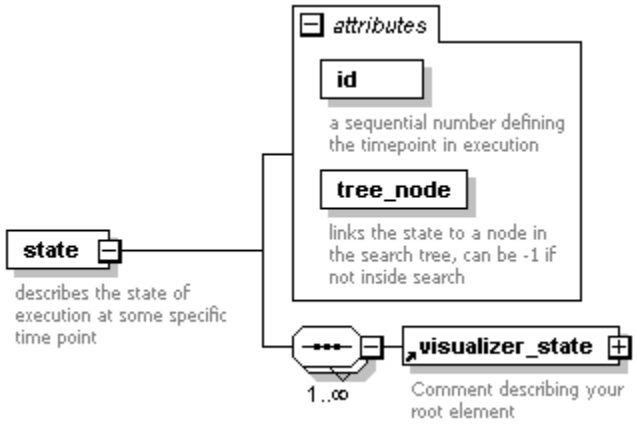
```
<xs:element name="sinteger">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute name="value" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

attribute **sinteger/@value**

type	**xs:string**
properties	isRef 0 use required
source	`<xs:attribute name="value" type="xs:string" use="required"/>`

element **state**

diagram



properties	content complex
children	**visualizer_state**

attributes	Name	Type	Use	Default	Fixed	annotation
	id	**xs:nonNegativeInteger**	required			documentation a sequential number defining the timepoint in execution
	tree_node	**xs:integer**	required			documentation links the state to a node in the search tree, can be -1 if not inside search

annotation	documentation describes the state of execution at some specific time point

source

```
<xs:element name="state">
  <xs:annotation>
    <xs:documentation>describes the state of execution at some specific time
```

```
point</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
        <xs:element ref="visualizer_state"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:nonNegativeInteger" use="required">
        <xs:annotation>
          <xs:documentation>a sequential number defining the timepoint in
execution</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="tree_node" type="xs:integer" use="required">
        <xs:annotation>
          <xs:documentation>links the state to a node in the search tree, can be -1 if not inside
search</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
</xs:element>
```

attribute **state/@id**

type	**xs:nonNegativeInteger**
properties	isRef 0 use required
annotation	documentation a sequential number defining the timepoint in execution

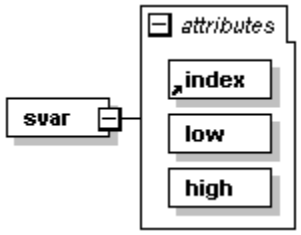
source	`<xs:attribute name="id" type="xs:nonNegativeInteger" use="required">`
	`  <xs:annotation>`
	`  <xs:documentation>a sequential number defining the timepoint in execution</xs:documentation>`
	`  </xs:annotation>`
	`</xs:attribute>`

attribute **state/@tree_node**

type	**xs:integer**
properties	isRef 0 use required
annotation	documentation links the state to a node in the search tree, can be -1 if not inside search

source	`<xs:attribute name="tree_node" type="xs:integer" use="required">`
	`  <xs:annotation>`
	`  <xs:documentation>links the state to a node in the search tree, can be -1 if not inside search</xs:documentation>`
	`  </xs:annotation>`
	`</xs:attribute>`

element **svar**

diagram	
properties	content complex
used by	element **visualizer_state** complexTypes **collectable items**

attributes

Name	Type	Use	Default	Fixed	annotation
index		required			
low	**xs:string**	required			
high	**xs:string**	required			

source

```
<xs:element name="svar">
  <xs:complexType>
    <xs:attribute ref="index" use="required"/>
    <xs:attribute name="low" type="xs:string" use="required"/>
    <xs:attribute name="high" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

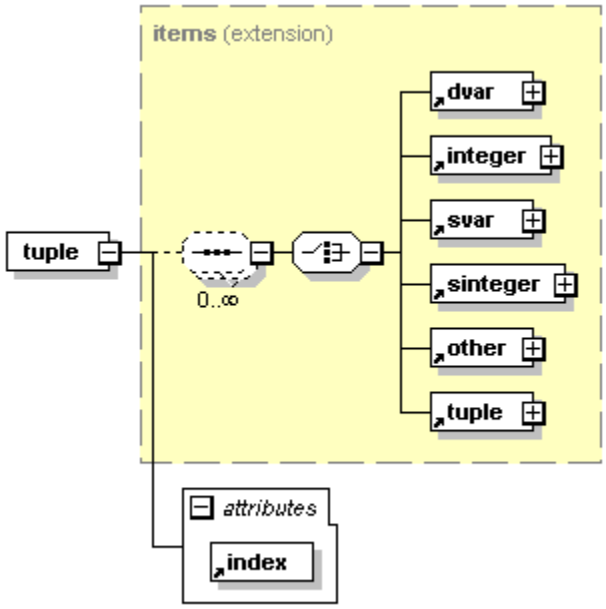
attribute **svar/@low**

type	**xs:string**
properties	isRef 0 use required
source	`<xs:attribute name="low" type="xs:string" use="required"/>`

attribute **svar/@high**

type	**xs:string**
properties	isRef 0 use required
source	`<xs:attribute name="high" type="xs:string" use="required"/>`

element **tuple**

diagram	
type	extension of **items**
properties	content complex
children	**dvar integer svar sinteger other tuple**
used by	complexTypes **collectable items**

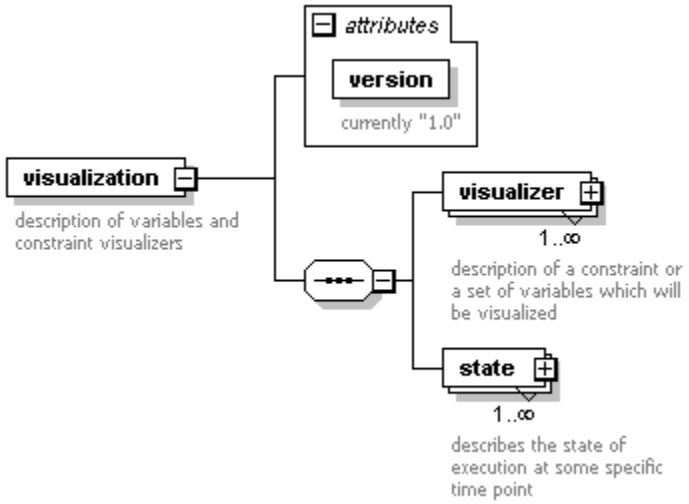
attributes	Name	Type	Use	Default	Fixed	annotation
	index		required			

source

```
<xs:element name="tuple">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="items">
        <xs:attribute ref="index" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

element **visualization**

diagram



properties
content complex

children
visualizer **state**

attributes

Name	Type	Use	Default	Fixed	annotation
version	**xs:string**	required			documentation currently "1.0"

annotation
documentation
description of variables and constraint visualizers

source

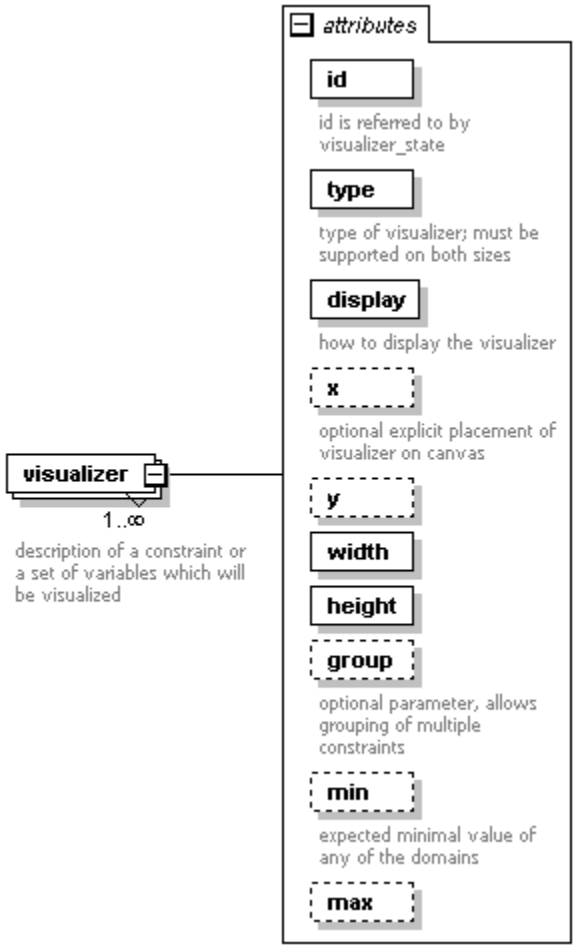
```
<xs:element name="visualization">
  <xs:annotation>
    <xs:documentation>description of variables and constraint visualizers</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="visualizer" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>description of a constraint or a set of variables which will be
visualized</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:attribute name="id" type="xs:string" use="required">
            <xs:annotation>
              <xs:documentation>id is referred to by visualizer_state</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="type" type="xs:string" use="required">
            <xs:annotation>
              <xs:documentation>type of visualizer; must be supported on both
sizes</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="display" type="xs:string" use="required">
            <xs:annotation>
              <xs:documentation>how to display the visualizer</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="x" type="xs:integer" default="0">
            <xs:annotation>
```

```
                <xs:documentation>optional explicit placement of visualizer on
canvas</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="y" type="xs:integer" default="0"/>
            <xs:attribute name="width" type="xs:integer" use="required"/>
            <xs:attribute name="height" type="xs:integer" use="required"/>
            <xs:attribute name="group" type="xs:string">
              <xs:annotation>
                <xs:documentation>optional parameter, allows grouping of multiple
constraints</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="min" type="xs:integer" default="0">
              <xs:annotation>
                <xs:documentation>expected minimal value of any of the
domains</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="max" type="xs:integer" default="0"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="state" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>describes the state of execution at some specific time
point</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
              <xs:element ref="visualizer_state"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:nonNegativeInteger" use="required">
              <xs:annotation>
                <xs:documentation>a sequential number defining the timepoint in
execution</xs:documentation>
              </xs:annotation>
            </xs:attribute>
            <xs:attribute name="tree_node" type="xs:integer" use="required">
              <xs:annotation>
                <xs:documentation>links the state to a node in the search tree, can be -1 if not inside
search</xs:documentation>
              </xs:annotation>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="version" type="xs:string" use="required">
        <xs:annotation>
          <xs:documentation>currently "1.0"</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
</xs:element>
```

attribute **visualization/@version**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation currently "1.0"
source	<xs:attribute name="version" type="xs:string" use="required"> <xs:annotation> <xs:documentation>currently "1.0"</xs:documentation> </xs:annotation> </xs:attribute>

element **visualization/visualizer**

diagram	
properties	isRef 0 minOcc 1 maxOcc unbounded content complex

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xs:string**	required			documentation id is referred to by visualizer_state
type	**xs:string**	required			documentation type of

					visualizer; must be supported on both sizes
	display	**xs:string**	required		documentation how to display the visualizer
	x	**xs:integer**		0	documentation optional explicit placement of visualizer on canvas
	y	**xs:integer**		0	
	width	**xs:integer**	required		
	height	**xs:integer**	required		
	group	**xs:string**			documentation optional parameter, allows grouping of multiple constraints
	min	**xs:integer**		0	documentation expected minimal value of any of the domains
	max	**xs:integer**		0	
annotation	documentation description of a constraint or a set of variables which will be visualized				

source

```
<xs:element name="visualizer" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>description of a constraint or a set of variables which will be visualized</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="id" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>id is referred to by visualizer_state</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>type of visualizer; must be supported on both sizes</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="display" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>how to display the visualizer</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="x" type="xs:integer" default="0">
      <xs:annotation>
        <xs:documentation>optional explicit placement of visualizer on canvas</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="y" type="xs:integer" default="0"/>
    <xs:attribute name="width" type="xs:integer" use="required"/>
    <xs:attribute name="height" type="xs:integer" use="required"/>
    <xs:attribute name="group" type="xs:string">
```

```
        <xs:annotation>
          <xs:documentation>optional parameter, allows grouping of multiple
constraints</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="min" type="xs:integer" default="0">
        <xs:annotation>
          <xs:documentation>expected minimal value of any of the domains</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="max" type="xs:integer" default="0"/>
    </xs:complexType>
  </xs:element>
```

attribute **visualization/visualizer/@id**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation id is referred to by visualizer_state

source	`<xs:attribute name="id" type="xs:string" use="required">`
	`  <xs:annotation>`
	`  <xs:documentation>id is referred to by visualizer_state</xs:documentation>`
	`  </xs:annotation>`
	`</xs:attribute>`

attribute **visualization/visualizer/@type**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation type of visualizer; must be supported on both sizes

source	`<xs:attribute name="type" type="xs:string" use="required">`
	`  <xs:annotation>`
	`  <xs:documentation>type of visualizer; must be supported on both sizes</xs:documentation>`
	`  </xs:annotation>`
	`</xs:attribute>`

attribute **visualization/visualizer/@display**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation how to display the visualizer

source	`<xs:attribute name="display" type="xs:string" use="required">`
	`  <xs:annotation>`
	`  <xs:documentation>how to display the visualizer</xs:documentation>`
	`  </xs:annotation>`
	`</xs:attribute>`

attribute **visualization/visualizer/@x**

type	**xs:integer**
properties	isRef 0 default 0
annotation	documentation optional explicit placement of visualizer on canvas
source	<xs:attribute name="x" type="xs:integer" default="0"> <xs:annotation> <xs:documentation>optional explicit placement of visualizer on canvas</xs:documentation> </xs:annotation> </xs:attribute>

attribute **visualization/visualizer/@y**

type	**xs:integer**
properties	isRef 0 default 0
source	<xs:attribute name="y" type="xs:integer" default="0"/>

attribute **visualization/visualizer/@width**

type	**xs:integer**
properties	isRef 0 use required
source	<xs:attribute name="width" type="xs:integer" use="required"/>

attribute **visualization/visualizer/@height**

type	**xs:integer**
properties	isRef 0 use required
source	<xs:attribute name="height" type="xs:integer" use="required"/>

attribute **visualization/visualizer/@group**

type	**xs:string**
properties	isRef 0
annotation	documentation optional parameter, allows grouping of multiple constraints
source	<xs:attribute name="group" type="xs:string"> <xs:annotation> <xs:documentation>optional parameter, allows grouping of multiple constraints</xs:documentation> </xs:annotation> </xs:attribute>

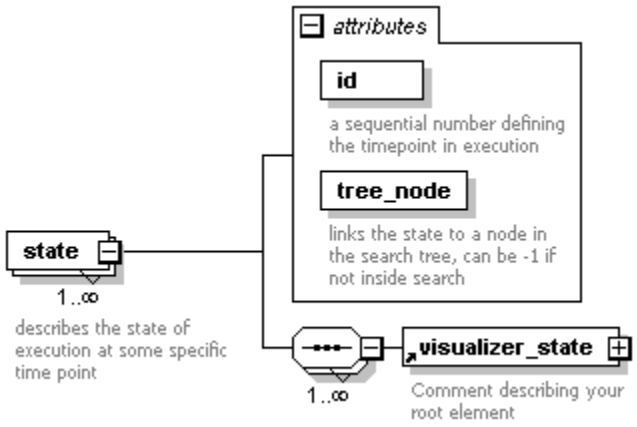
attribute **visualization/visualizer/@min**

type	**xs:integer**
properties	isRef 0 default 0
annotation	documentation expected minimal value of any of the domains
source	<xs:attribute name="min" type="xs:integer" default="0"> <xs:annotation> <xs:documentation>expected minimal value of any of the domains</xs:documentation> </xs:annotation> </xs:attribute>

attribute **visualization/visualizer/@max**

type	**xs:integer**
properties	isRef 0 default 0
source	<xs:attribute name="max" type="xs:integer" default="0"/>

element **visualization/state**

diagram	
properties	isRef 0 minOcc 1 maxOcc unbounded content complex
children	**visualizer_state**

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xs:nonNegativeInteger**	required			documentation a sequential number defining the timepoint in execution
tree_node	**xs:integer**	required			documentation links the state to a node in the search tree, can be -1 if not inside search

annotation	documentation describes the state of execution at some specific time point

source	<xs:element name="state" maxOccurs="unbounded"> <xs:annotation> <xs:documentation>describes the state of execution at some specific time point</xs:documentation> </xs:annotation> <xs:complexType> <xs:sequence maxOccurs="unbounded"> <xs:element ref="visualizer_state"/> </xs:sequence> <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"> <xs:annotation> <xs:documentation>a sequential number defining the timepoint in execution</xs:documentation> </xs:annotation> </xs:attribute> <xs:attribute name="tree_node" type="xs:integer" use="required"> <xs:annotation> <xs:documentation>links the state to a node in the search tree, can be -1 if not inside search</xs:documentation> </xs:annotation> </xs:attribute> </xs:complexType> </xs:element>

attribute **visualization/state/@id**

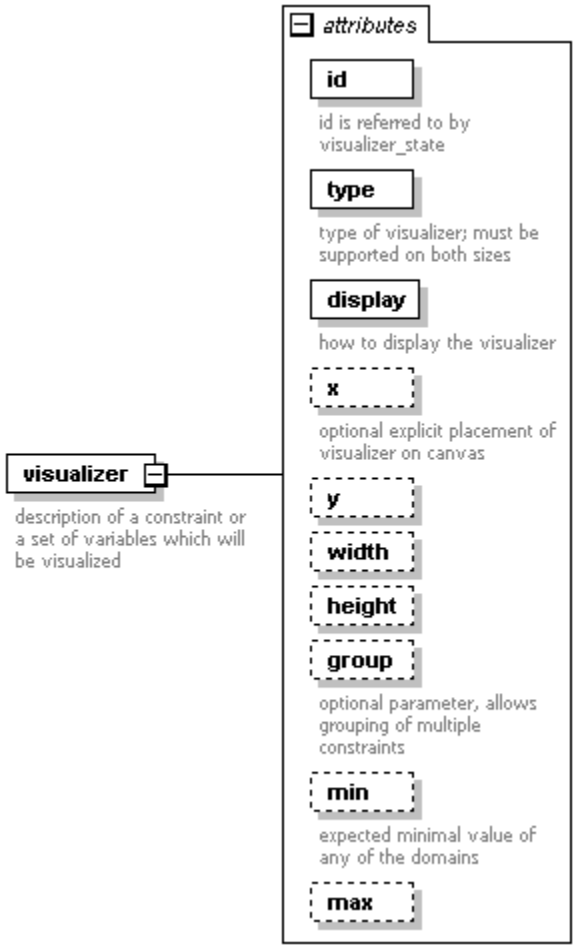
type	**xs:nonNegativeInteger**
properties	isRef 0 use required
annotation	documentation a sequential number defining the timepoint in execution
source	<xs:attribute name="id" type="xs:nonNegativeInteger" use="required"> <xs:annotation> <xs:documentation>a sequential number defining the timepoint in execution</xs:documentation> </xs:annotation> </xs:attribute>

attribute **visualization/state/@tree_node**

type	**xs:integer**
properties	isRef 0 use required
annotation	documentation links the state to a node in the search tree, can be -1 if not inside search
source	<xs:attribute name="tree_node" type="xs:integer" use="required"> <xs:annotation> <xs:documentation>links the state to a node in the search tree, can be -1 if not inside search</xs:documentation> </xs:annotation> </xs:attribute>

element **visualizer**

diagram



properties

content complex

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xs:string**	required			documentation id is referred to by visualizer_state
type	**xs:string**	required			documentation type of visualizer; must be supported on both sizes
display	**xs:string**	required			documentation how to display the visualizer
x	**xs:integer**		0		documentation optional explicit placement of visualizer on canvas
y	**xs:integer**		0		
width	**xs:integer**		0		
height	**xs:integer**		0		
group	**xs:string**				documentation optional parameter, allows

	group	xs:string		grouping of multiple constraints
	min	xs:integer	0	documentation expected minimal value of any of the domains
	max	xs:integer	0	

annotation

documentation
description of a constraint or a set of variables which will be visualized

source

```
<xs:element name="visualizer">
  <xs:annotation>
    <xs:documentation>description of a constraint or a set of variables which will be
visualized</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="id" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>id is referred to by visualizer_state</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>type of visualizer; must be supported on both
sizes</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="display" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>how to display the visualizer</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="x" type="xs:integer" default="0">
      <xs:annotation>
        <xs:documentation>optional explicit placement of visualizer on canvas</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="y" type="xs:integer" default="0"/>
    <xs:attribute name="width" type="xs:integer" default="0"/>
    <xs:attribute name="height" type="xs:integer" default="0"/>
    <xs:attribute name="group" type="xs:string">
      <xs:annotation>
        <xs:documentation>optional parameter, allows grouping of multiple
constraints</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="min" type="xs:integer" default="0">
      <xs:annotation>
        <xs:documentation>expected minimal value of any of the domains</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="max" type="xs:integer" default="0"/>
  </xs:complexType>
</xs:element>
```

attribute **visualizer/@id**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation id is referred to by visualizer_state
source	<xs:attribute name="id" type="xs:string" use="required"> <xs:annotation> <xs:documentation>id is referred to by visualizer_state</xs:documentation> </xs:annotation> </xs:attribute>

attribute **visualizer/@type**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation type of visualizer; must be supported on both sizes
source	<xs:attribute name="type" type="xs:string" use="required"> <xs:annotation> <xs:documentation>type of visualizer; must be supported on both sizes</xs:documentation> </xs:annotation> </xs:attribute>

attribute **visualizer/@display**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation how to display the visualizer
source	<xs:attribute name="display" type="xs:string" use="required"> <xs:annotation> <xs:documentation>how to display the visualizer</xs:documentation> </xs:annotation> </xs:attribute>

attribute **visualizer/@x**

type	**xs:integer**
properties	isRef 0 default 0
annotation	documentation optional explicit placement of visualizer on canvas
source	<xs:attribute name="x" type="xs:integer" default="0"> <xs:annotation> <xs:documentation>optional explicit placement of visualizer on canvas</xs:documentation> </xs:annotation> </xs:attribute>

attribute **visualizer/@y**

type	**xs:integer**
properties	isRef 0 default 0
source	<xs:attribute name="y" type="xs:integer" default="0"/>

attribute **visualizer/@width**

type	**xs:integer**
properties	isRef 0 default 0
source	<xs:attribute name="width" type="xs:integer" default="0"/>

attribute **visualizer/@height**

type	**xs:integer**
properties	isRef 0 default 0
source	<xs:attribute name="height" type="xs:integer" default="0"/>

attribute **visualizer/@group**

type	**xs:string**
properties	isRef 0
annotation	documentation optional parameter, allows grouping of multiple constraints
source	<xs:attribute name="group" type="xs:string"> <xs:annotation> <xs:documentation>optional parameter, allows grouping of multiple constraints</xs:documentation> </xs:annotation> </xs:attribute>

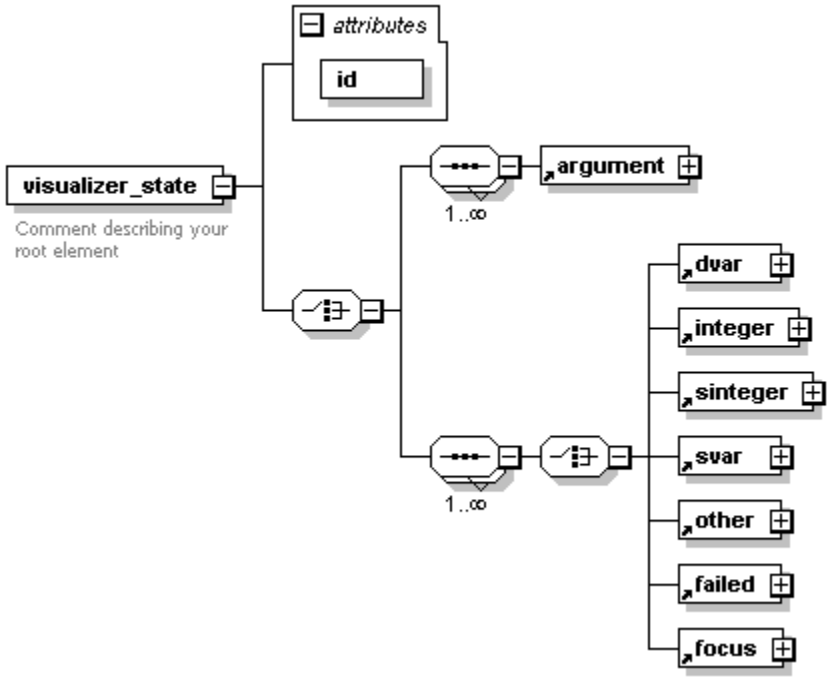
attribute **visualizer/@min**

type	**xs:integer**
properties	isRef 0 default 0
annotation	documentation expected minimal value of any of the domains
source	<xs:attribute name="min" type="xs:integer" default="0"> <xs:annotation> <xs:documentation>expected minimal value of any of the domains</xs:documentation> </xs:annotation> </xs:attribute>

attribute **visualizer/@max**

type	**xs:integer**
properties	isRef 0 default 0
source	<xs:attribute name="max" type="xs:integer" default="0"/>

element **visualizer_state**

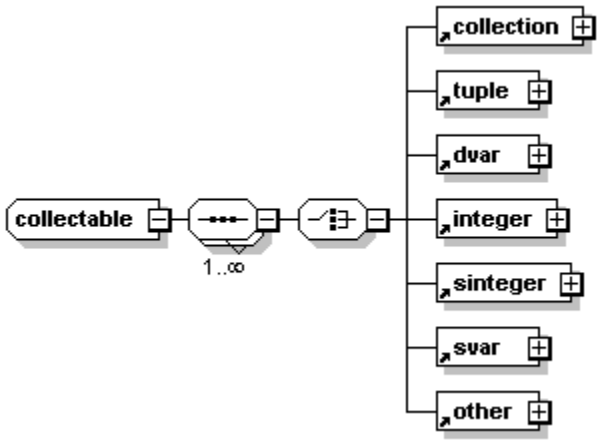
diagram	
properties	content complex
children	**argument dvar integer sinteger svar other failed focus**
used by	elements **visualization/state state**
attributes	Name: id; Type: **xs:nonNegativeInteger**; Use: required; Default; Fixed; annotation
annotation	documentation Comment describing your root element
source	<xs:element name="visualizer_state"> <xs:annotation> <xs:documentation>Comment describing your root element</xs:documentation> </xs:annotation> <xs:complexType> <xs:choice> <xs:sequence maxOccurs="unbounded"> <xs:element ref="argument"/> </xs:sequence> <xs:sequence maxOccurs="unbounded"> <xs:choice> <xs:element ref="dvar"/> <xs:element ref="integer"/> <xs:element ref="sinteger"/> <xs:element ref="svar"/>

```
            <xs:element ref="other"/>
            <xs:element ref="failed"/>
            <xs:element ref="focus"/>
          </xs:choice>
        </xs:sequence>
      </xs:choice>
      <xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>
    </xs:complexType>
</xs:element>
```

attribute **visualizer_state/@id**

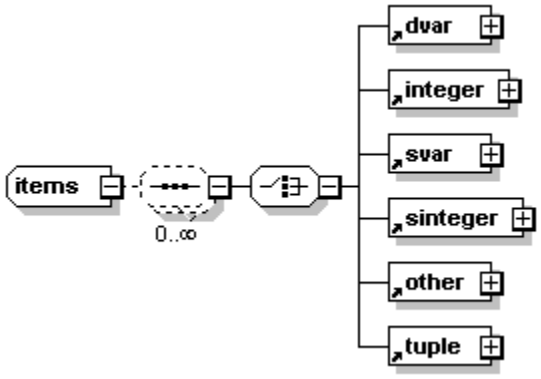
type	**xs:nonNegativeInteger**
properties	isRef 0 use required
source	`<xs:attribute name="id" type="xs:nonNegativeInteger" use="required"/>`

complexType **collectable**

diagram	
children	**collection tuple dvar integer sinteger svar other**
used by	elements **argument collection**

source	`<xs:complexType name="collectable">`
	`  <xs:sequence maxOccurs="unbounded">`
	`  <xs:choice>`
	`  <xs:element ref="collection"/>`
	`  <xs:element ref="tuple"/>`
	`  <xs:element ref="dvar"/>`
	`  <xs:element ref="integer"/>`
	`  <xs:element ref="sinteger"/>`
	`  <xs:element ref="svar"/>`
	`  <xs:element ref="other"/>`
	`  </xs:choice>`
	`  </xs:sequence>`
	`</xs:complexType>`

complexType **items**

diagram	
children	**dvar integer svar sinteger other tuple**
used by	element **tuple**
source	<xs:complexType name="items"> <xs:sequence minOccurs="0" maxOccurs="unbounded"> <xs:choice> <xs:element ref="dvar"/> <xs:element ref="integer"/> <xs:element ref="svar"/> <xs:element ref="sinteger"/> <xs:element ref="other"/> <xs:element ref="tuple"/> </xs:choice> </xs:sequence> </xs:complexType>

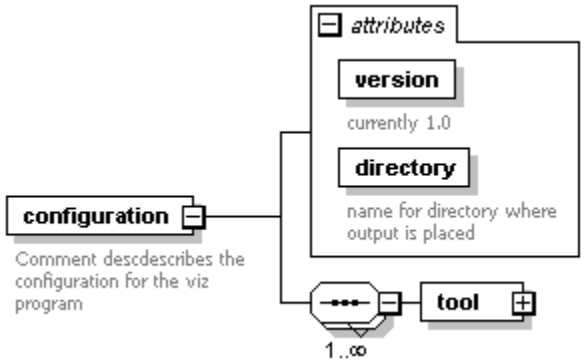
Schema **configuration.xsd**

schema location: **configuration.xsd**
attribute form default: **unqualified**
element form default: **qualified**

Elements
configuration

element **configuration**

diagram



properties
content complex

children
tool

attributes

Name	Type	Use	Default	Fixed	annotation
version	**xs:string**	required			documentation currently 1.0
directory	**xs:string**	required			documentation name for directory where output is placed

annotation
documentation
Comment descdescribes the configuration for the viz program

source

```
<xs:element name="configuration">
  <xs:annotation>
    <xs:documentation>Comment descdescribes the configuration for the viz
program</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="tool">
        <xs:complexType>
          <xs:attribute name="show" type="xs:string" use="required">
            <xs:annotation>
              <xs:documentation>tree or viz</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="type" type="xs:string" use="optional" default="layout">
            <xs:annotation>
              <xs:documentation>layout, distribution, treemap</xs:documentation>
```

```
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="display" type="xs:string" use="optional" default="compact">
            <xs:annotation>
              <xs:documentation>compact or expanded</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="repeat" type="xs:string" use="optional" default="final">
            <xs:annotation>
              <xs:documentation>all, final , i or -i</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="width" type="xs:nonNegativeInteger" use="optional" default="500">
            <xs:annotation>
              <xs:documentation>width of SVG canvas in screen pixels</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="height" type="xs:nonNegativeInteger" use="optional" default="500">
            <xs:annotation>
              <xs:documentation>height of SVG canvas in screen pixels</xs:documentation>
            </xs:annotation>
          </xs:attribute>
          <xs:attribute name="fileroot" type="xs:string" use="required">
            <xs:annotation>
              <xs:documentation>root name of output files</xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="version" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>currently 1.0</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="directory" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>name for directory where output is placed</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

attribute **configuration/@version**

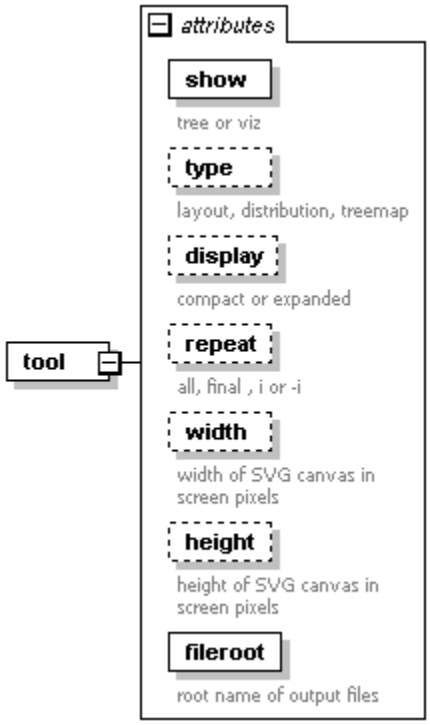
type	**xs:string**
properties	isRef 0 use required
annotation	documentation currently 1.0

source	`<xs:attribute name="version" type="xs:string" use="required">`
	`  <xs:annotation>`
	`  <xs:documentation>currently 1.0</xs:documentation>`
	`  </xs:annotation>`
	`</xs:attribute>`

attribute **configuration/@directory**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation name for directory where output is placed
source	<xs:attribute name="directory" type="xs:string" use="required"> <xs:annotation> <xs:documentation>name for directory where output is placed</xs:documentation> </xs:annotation> </xs:attribute>

element **configuration/tool**

diagram	
properties	isRef 0 content complex

attributes

Name	Type	Use	Default	Fixed	annotation
show	**xs:string**	required			documentation tree or viz
type	**xs:string**	optional	layout		documentation layout, distribution, treemap
display	**xs:string**	optional	compact		documentation compact or expanded
repeat	**xs:string**	optional	final		documentation all, final , i or -i
width	**xs:nonNegativeInteger**	optional	500		documentation width of SVG canvas in

	height	**xs:nonNegativeInteger**	optional	500	screen pixels documentation height of SVG canvas in screen pixels
	fileroot	**xs:string**	required		documentation root name of output files

source

```
<xs:element name="tool">
  <xs:complexType>
    <xs:attribute name="show" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>tree or viz</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type" type="xs:string" use="optional" default="layout">
      <xs:annotation>
        <xs:documentation>layout, distribution, treemap</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="display" type="xs:string" use="optional" default="compact">
      <xs:annotation>
        <xs:documentation>compact or expanded</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="repeat" type="xs:string" use="optional" default="final">
      <xs:annotation>
        <xs:documentation>all, final , i or -i</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="width" type="xs:nonNegativeInteger" use="optional" default="500">
      <xs:annotation>
        <xs:documentation>width of SVG canvas in screen pixels</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="height" type="xs:nonNegativeInteger" use="optional" default="500">
      <xs:annotation>
        <xs:documentation>height of SVG canvas in screen pixels</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="fileroot" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>root name of output files</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

attribute **configuration/tool/@show**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation tree or viz
source	`<xs:attribute name="show" type="xs:string" use="required">`

	<xs:annotation> <xs:documentation>tree or viz</xs:documentation> </xs:annotation> </xs:attribute>

attribute **configuration/tool/@type**

type	**xs:string**
properties	isRef 0 default layout use optional
annotation	documentation layout, distribution, treemap
source	<xs:attribute name="type" type="xs:string" use="optional" default="layout"> <xs:annotation> <xs:documentation>layout, distribution, treemap</xs:documentation> </xs:annotation> </xs:attribute>

attribute **configuration/tool/@display**

type	**xs:string**
properties	isRef 0 default compact use optional
annotation	documentation compact or expanded
source	<xs:attribute name="display" type="xs:string" use="optional" default="compact"> <xs:annotation> <xs:documentation>compact or expanded</xs:documentation> </xs:annotation> </xs:attribute>

attribute **configuration/tool/@repeat**

type	**xs:string**
properties	isRef 0 default final use optional
annotation	documentation all, final , i or -i
source	<xs:attribute name="repeat" type="xs:string" use="optional" default="final"> <xs:annotation> <xs:documentation>all, final , i or -i</xs:documentation> </xs:annotation> </xs:attribute>

attribute **configuration/tool/@width**

type	**xs:nonNegativeInteger**
properties	isRef 0 default 500 use optional

annotation	documentation width of SVG canvas in screen pixels
source	<xs:attribute name="width" type="xs:nonNegativeInteger" use="optional" default="500"> <xs:annotation> <xs:documentation>width of SVG canvas in screen pixels</xs:documentation> </xs:annotation> </xs:attribute>

attribute **configuration/tool/@height**

type	**xs:nonNegativeInteger**
properties	isRef 0 default 500 use optional
annotation	documentation height of SVG canvas in screen pixels
source	<xs:attribute name="height" type="xs:nonNegativeInteger" use="optional" default="500"> <xs:annotation> <xs:documentation>height of SVG canvas in screen pixels</xs:documentation> </xs:annotation> </xs:attribute>

attribute **configuration/tool/@fileroot**

type	**xs:string**
properties	isRef 0 use required
annotation	documentation root name of output files
source	<xs:attribute name="fileroot" type="xs:string" use="required"> <xs:annotation> <xs:documentation>root name of output files</xs:documentation> </xs:annotation> </xs:attribute>