VN - Visualization of Nondeterminism User's Guide

Version 3.2.3

Mordechai (Moti) Ben-Ari
Department of Science Teaching
Weizmann Institute of Science
Rehovot 76100 Israel
http://stwww.weizmann.ac.il/g-cs/benari/

January 8, 2013

1 Introduction

VN is a tool for studying the behavior of nondeterministic finite automata (NDFA). It takes a description of an automaton and generates a nondeterministic program; the program can then be executed randomly or guided interactively. The automaton and the execution path are graphically displayed.

VN is written in Java for portability. It is based on other software tools:

- The program generated from the automaton is written in PROMELA, the language of the SPIN model checker. See: M. Ben-Ari. *Principles of the Spin Model Checker*. Springer, 2008. The program is run using the ERIGONE model checker, which also uses the PROMELA language. `http://code.google.com/p/erigone/`.
- The graphical description of the automaton and path are created in the DOT language and layed out by the DOT tool. Graphs in PNG format are created and are then displayed within VN. DOT is part of the GRAPHVIZ package. `http://graphviz.org/`.

The VN software is copyrighted according the the GNU General Public License. See the files `copyright.txt` and `gpl.txt` in the archive.

The VN webpage is `http://code.google.com/p/v-n/`.

Acknowledgement: Michal Armoni assisted in the design of VN.

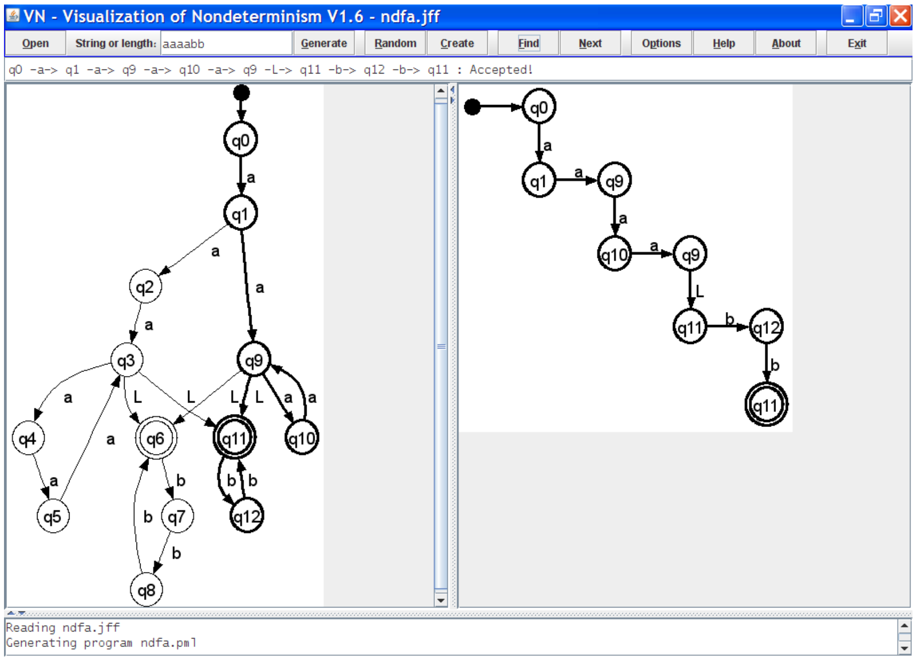
2 Installation and execution

- Install the Java SDK or JRE (`http://java.sun.com`). **VN needs Java 1.5 at least.**
- For Windows: download the VN installation file called `vn-N.exe`, where `N` is the version number, and execute the installation file. For other systems, you will have to build VN, and download and install ERIGONE and DOT.
- The installation will create the following subdirectories: `docs` for the documentation; `vn` for the source files; `bin` for the libraries and executables for ERIGONE and DOT; `txts` for the text files (help, about and copyright); and `examples` for example programs.
- To run VN, execute the command `javaw -jar vn.jar`. An optional argument names the file containing a representation of an automaton. A batch file `run.bat` is supplied which contains this command.
- Configuration data for VN is given in the source file `Config.java`, as well as in the file `config.cfg`. The latter is reset to its default values if it is erased.

- To rebuild VN, execute `build.bat`, which will compile all the source files and create the file `vn.jar` with the manifest file.

3 Interacting with VN

The display of VN is divided into three scrollable panes. Messages from VN are displayed in the text area at the bottom of the screen. Graphs of automata are displayed in the left-hand pane and graphs of the paths are displayed in the right-hand pane. The size of the panes is adjustable and the small triangles on the dividers can be used to maximize a pane. Above the panes is another text area called the path area where execution paths of the automaton are displayed. Interaction with VN is through a toolbar. All toolbar buttons have mnemonics (Alt-character).



aaaabb

```
q0 -a-> q1 -a-> q9 -a-> q10 -a-> q9 -L-> q11 -b-> q12 -b-> q11 : Accepted!
```

```
Reading ndfa.jff
Generating program ndfa.pml
```

Open Brings up a file selector; select a `jff` file and the automaton described in the file will be displayed.

Edit Invokes the grapical editor on the current automaton or on an empty automaton.

String or length Enter an input string for the automaton in this text field. See below on entering a length.

Generate Once an automaton has be opened and an input string entered in the text field, selecting Generate will create a program to execute the automaton.

Random This will execute the program for the automaton with random resolution of nondeterminism. The sequence of states will appear in the text area below the toolbar, along with an indication if the input string was accepted or rejected.

Create This will execute the program for the automaton, resolving nondeterminism interactively. Deterministic choices will be made automatically. If a potential nondeterministic choice cannot be taken, it will be displayed in square brackets and cannot be selected. Quit terminates the execution. Keyboard shortcuts: Tab moves between buttons and Space or Enter selects the highlighted button.

Find This searches for an accepting computation of the automaton.

Next This searches for the next accepting computation of the automaton. When no more accepting computations exist, the number of accepting computations is displayed in the path area.

DFA See below.

Options Displays a dialog for choosing the size of the graphs (small, medium, large) and whether the highlighting of the paths will be in color or bold. When you select OK the changes will be saved, along with the directory of the current open file.

Help Displays a short help file.

About Displays copyright information about the software.

Exit Exit the application.

Once a computation has been found , the *full path* is displayed in the right-hand pane, while the *highlighted path* is displayed in the automaton in the left-hand pane. The difference is that the full path includes multiple visits to the same state, while the highlighted path within the automaton will, of course, include just one state for all occurences.

Searching for all inputs that are accepted

If you enter an integer value in the field String or length, each selection of Next searches for an accepting computation for *any* input string of this length. When no more accepting computations are found, the set of strings that are accepted is displayed in path area. For the automaton `ndfa.jff` supplied in the archive and with input length six, there are seven accepting computations for the four inputs: `aaaaaa, aaaabb, aaabbb, aabbbb`.

Equivalence classes of deterministic finite automata

If the automaton is deterministic, VN can display the partition of the set of input strings of a given length into the equivalence classes associated with each state.[1] Enter a length

[1]This functionality was inspired by ProofChecker:
`http://research.csc.ncsu.edu/accessibility/ProofChecker/index.html`.

in the text field and select Generate. Now select DFA repeatedly; for each state, starting from the first state, the set of input strings "accepted" by that state is displayed. When DFA has been selected for all states, the set of equivalence classes is displayed in the right-hand pane. Generate can be selected at any time to reset to the first state.

For the file `dfa.jff` in the `examples` directory, the equivalence classes of length 4 are:

```
q0: [bbaa, baba, abba, aaaa]
q1: [bbab, babb, abbb, aaab]
q2: [bbba, baaa, abaa, aaba]
q3: [bbbb, baab, abab, aabb]
```

4 The graphical editor

The input to VN is a representation of an NDFA in XML. The representation is that of the JFLAP software tool for studying formal languages and automata. Therefore, JFLAP can be used as an external graphics editor together with VN. In addition, VN contains a graphics editor that is compatible with the JFLAP file format.

The editor has two *tools*: one for creating states and one for creating transitions. Click on the buttons State or Transition to select one of the tools. Initially, the state tool is active. Other buttons are: Save as for saving the NDFA to another file and for the initial save of a new NDFA. Quit terminates the editor without saving the file and Exit saves the file and terminates.

When the save tool is active, clicking on the canvas will create a new state. Right-clicking on the state will bring up a menu to designate the state as initial and/or final, as well as to delete the state.

When the transition tool is active, a transition is created by pointing the cursor at a state, pressing the left mouse button, moving to another state and releasing the button. A self-loop can be created simply by clicking on a state. The default symbol for the transition is `L` for a λ-transition. Right-clicking on the symbol will bring up a text field where a new symbol can be entered. If you delete the symbol, the transition itself will be deleted when OK is clicked.

The left mouse button can be used to drag and drop states or transitions to reformat the NDFA. The state coordinates will be remembered if the automaton is closed and later re-opened.

5 Files

The different phases of processing VN communicate through files. Here is a list of the extensions of these files:

`jff`	XML representation of an automaton
`pml`	Promela source generated from the automaton
`pth`	Path resulting from running SPIN on the `pml` file
`dot`	Graphics file describing automaton and path
`png`	PNG graphics file after layout by DOT

6 Software structure

The source code can be downloaded from GoogleCode: `http://code.google.com/p/v-n/`. If you wish to use obtain source code from the SUBVERSION repository, please note that the current code is in the branch named `vne`; the trunk represents an older version that will not be maintained.

`VN` is the main class and contains declarations of the GUI components and the event handler for all the buttons.

`Config` contains constants and properties.

`Options` displays a dialog for changing the size and highlight of the graphs.

`DisplayFile` displays the files for About and Help in a `JFrame`.

`JFFFileFilter` is used with a `JFileChooser` to open a `jff` file.

`ImagePanel` displays the PNG files.

`GenerateSpin` generates the PROMELA program from the automaton.

`ReadXML` reads the XML description of the automaton from the `jff` file and stores the data in `states` and `transitions`. These types are defined in classes `State` and `Transition`.

`ReadPath` reads the path data printed by the PROMELA program and stores it in `pathStates` and `pathTransitions`.

`WriteGraph` writes the automaton and paths in DOT format and forks a process to run DOT to layout the graph.

`RunSpin` forks a process to execute SPIN. In interactive mode, it reads the PROMELA program so that the `JOptionPane` used for selecting among nondeterministic choices can display actual source code. For Find and Next processes are forked to run the C compiler and `pan`. For Next, `pan` is run with the `-c` argument.

The graphics editor is in the subdirectory `editor`. It was adapted from the AUTOSIM software by Carl Burch: `http://ozark.hendrix.edu/~burch/proj/autosim/index.html`