Graphical User Interface (GUI) for Augur 2

April 8, 2008

Contents

1 Download

Augur 2 and its graphical user interface can be obtained from `http://www.ti.inf.uni-due.de/research/augur`.

2 Introduction

The idea behind Augur 2 is to provide a tool to verify systems with dynamically evolving structure using suitable approximation techniques. Systems of this kind appear in many places: as pointer structures on a program heap or as reconfigurable networks with mobile processes. They are characterized by the creation and deletion of objects and by changes in the system topology during runtime.

Augur 2 takes as input language an expressive specification language: (attributed) graph transformation systems (GTS/AGTS). Graph transformation systems extend static graph structures with the possibility to describe dynamic changes using transformation rules. They are well-suited to describe dynamic behavior, especially of concurrent and distributed systems.

It is often convenient to extend a modelling language by adding attributes which carry values of some data types. Extending GTSs with attributes allows one to combine the intuitive graphical aspects of the modelled systems with natural data structures, which makes such extended GTSs more suitable for practical applications. Usually this leads to more compact models, because many well-known operations need not be described in an artificial way using the graph structures.

Since data types are often infinite, abstraction of data types is needed in order to do automatic verification. This is usually done in one of the following ways: data abstraction, abstract interpretation or predicate abstraction. In Augur 2 the first two approaches are used.

Augur 2 verifies systems described by (attributed) graph transformation systems using approximated unfoldings. The obtained over-approximation consists of an (attributed) Petri net and a hypergraph structure over it. Properties of graph transformation systems can be verified by analyzing the approximation, using regular expressions, first order logic and coverability checking techniques for (attributed) Petri nets.

If the obtained over-approximation is too coarse to verify the property, techniques for refining the approximation are available. One such technique is

counterexample-guided abstraction refinement which starts from a spurious counterexample found by coverability checking. Subsequently the approximation is refined in such a way that this spurious counterexample disappears. Another possibility is to use depth-based refinement, which constructs an over-approximation exact up to a pre-defined depth in the unfolding.

More information on the theoretical background behind AUGUR 2 can be found in [BCK01, BCKar, KK06].

This GUI allows one to use commands of AUGUR 2 via a simple point-and-click interface and to visualize the verification results. Please read the AUGUR 2 documentation for more details about the functionality of the underlying verification procedures. Note also that the GUI does not allow the user to "draw" graph transformation systems. For this we established an interface to the AGG tool (see the corresponding documentation below).

3 Installation

The source code of AUGUR 2is available as a tar.gz archive. After unpacking you should compile the programs by calling `make` from the root directory. In most Linux distributions the library `libxml2` is located in `/usr/include/libxml2/` and `/usr/lib/libxml2.so.2`. If the library is in another directory, please change the file `src/Makefile` (the first three lines) accordingly. You should also check the path to the lpsolve library in the file `src/Makefile`. The default path is `/usr/lib/lp_solve/liblpsolve55.a`. We have experienced some problems by using the precompiled version of the library. In this way we would recommend to compile it from the source code.

The executables can subsequently be found in the directory `/bin` and can be started as shell applications or by using a graphical user interface (GUI).

The recommended way of using AUGUR 2 is with the GUI. The GUI is written in Java and can be called in the following way:

The file `aunfoldGUI.jar` is in the distribution of AUGUR 2, while the source code of the GUI is in the directory `/gui`.

If you use MetricFF (`http://members.deri.at/~joergh/metric-ff.html`) please copy the executable file "ff" into the directory `/bin`.

In the directory `/example` several examples can be found:

1. `/example/1st_order_logic`: Examples of properties in first-order logic format.
2. `/example/agg`: Examples of GTSs in AGG format.
3. `/example/attributes`: Examples of attributed GTSs (AGTSs) in GTXL format.
4. `/example/gtxl`: Example of GTSs in GTXL format.
5. `/example/new_gtxl`: Example of GTSs in the new GTXL format.
6. `/example/spl`: Example of GTSs in single pointer language (SPL) format.

Most of the examples verified with AUGUR 2 are provided in GTXL format.

4 Configuration Steps

You should have LaTeX and Graphviz (`http://www.graphviz.org/`) installed on your computer. Furthermore you need Java (version number ≥ 1.5). In order to start the GUI you should execute `aunfoldGUI.jar`. The exact syntax is:

```
java -jar aunfoldGUI.jar
```

If you are starting the GUI for the first time, please check the options by choosing "Options", followed by "System Options". In the "Options" panel please set the following parameters:

1. Postscript viewer (default `gv`).
2. Web browser (default `firefox`).
3. Editor for XML files (default `emacs`).
4. Editor for text files (default `emacs`).
5. Path to AUGUR 2: Choose the path to your current installation of AUGUR 2 having in its `bin/` directory at least following binaries: augur, sponge. Example path: `/home/user/AUGUR/` (omit the trailing `bin/`).
6. Path to the Working Directory: For this you should set up a new working directory which is different from the directory containing the example graph transformation files. Into this directory all the output and temporary files (XML, Postscript, ...) will be written. Files might be overwritten without a warning. Example path: `work/`.

7. Path to the database: The default database (containing information about the available scenarios and algorithms) is `augur/db/default.xml`. For more information about the database and its usage please read the AUGUR 2 documentation.

After changing the settings in the "Options" panel press OK. In order to check if AUGUR 2 is properly installed choose "System Check" in the "Files" menu.

5 Start Working

The GUI has two modes for unfolding and analysing graph transformation systems and one mode for emulating the behaviour of graph transformation systems:

1. The standard mode is visible in the start window. In this mode each single step of the verification can be controlled.
2. The demo mode starts the complete verification process (including the abstraction refinement loop) by pressing a single button.
3. The simulation mode allows one to apply chosen rules to the current hypergraph.

5.1 Standard Mode

Fig. 1 represents a screenshot of a GUI in the standard mode. Start by choosing a GTS via "GTS Start File". This GTS can be visualized with the "Show GTS"button. If the initial graph or the rules become too large, the size of the graphs can be adjusted via the parameter "Size" (default: 0.4).

Now the unfolding procedure can be started by pressing "Unfold". With the button "Step++" it is also possible to do unfolding step-by-step. The unfolding procedure can be stopped any time by hitting "Stop Unfolding".

After the unfolding procedure has finished you can specify the property to be verified as a regular expression or as a first-order formula in the text box "Property". It might be for example the regular expressions `'Sprv' 'C*' 'Spub'` or `0'Error'0` or the formula `ex1 test (lab(test)='Spub')`. The specification language suitable for the property and the corresponding encoder are set in the database dialog (see Section 6). See also the documentation of AUGUR 2 for more details.

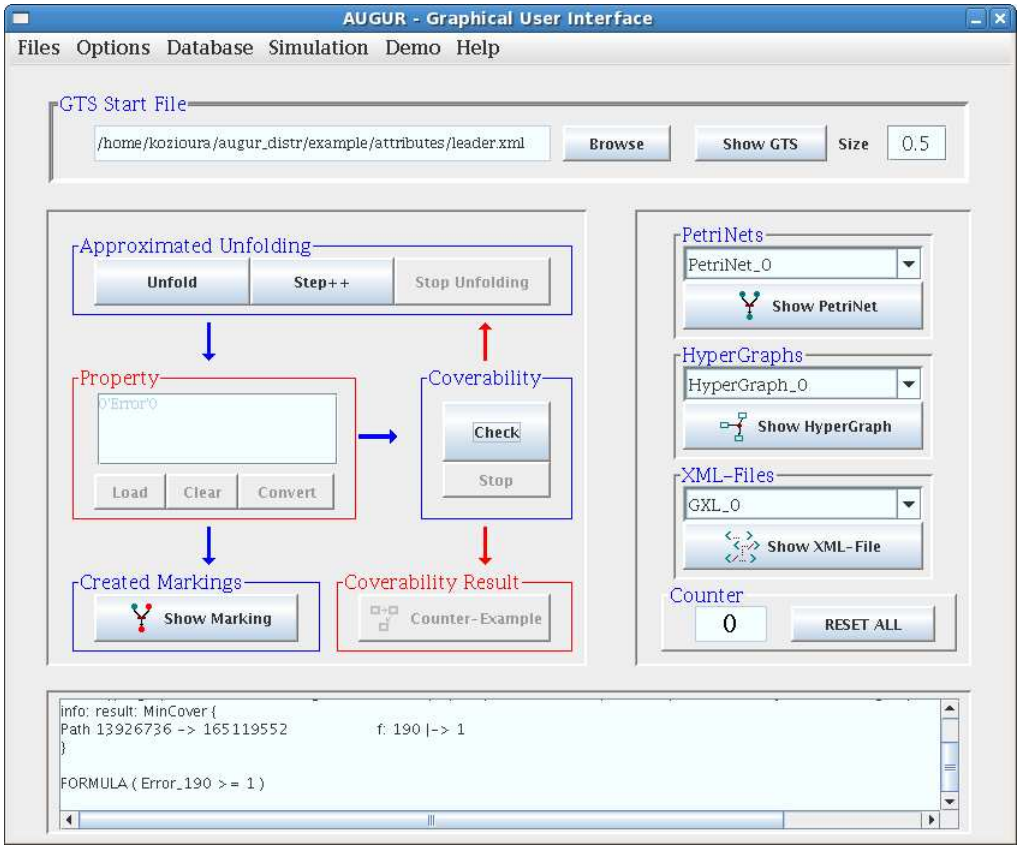

Figure 1: Screenshot of the GUI Panel (standard mode)

Alternatively the property can be loaded from a file. By pressing "Convert" the property will be converted into a marking of a Petri net which can then be inspected via "Show Marking".

As a next step, the coverability of this marking can be checked. There exist several algorithms for doing this check and the algorithm can be chosen in the database dialog (see Section 6). Afterwards one can start a coverability check using the "Check" button. If the coverability check takes too long, it can always be aborted using the corresponding "Stop" button. If the marking is coverable (this will be shown in the output window) you can see the computed trace to the marking by pressing "Counter-Example".

If the marking is coverable (and hence the non-reachability of the property above could not be verified) you can start the next verification cycle (with abstraction refinement) by pressing again "Unfold". AUGUR 2 will automatically choose between different refinement algorithms. "Counter" shows the number of the current verification step.

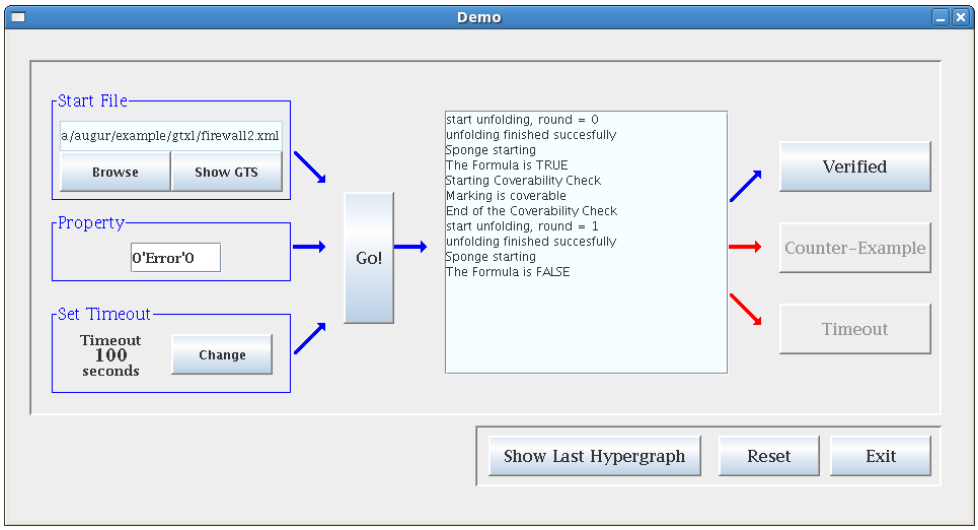

Figure 2: Demo Panel of the GUI

Note however that abstraction refinement does not always integrate well with regular expressions and formulas. It is only guaranteed to work with error rules (and regular expression `0'Error'0`).

On the right-hand side of the panel there are visualization buttons "Show PetriNet" and "Show Hypergraph". With the help of the pull-down menus you can see the Petri nets and hypergraphs obtained after each verification step. You can also inspect the corresponding XML files by pressing "Show XML-file".

With the "Reset all" button or by choosing a new GTS via "Browse" the verification procedure can be restarted.

5.2 Demo Mode

Fig. 2 represents a screenshot of a GUI in demo mode. In order to enter the Demo mode change the view by choosing "Show Demo ..." in the "Demo" menu. You will see a new window where you can choose a "Start File" (containing the GTS), enter some "Property" (regular expression `0'Error'0` is the default value) and "Set Timeout" (default: 100 seconds). Than you can start the whole verification process by simply clicking on the "Go!" button. The tool will make all necessary verification steps automatically.

The possible answers of the tool in the demo mode are:

1. "Verified" - if you have successfully verified the property. In this case you can see the hypergraph obtained in the last step.

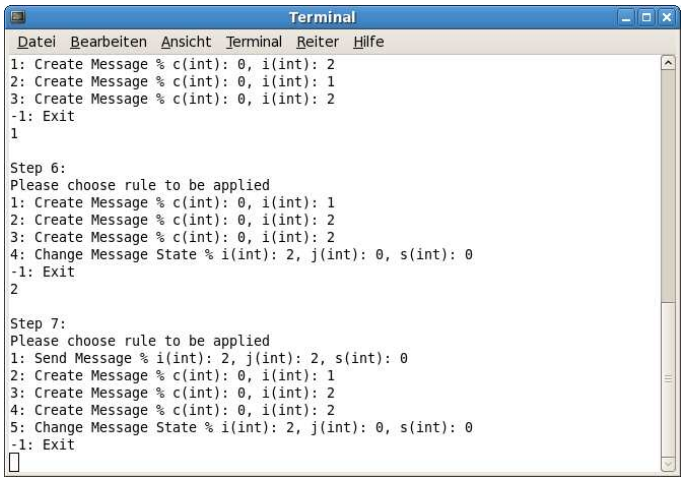

Figure 3: Simulation Mode

2. "Counter-Example" - if some real (non-spurious) counter-example has been found.
3. "Timeout" - if the verification task could not be solved in time due to the predefined timeout.

5.3 Simulation Mode

In order to enter the simulation mode choose "Start Simulation ..." in the "Simulation" menu. Fig. 3 represents a screenshot of a GUI in the simulation mode. The simulation mode works in a standard Gnome terminal.

Here one can start with the initial hypergraph of the considered graph transformation system and choose at each step the rule to be applied. By choosing "Visualize Hypergraph" in the "Simulation" menu one can visualize the hypergraph obtained after each step.

6 Work with the Database

Using the "Database" menu and the dialog panels several important global parameters and algorithms can be easily set (see Fig 4 for the global parameters dialog and Fig 5 for the algorithms dialog).

The following global parameters can be set:

1. "Attributed" - *True* if attributes are used.

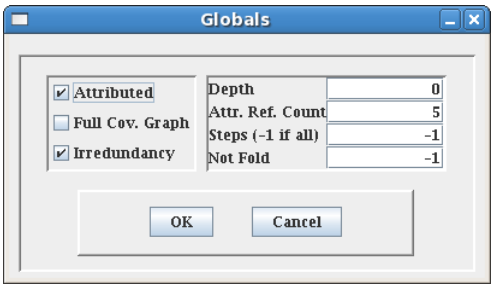

Figure 4: Parameter Dialog of the GUI

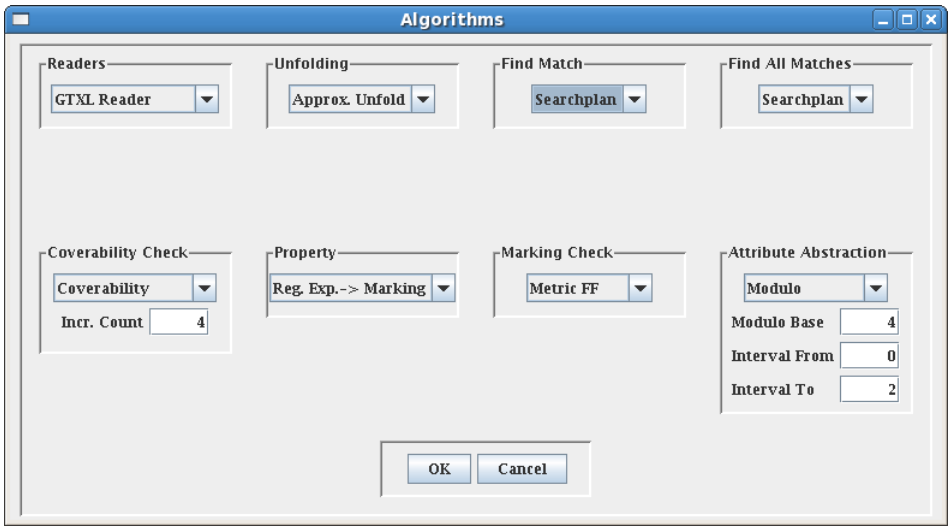

Figure 5: Algorithm Dialog of the GUI

2. "Full Cov.Graph" - *True* if each coverability graph must be completely constructed.
3. "Irredundancy" - *True* if the irredundancy condition is used.
4. "Depth" - Depth of the unfolding (default 0).
5. "Attr.Ref.Count" - Number of attribute refinement steps (default 5).
6. "Steps" - Number of unfolding steps (default -1 — all steps).
7. "Not Fold" - If folding is not used, then the calculation stops after this depth (default -1 — use folding steps).

In the algorithm dialog one can set following properties (see AUGUR 2 documentation for more details on possible algorithms):

1. "Readers" - which reader is used for the GTS/AGTS.

2. “Unfolding ”- which unfolding algorithm is used.
3. “Find Match/Find All Matches” - which match algorithms are used.
4. “Coverability Check” - if one uses a coverability check during the unfolding, then the mode in which this is is done can be chosen here.
5. “Property” - an encoder for properties.
6. “Marking Check” - which Petri net analysis algorithm is used.
7. “Attribute Abstraction” - method of abstraction for attributes.

7 Augur Documentation

1. Documentation of Augur 2
   `http://www.ti.inf.uni-due.de/research/AUGUR/doc_augur_2.pdf`
2. Connection to AGG
   `http://www.ti.inf.uni-due.de/research/AUGUR/agg.pdf`
3. Documentation of Augur 1.1
   `http://www.ti.inf.uni-due.de/research/AUGUR_1/doc_augur_1.pdf`
4. Documentation for the Abstraction Refinement Module of Augur 1.1
   `http://www.ti.inf.uni-due.de/research/AUGUR_1/abstref_doc.pdf`

8 License Agreement

References

[BCK01] Paolo Baldan, Andrea Corradini, and Barbara König. A static analysis technique for graph transformation systems. In *Proc. of CONCUR '01*, pages 381–395. Springer-Verlag, 2001. LNCS 2154.

[BCKar] Paolo Baldan, Andrea Corradini, and Barbara König. A framework for the verification of infinite-state graph transformation systems. *Information and Computation*, to appear.

[KK06] Barbara König and Vitali Kozioura. Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In *Proc. of TACAS '06*, pages 197–211. Springer, 2006. LNCS 3920.