Evidence Explorer: A Tool for Exploring Model-Checking Proofs

Yifei Dong, C.R. Ramakrishnan, and Scott A. Smolka

State University of New York, Stony Brook, NY 11794-4400, USA
{ydong,cram,sas}@cs.sunysb.edu

1 Introduction

We present the Evidence Explorer (`http://www.cs.sunysb.edu/~lmc/ee/`), a new tool for assisting users in navigating the proof structure, or *evidence*, produced by a model checker when attempting to verify a system specification for a temporal-logic property. Due to the sheer size of such evidence, single-step traversal is prohibitive and smarter exploration methods are required. The Evidence Explorer enables users to explore evidence through a collection of orthogonal but coordinated *views*. These views allow one to quickly ascertain the overall perception of evidence through consistent visual cues, and easily locate interesting regions by simple drill-down operations. As described in [3], views are definable in *relational graph algebra*, a natural extension of relational algebra to graph structures such as model-checking evidence.

Our experience in using the Evidence Explorer on several case studies of real-life systems indicates that its use can lead to increased productivity due to shortened evidence traversal time. For example, in the case of formally verifying the Sun Microsystems Java meta-locking algorithm for mutual exclusion and freedom from lockout [1], we had to spend nearly an hour to expand and step through one of the generated model-checking proofs using a standard tree browser. With the Evidence Explorer, we not only cut the process to only a couple of minutes but also were able to recognize the key elements instantly and experiment with the specification via more frequent modifications.

2 Features and User Interface

The Evidence Explorer allows users to effectively and intuitively explore the evidence [8,3] produced by a model checker. In this context, an *evidence* is a directed graph whose nodes are pairs of the form $\langle s, \phi \rangle$, where s is a state and ϕ is a (sub-)formula. Such a node represents the assertion that s satisfies ϕ. Since we use a logic that is closed under negation, the assertion s *does not* satisfy ϕ is represented by $\langle s, \neg\phi \rangle$. An edge in the graph from $\langle s, \phi \rangle$ to $\langle s', \phi' \rangle$ means that s satisfies ϕ only if s' satisfies ϕ'. A global constraint demands that a least fixed point cannot be the outermost fixed point in a loop, *i.e.* no circular reasoning is allowed for least fixed points. Please see [3] for the full definition of evidence.

The views supported by the Evidence Explorer are organized in six windows as illustrated in Figure 1:

W.A. Hunt, Jr. and F. Somenzi (Eds.): CAV 2003, LNCS 2725, pp. 215–218, 2003.

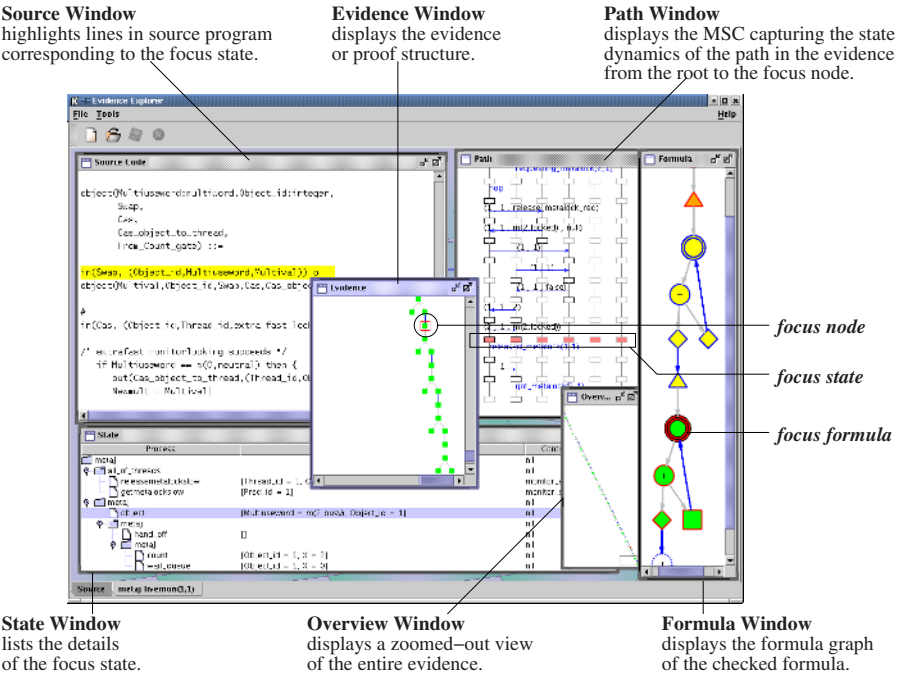

Fig. 1. The six synchronized views of the Evidence Explorer.

– The Evidence and Overview windows display the evidence by its spanning tree in zoomed-in and zoom-out resolutions, respectively. Edges not covered by the spanning tree, *i.e.* cross edges and back edges, are indicated using special icons. Each evidence node is painted in the color its formula component assumes in the Formula window.

– The State window lists the details of the focus state (defined below) in a table. A state in a concurrent system may be composed of several substates; each substate may also be a concurrent state having substates. This hierarchy of parallel composition is depicted as a tree. When the user selects a substate of a sequential process from the table, the line of source code corresponding to that process's control point is highlighted in the Source Code window. The Path window displays a message sequence chart (MSC) capturing the state dynamics of the currently visited path where the focus state belongs.

– The Formula window depicts the formula graph of the checked formula. Several visual cues help users instantly recognize certain properties of the formula and evidence:

  • The *shape* of each formula-graph node is determined by the type of the (sub-)formula's top-level operator, e.g. box for [·] operators, circle-plus for disjunctions, and double-circle for fixed points.

- Formula nodes are partitioned into *groups* according to the scope of their fixed points. Nodes in the same group are painted with the same *background color*.
- The *border color* of a formula node indicates the truth value of the evidence nodes associated with the formula: blue, if the truth value is true; red, otherwise.
- When a formula does *not* occur in an evidence, the background of the corresponding node is set to *transparent* so that the user immediately knows that the formula is *vacuous* [2].

Views are synchronized by the *focus node* in the evidence, consisting of the *focus state* and *focus formula*. If the focus is changed in one window, the other windows will automatically update their displays accordingly. Users explore evidence via various ways of changing the focus node:

– Change the focus node directly in the Evidence window. The focus node is highlighted by a red square. The user can move the focus to the parent, children, or siblings of the current focus by pressing arrow keys, or set the new focus to any node by clicking at that node.
– Change the focus state in the Path window. When the user clicks a state in the MSC, the new focus node is set to the node in the evidence path whose state component is the selected state.
– Change the focus formula in the Formula window. When the user clicks on a formula, the new focus is set to an evidence node whose formula component matches the selected formula.

A typical run of the Evidence Explorer starts with the user observing the overall structure of the evidence in the Overview window. He may click the interesting portion to obtain a zoomed-in view in the Evidence window, or explore the evidence by the methods listed above. For debugging purposes, he may also examine the interesting witness/counterexample represented by the MSC in the Path window or key states in the State and Source Code windows.

3 Implementation and Extensions

The Evidence Explorer is implemented mainly in Java using the AT&T `dot` package as the graph-layout processor. It uses the XMC verification system [7] as the model-checking back-end. XMC supports the specification of systems in a language based on value-passing CCS and properties in the modal mu-calculus. The architecture of the Evidence Explorer follows the model-view-controller (MVC) paradigm [4] as illustrated in Figure 2.

We have designed the Evidence Explorer to be highly extensible and customizable. A planned API will allow tool developers to extend the tool along the following lines:

– utilize alternative graphical components to visualize views;

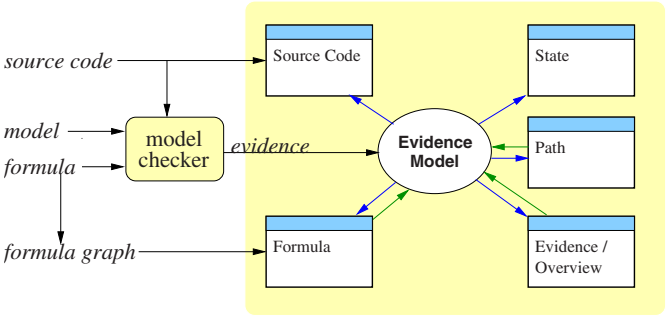

Fig. 2. Architecture of Evidence Explorer.

- define customized views and operations;
- explore multiple evidences;
- plug in other model checkers;
- support compact data structures.

Regarding the last two items, note that we use a conceptual definition of evidence; the physical storage, *e.g.* in a symbolic model checker, can be compressed. When compact data structures are adopted, we map them to the conceptual definition on demand during exploration, without compromising the model checker's efficiency. With this technique, we can also easily incorporate other forms of evidence such as those in [5,6].

References

1. S. Basu, S. A. Smolka, and O. R. Ward. Model checking the Java Meta-Locking algorithm. In *ECBS 2000*, pages 342–350.
2. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in temporal model checking. *Formal Methods in System Design*, 18(2):141–163, 2001.
3. Y. Dong, C. Ramakrishnan, and S. A. Smolka. Model checking and evidence exploration. In *ECBS 2003*, pages 214–223.
4. G. Krasner and S. Pope. A description of the model-view-controller user interface paradigm in the Smalltalk-80 system. *J. Object Oriented Prog.*, 1(3):26–49, 1988.
5. K. S. Namjoshi. Certifying model checkers. In *CAV 2001*, LNCS 2102, pages 2–13.
6. D. Peled, A. Pnueli, and L. Zuck. From falsification to verification. In *FSTTCS 2001*, LNCS 2245, pages 292–304.
7. C. Ramakrishnan, I. Ramakrishnan, S. A. Smolka, Y. Dong, X. Du, A. Roychoudhury, and V. Venkatakrishnan. XMC: A logic-programming-based verification toolset. In *CAV 2000*, LNCS 1855, pages 576–580.
8. L. Tan and R. Cleaveland. Evidence-based model checking. In *CAV 2002*, LNCS 2404, pages 455–470.