From Counterexamples to Incremental Interactive Tracing of Errors

Schrittweise Fehleranalyse auf der Grundlage von Model-Checking

Franz Weitl, Shin Nakajima, National Institute of Informatics (NII), Tokyo, Japan, Burkhard Freitag, University of Passau

Summary
This article summarizes the goals and the first results of the project CITE – from Counterexamples to Interactive Incremental Tracing of Errors – at the National Institute of Informatics (NII) in Tokyo, Japan. CITE aims at developing the fundamental methods for generating both comprehensive and comprehensible error reports based on model checking results. Model checking is a powerful and efficient method for finding flaws in hardware designs, object-oriented software, business processes, and hypermedia applications. One remaining major obstacle to a broader application of model checking is its limited usability for non-experts. It requires much effort and insight to determine the root cause of errors from counterexamples generated by model checkers in the case of a specification violation. The CITE project addresses this problem by proposing tree structured error reports that can be refined successively to the desired level of detail according to the user’s interest. First evaluations demonstrate that the proposed approach reveals more errors and explains the cause of errors more accurately than the counterexamples of existing model checkers.

1 Introduction
The design and development of dependable systems that are provably correct w. r. t. some unambiguous specification is one of the major concerns in software engineering. Model checking is an efficient formal method for automatically verifying hardware and software processes. Here, a process definition is typically represented by a finite state transition system $M$, and a target property $\phi$. Model checking repeatedly executes the program on all paths satisfied by $\phi$ to check whether all possible executions satisfy $\phi$, i.e., whether $M, w \models \phi$.
is represented by a temporal logic formula $f$. A model checker determines whether $M$ is a model of $f$ and provides a counterexample if $M$ does not satisfy $f$. By model checking, errors in complex systems composed of concurrent processes could be detected [3].

In the development of hard- and software, numerous documents are created that describe the product. It is vital for the usability, safe operation, and maintainability of the product that its documentation is correct, complete, and easy to understand, i.e., the documentation should be consistent and coherent.

In our previous work [6; 8], we have shown that model checking, when being combined with methods of semantic modeling, information extraction, and user support, is an efficient method to verify the consistency and coherence of semi-structured documents. Standard methods such as XML document validation are outperformed in expressiveness and speed. A remaining challenge, however, is the generation of error reports that localize errors precisely and explain well why some property is violated.

The CITE project – from Counterexamples to Interactive Incremental Tracing of Errors – aims at improving the quality of counterexamples generated by model checking. We propose a new type of counterexamples for the temporal description logic $\text{ALC} @ \text{CTL}$ [7], a decidable combination of the branching time temporal logic $\text{CTL}$ [3] and the description logic $\text{ALC}$ [1]. These counterexamples support the interactive exploration of error scenarios: An initially provided coarse error description is extended successively at locations selected by the user. This way, structured and complete descriptions of error scenarios are generated without overwhelming the user with details.

2 Problem and Approach

The counterexamples provided by current model checkers are not sufficient for generating concise and precise error reports because
1. they are not complete, i.e., they do not reveal all errors,
2. they may become large and difficult to analyze, and
3. they do not explain why a formula is violated.

We solve these problems by incrementally generating counterexamples along the term structure of a violated formula, according to the user’s interest.

For illustration, let us consider a web-based course book in computer science, containing various definitions of concepts. Since different definitions of the same concept may confuse the reader, we require that each concept is defined only once on any path of browsing the content. This property can be expressed in $\text{ALC} @ \text{CTL}$ [7] as follows:

$$\text{AG(Definition} \sqsubseteq \text{HasTopic).}$$

$$\neg \text{EX EF } \exists \text{topicOf. Definition}) \tag{1}$$

“It must always be the case (AG) that for each definition it holds (Definition $\sqsubseteq$): every topic of the definition ($\text{HasTopic}$) is, from the next page on, never ($\neg \text{EX EF}$) a topic of a definition again ($\exists \text{topicOf. Definition}$).”

$\text{AG, EX, EF are CTL temporal connectives.}$

In the context of web documents, they express that some property holds always (AG), holds on some next page (EX), and holds on some page reachable from the current page (EF), respectively.

$\sqsubseteq$ (read “subsumes”), $\forall$, and $\exists$ are first order quantifiers of implicit variables, inherited from the description logic $\text{ALC}$. $\text{Definition}$ is a concept representing the set of definitions on a given page. $\text{hasTopic}$ and $\text{topicOf}$ are roles representing the binary relation between document parts and their topics [7].

Suppose that formula (1) does not hold. Then, as a first step, a page $p_1$ is determined on which the subsumption expression $\text{Definition} \sqsubseteq \text{hasTopic}$... of formula (1) is not satisfied, in other words, there are definitions on page $p_1$ that define something that is re-defined later. Next, the set $D$ of these faulty definitions is determined. The user may now be interested which topics of a selected definition $d_1 \in D$ are re-defined. Therefore, she issues a refinement request for definition $d_1$ on page $p_1$. The system returns the set of topics $T$ that violate the expression $\neg \text{EX EF } \exists \text{topicOf. Definition}$. To determine where a specific topic $t \in T$ is re-defined, the user issues another refinement request for topic $t$ of definition $d_1$ on page $p_1$. The system responds with a path from page $p_1$ to a page $p_2$ on which $t$ is an instance of $\text{topicOf. Definition}$, i.e., $t$ is topic of another definition. A final refinement request for topic $t$ on page $p_2$ yields the prohibited second definition $d_2$ of topic $t$. At this stage, a complete description of all objects and locations involved in a selected error scenario has been provided. After that, the user may analyze different error scenarios in the same step-by-step manner.

3 First Results

As a formal basis of incremental interactive generation of counterexamples, we introduced the notion of evidence trees as opposed to linear counterexamples of previous approaches [10]. Evidence trees are hierarchically structured counterexamples and witnesses that allow drilling down the cause of a property violation along the term structure of the checked formula. Evidence trees precisely identify the locations and the objects contributing to an error scenario, and offer a more complete description of an error scenario than previous notions of counterexamples. Their hierarchical structure allows generating error reports at different levels of detail which prevents overwhelming the user with bulky unstructured information.

A sound and complete algorithm for generating evidence trees has been defined, implemented in Java, and evaluated in the domain of document verification [10]. The generated counterexamples helped to identify about twice as many errors than those generated by the state-of-the-art model checker NuSMV [2]. The algorithm delivered competitive runtime results, matching or even outperforming those of NuSMV.
In [9] we extend the algorithm for generating evidence trees towards incremental expansion of counterexamples and witnesses in interaction with the user as sketched in Sect. 2. To ensure rapid response in interactive use, the algorithm makes use of intermediate results of our model checking algorithm for $\mathbf{ALC}^\mathbf{CTL}$ [7]. In an experiment on web-based e-learning documents, the maximum response time remained below 100 ms even in the case of documents with more than 4000 html pages and evidence trees with more than 7000 nodes. Although the hierarchical structure of the provided evidence helps to keep the overview in the case of complex error scenarios, a more compact symbolic representation becomes desirable when evidence trees grow beyond comprehensible dimensions. This symbolic representation of evidence trees will be one of the major topics of future research.

4 Related Research at the NII

The CITE project correlates with other research projects of the second author at the National Institute of Informatics (NII). In recent work, model checking has been applied to analyze business process models [5] and C programs in industrial settings [4]. One of the current research topics is satisfiability modulo theories (SMT), an extension of the satisfiability problem of propositional logic with background theories, for instance for integer arithmetic. SMT provides the basis to solve problems in software model checking. Integrating methods developed for model checking $\mathbf{ALC}^\mathbf{CTL}$ into the framework of SMT may help to make them applicable to the verification of dynamic web-applications or software processes the models of which are usually much larger than those of static documents.

5 Conclusion

Model checking is emerging from a powerful but hard to handle formal method to a tool that enables solving problems in rather informal domains such as technical documentation. To obtain detailed, structured, and comprehensible error reports, methods for generating counterexamples need to be improved. We propose generating counterexamples incrementally in interaction with the user. First evaluations demonstrate that more structured and precise error reports are obtained in the domain of document verification. To be applicable also to the verification of software, the integration of the developed methods into the framework of satisfiability modulo theories is targeted in future work.

References


Received: July 6, 2010

Dr. Franz Weitl is a DAAD fellow and postdoc at the National Institute of Informatics (NII) in Tokyo. He received his doctoral degree from the University of Passau in 2008. His research interests are in new applications of model checking such as document and business process verification.

Address: National Institute of Informatics (NII), 2-1-2 Hirosenbashii, Chiyoda-ku, Tokyo 101-8430, Japan, e-mail: weitl@nii.ac.jp

Prof. Dr. Shin Nakajima is professor at the National Institute of Informatics (NII) in Tokyo. His research is concerned with software engineering for dependable systems. He is particularly interested in formal methods including formal specification languages and logic model checking. Furthermore, he is involved in several projects to deploy formal methods in industry in Japan.

Address: National Institute of Informatics (NII), 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan, e-mail: nkjm@nii.ac.jp

Prof. Dr. Burkhard Freitag is professor of computer science at the University of Passau. His research interests are in the areas of databases and information systems. Currently, his work concentrates on document management and context-sensitive information systems using methods from formal logic, database systems and knowledge representation.

Address: University of Passau, Innstrasse 43, D-94032, Passau, Germany, e-mail: burkhard.freitag@uni-passau.de