χChek: A Model Checker for Multi-Valued Reasoning

Steve Easterbrook, Marsha Chechik, Benet Devereux, Arie Gurfinkel, Albert Lai,
Victor Petrovykh, Anya Tafliovich and Christopher Thompson-Walsh
Department of Computer Science
University of Toronto, Toronto, Canada M5S 3H5
xchek@cs.toronto.edu

This paper describes our multi-valued symbolic model-checker χChek. χChek is a generalization of an existing symbolic model-checking algorithm for a multi-valued extension of the temporal logic CTL. Multi-valued model-checking supports reasoning with values other than just TRUE and FALSE.

Multi-valued logics are useful in software engineering because they support explicit modeling of uncertainty, disagreement, and relative desirability or priority. For example, 3-valued logics have been used for interpreting results of static analysis with abstraction [5, 10], and for analyzing partial models [1]. The intermediate value of the logic is used to denote missing information. 4-valued logics have been used to model disagreements that arise when models drawn from different sources are composed [6]. The four values represent the four possible ways of combining the two classical values of the source models.

Our model checker generalizes these approaches — it works for the class of multi-valued logics whose logical values form a finite distributive lattice, and where there is a suitably defined negation operator that preserves De Morgan laws and involution ($\neg\neg a = a$). Such structures are called *quasi-boolean algebras* [9]. Classical logic, as well as the 3- and 4-valued logics described in the literature, are examples of quasi-boolean algebras. In [3], we describe the properties of these logics. For tractability, we restrict ourselves to logics with a finite number of values.

Examples of these logics are shown in Figure 1. 1(a) is classical 2-valued logic. 1(b) is a 3-valued logic suitable for representing partial models. 1(c) is the 4-valued logic describe above. Note that its lattice is the product of two 2-valued lattices. 1(d) is the product of two 3-valued lattices, and is suitable for composing two *partial* models allowing both disagreement *and* missing information.

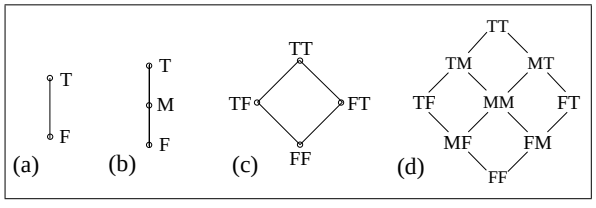

Figure 1. Some example lattices.

Implementation

χChek is implemented in Java, and provides support for both model-checking with fairness and the generation of counter-examples (or witnesses). The tool consists of three components: (1) the model-checking engine itself (χChek); (2) a counter-example generator (KEG); (3) a web-based front-end for interactive exploration and visualization of counter-examples (KegVis).

χChek receives a χKripke structure (a multi-valued generalization of a Kripke structure) K and a χCTL formula φ, and produces a value of φ at every state of K. The modular implementation of χChek allows it to support a wide variety of specification languages for χKripke structures. Currently, these structures can be specified either explicitly, as directed graphs in XML, or as compositions of modules expressed in an SMV-like notation.

The analysis is performed using different decision diagrams: MDDs and MBTDDs implemented as a custom decision diagram package [4], as well as BDDs and ADDs using the standard CUDD library. The complexity of model-checking of a χCTL formula φ, under the assumption that all operations on decision diagrams take constant time, is $O(|S| \times |\varphi|)$, where S is the state space of the model [7].

[3] presents a more detailed description of the architecture of χChek. The tool itself can be downloaded from http://www.cs.toronto.edu/fm.

Applications

Multi-valued model-checking has a number of potential applications in software engineering.

The intermediate values of the logic can represent incomplete information (or uncertainty). Such applications typically use a 3-valued logic. A 3-valued model can be

interpreted as a compact representation for a set of *completions* [1], where a completion is generated by replacing each M value in the model by either T or F. If a property is T (respectively, F) in a partial model, then it is T (F) in all completions. If a property is M in a partial model, then it takes different values in different completions; the missing information affects the property. Thus, χChek can determine if a property holds, even though the model is incomplete. We can also use this approach to reduce the size of classical model-checking problems by creating (partial) abstractions of models that have large state-spaces. The approach can be generalized to logics with more than 3 values, to distinguish levels of uncertainty for the incomplete information, but we have not yet explored such applications.

The intermediate values of the logic can represent disagreement. Such applications typically use product lattices. A model based on a product lattice can be interpreted as a compact representation for a set of models (or *views*), where the views may disagree on the values of some transitions or propositions. For example, a 4-valued model based on the lattice of Figure 1(c) can be formed by merging information from two 2-valued views. Where the views disagree on the value of a transition or proposition, it will take the value TF or FT. If a property is TT (respectively, FF) in each individual view, then it will be TT (FF) in the merged model. If a property is FT or TF in the merged model, then the disagreement affects the property. χChek can check properties that cannot be expressed in the individual views, because the properties combine vocabulary of several views or refer to interactions between different views. We are exploring this approach for the feature interaction problem in telephony [6], and as a tool to support stakeholder negotiations in requirements engineering.

The intermediate values of the logic can represent relative desirability (or criticality). Such applications typically use chain lattices (total orders). A model based on a chain lattice can be interpreted as a compact representation for a set of partial *layers*, where each successive layer specifies values for transitions left unspecified by previous layers. For example, a model based on a 4-valued chain lattice can be used to represent a system with two levels of criticality. Transitions labeled T and F represent core functionality—transitions that must (or must not) occur. Transitions labeled with other values represent optional functionality. If a property is T (respectively, F) in this model, then it is true (false) in just the core layer, irrespective of behaviors at the optional layer. We are exploring this approach for reasoning about requirements prioritization and survivable systems. χChek allows us to check which properties are supported by which layer, without having to maintain separate models of the individual layers.

Elements of our quasi-boolean algebras need not be interpreted as logical values. Consider the *query-checking* problem [2] for which the inputs are a (classical) model and a temporal logic query (TLQ). A TLQ is a temporal logic formula with placeholders for some subformulas, e.g., $AG?$. A query-checker finds the strongest set of assignments of propositional formulas for each placeholder, such that replacing each placeholder with any assignment chosen from its set gives a temporal logic formula that holds in the model. Thus, query-checking can be used to discover invariants, guards, and postconditions of (sets of) transitions in the model. The query checking problem can be formulated as a multi-valued model checking problem, where the elements of the lattices are sets of propositional formulas ordered by set inclusion [8].

Acknowledgements: *Financial support was provided by NSERC and CITO.*

References

[1] G. Bruns and P. Godefroid. "Generalized Model Checking: Reasoning about Partial State Spaces". In C. Palamidessi, editor, *Proc. 11th Int. Conf. on Concurrency Theory (CONCUR'00)*, volume 1877 of *LNCS*, pp 168–182, University Park, PA, USA, Aug 2000. Springer.

[2] W. Chan. "Temporal-Logic Queries". In E. Emerson and A. Sistla, editors, *Proc. 12th Conf. on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pp 450–463, Chicago, IL, USA, July 2000. Springer.

[3] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. "Multi-Valued Symbolic Model-Checking". *ACM Trans. on Software Engineering and Methodology*, Jan 2003. (Accepted for publication).

[4] M. Chechik, B. Devereux, S. Easterbrook, A. Lai, and V. Petrovykh. "Efficient Multiple-Valued Model-Checking Using Lattice Representations". In *Proc. 12th Int. Conf. on Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pp 451–465, Aalborg, Denmark, Aug 2001. Springer.

[5] M. Chechik and W. Ding. "Lightweight Reasoning about Program Correctness". *Information Systems Frontiers*, 4(4), Nov 2002.

[6] S. Easterbrook and M. Chechik. "A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints". In *Proc. Int. Conf. on Software Engineering (ICSE'01)*, pp 411–420, Toronto, Canada, May 2001. IEEE CS Press.

[7] A. Gurfinkel. "Multi-Valued Symbolic Model-Checking: Fairness, Counter-Examples, Running Time". Master's thesis, U. of Toronto, Dept. of Computer Science, Oct 2002.

[8] A. Gurfinkel, B. Devereux, and M. Chechik. "Model Exploration with Temporal Logic Query Checking". In *Proc. SIGSOFT Conf. on Foundations of Software Engineering (FSE'02)*, pp 139–148, Charleston, SC, Nov 2002.

[9] H. Rasiowa. *An Algebraic Approach to Non-Classical Logics. Studies in Logic and the Foundations of Mathematics*. Amsterdam: North-Holland, 1978.

[10] M. Sagiv, T. Reps, and R. Wilhelm. "Parametric Shape Analysis via 3-Valued Logic". In *Proc. 26th Annual ACM Symp. on Principles of Programming Languages*, pp 105–118, New York, NY, 1999.