# 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

## Abstract

*This paper describes our multi-valued symbolic model-checker χChek. Multi-valued model-checking generalizes classical model-checking and is useful for analyzing models where there is uncertainty (e.g. missing information) or inconsistency (e.g. disagreement between different views). Multi-valued logics support the explicit modeling of uncertainty and disagreement by providing additional truth values in the logic. χChek works for any member of a large class of multi-valued logics. Our modeling language is based on a generalization of Kripke structures, where both atomic propositions and transitions between states may take any of the truth values of a given multi-valued logic. Properties are expressed in χCTL, our multi-valued extension of the temporal logic CTL. This paper gives a brief summary of the model checker and describes some applications.*

## 1. Introduction

This paper describes our multi-valued symbolic model-checker χChek. χChek [6] is a generalization of an existing symbolic model-checking algorithm for a multi-valued extension of the temporal logic CTL.

A classical model-checker takes a model, $M$, of a system (expressed as a finite state machine), and a temporal correctness property, $\varphi$, (expressed as a formula in a suitable temporal logic), and determines whether the model satisfies the property, i.e., it returns the value of the relation $M \models \varphi$. Multi-valued model-checking is a generalization of classical model-checking for 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.

The advantages of automated reasoning with multiple values have been recognized by a number of researchers.

For example, a 3-valued logic has been used for interpreting results of static analysis with abstraction [7, 13], and for analyzing partial models [1, 2]. In the latter case, the intermediate value of the logic is used to denote missing information, and checking the models allows the analyst to determine whether the desired properties are preserved in all refinements of this system.

A 4-valued logic has been used to model disagreements that arise when we compose two models drawn from different sources [9]. The four values of the logic represent the four possible ways of combining the two classical values of the source models. By checking a model in which disagreements are explicitly represented, the analyst can reason about how such disagreements affect various temporal properties and thus support potential negotiation.

Our model checker generalizes these approaches — it works for a large class of multi-valued logics, including classical (two-valued) logic. The class of logics we use are those 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 lattices are called *quasi-boolean*, and the resulting structures are called *quasi-boolean algebras* [12]. Classical logic, as well as the 3- and 4-valued logics described in the literature, are examples of quasi-boolean algebras. In [4], we describe the properties of these logics, showing that the logical operators have the expected properties (associativity, commutativity, idempotence, etc). However, some classical laws are lost, such as the law of excluded middle, and the law of non-contradiction. For tractability, we restrict ourselves to logics with a finite number of values.

Some example logics are shown in Figure 1. Figure 1(a) is classical 2-valued logic. Figure 1(b) is a 3-valued logic suitable for representing partial models. Figure 1(c) is the 4-valued logic describe above. Note that its lattice is the product of two 2-valued lattices. Figure 1(d) is the product of two 3-valued lattices, and is suitable for composing two *partial* models where there may be both disagreement *and*
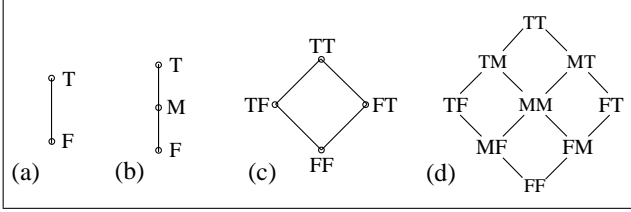
**Figure 1. Some example lattices.**

missing information.

In the next section we present a short example, to show how multi-valued logics can be used in modeling. Section 3 briefly describes the implementation of $\mathcal{X}$Chek. Section 4 outlines a number of different applications.

## 2. Example

We illustrate the use of $\mathcal{X}$Chek on a simple example of a thermostat controller. The thermostat is described using two aspects: Heater and Air Conditioner (AC). The Heater aspect is responsible for activating the heat when the temperature drops below desired, and the AC aspect is responsible for activating the air conditioning. We first model each of the aspects individually, and then merge them to produce a model of the thermostat.

The Heater aspect, shown in Figure 2(a), consists of a switch to turn the thermostat on and off (`Running`), a temperature indicator (`Below`), and a variable indicating whether the heater is on (`Heat`). Notice that in the states `OFF` and `IDLE`$_1$, the current temperature is unknown. This could be modeled by splitting these states, assigning `Below` a value T in one copy and F in another. Instead, we model this using a 3-valued logic, assigning `Below` the value M. The AC aspect, shown in Figure 2(b), is similar. The resulting models are generalized Kripke structures, called $\mathcal{X}$Kripke structures, where both transitions and state variables are assigned values from a multi-valued logic.

We merge the two aspects to construct a single model of the thermostat, shown in Figure 2(c). The composition that was chosen for this example is similar to parallel asynchronous composition with a special treatment of shared states. First, we identify the states `OFF` and `IDLE`$_1$ as shared, thus requiring that they can only be merged with themselves. Second, we add an environmental constraint that `Above` $\wedge$ `Below` is not *true*, making the state (`Heat`, `AC`) unreachable in the composition.

For a logic of composition, we choose the logic **3x3**, shown in Figure 1(d). Values of state variables in a merged state are computed as follows: a value of a shared variable is a tuple formed from values of this variable in the original aspects. For example, the value of `Running` in state (`OFF`, `OFF`) is (F,F), which we write as FF. A value of a variable that is local to one aspect is a tuple where all elements

| Property | Result |
|---|---|
| $E[\neg\texttt{Below}\ U\ (\neg\texttt{Below} \wedge \texttt{Heat})]$ | FF |
| $AG(\texttt{Heat} \rightarrow \neg\texttt{Air})$ | TT |
| $AG(\texttt{Heat=TT} \rightarrow EXEX(\texttt{Running=FF})$ | TT |

**Table 1. Verification results.**

are equal to the value this variable has in the "host" aspect. For example, the value of `Below` in state (`IDLE`$_2$, `AC`) is MM because `Below` has value M in the Heater aspect and is not present in the AC aspect. A transition between two states $(s_1, t_1)$ and $(s_2, t_2)$ is also multi-valued, and defined as $(R_1(s_1, s_2), R_2(t_1, t_2))$, where $R_i(x, y)$ is the value of the transition between states $x$ and $y$ in system $i$. For example, the transition between (`IDLE`$_2$, `IDLE`$_2$) and (`IDLE`$_1$, `IDLE`$_1$) is FT because the transition between `IDLE`$_2$ and `IDLE`$_1$ in the Heater aspect is F and in the AC aspect is T. This value denotes disagreement between the two aspects on the value of the transition. We annotate transitions with their values, but omit FF transitions to avoid clutter. The resulting composition is shown in Figure 2(c).

For this example, we identify the following three properties: (1) Is the heat ever turned on before the temperature falls below desired? (2) Is heat on only if air conditioning is off? (3) When the system is heating, can it reach the `OFF` state in two steps? The formalization of these properties in $\mathcal{X}$CTL is given in Table 1.

We use $\mathcal{X}$Chek to verify the properties, with the results shown in Table 1. The first property can be verified directly on the Heater aspect, the second can only be verified on the combined model, and the third can be verified on either aspect. Thus, the result TT for the third property is interpreted to mean that the property is T in either of the aspects. However, since the combined system still contains disagreements, it is possible that the two aspects agree on the value of the property but disagree on the *reason* why it holds. $\mathcal{X}$Chek helps us discover this by generating a witness, shown in Figure 2(d). As in the classical case, such witnesses can be trees [10]. This witness shows that the property is satisfied in the Heater aspect because the system can move to `OFF` directly from `HEAT`, and then remain in the `OFF` state indefinitely. On the other hand, the AC aspect requires the system to transition to `IDLE`$_1$, and only then proceed to `OFF`. Moreover, since our counter-example generator is guaranteed to produce a single common execution if one exists [10], it is clear that the disagreement does affect this property, and further analysis may be needed to determine whether this is a problem.

## 3. Implementation

$\mathcal{X}$Chek 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
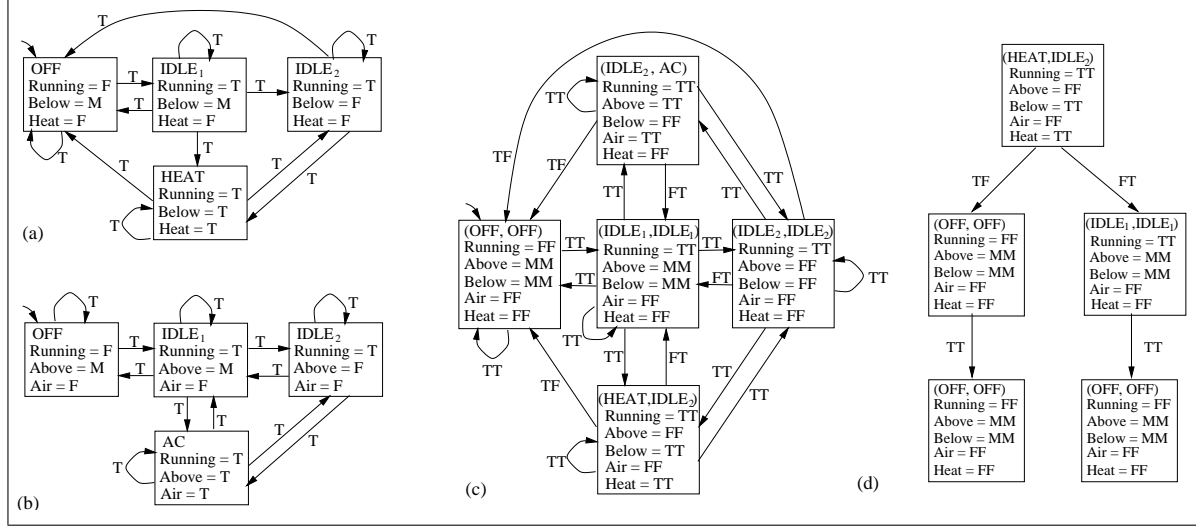
**Figure 2. Models of the thermostat. (a) Heater aspect; (b) AC aspect; (c) a combined model over logic $3\times3$; (d) witness for a temporal logic property.**

components: (1) the model-checking engine itself ($\mathcal{X}$Chek); (2) a counter-example generator (KEG); (3) a web-based front-end for interactive exploration and visualization of counter-examples (KegVis).

$\mathcal{X}$Chek receives a $\mathcal{X}$Kripke structure (a multi-valued generalization of a Kripke structure) $K$ and a $\mathcal{X}$CTL formula $\varphi$, and produces a value of $\varphi$ at every state of $K$. The modular implementation of $\mathcal{X}$Chek allows it to support a wide variety of specification languages for $\mathcal{X}$Kripke 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 later enables $\mathcal{X}$Chek to verify SMV models as well as abstractions and merges of these models.

The analysis is performed using different decision diagrams: MDDs and MBTDDs implemented as a custom decision diagram package [5], as well as BDDs and ADDs using the standard CUDD library. The complexity of model-checking of a $\mathcal{X}$CTL formula $\varphi$, 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 [10].

Please see [4, 8] for a more detailed description of the architecture of $\mathcal{X}$Chek. The tool itself can be downloaded from `http://www.cs.toronto.edu/fm`.

## 4. Applications

Multi-valued model-checking has a number of potential applications in software engineering, for analyzing models that contain uncertainty, disagreement, or relative priority, and for general model exploration.

The intermediate values of the logic can represent incomplete information (or uncertainty). Such applications typically use a 3-valued logic, with the values T, F and M ("Maybe"). A 3-valued model can be interpreted as a compact representation for a set of *completions* [2], 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, we can use $\mathcal{X}$Chek to 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. It is possible to generalize this approach 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 quasi-boolean algebras defined over 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 separate 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. Multi-valued model checking over such models is particularly useful if the views are partial, representing, for example, different modules, features or slices of a larger system. In this case, $\mathcal{X}$Chek can

3

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 [9], and as a tool to support stakeholder negotiations in requirements engineering by tracing from specific disagreements to the properties they affect.

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 the intermediate 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. $\mathcal{X}$Chek 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 [3] 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 is a form of model exploration—it 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 on *upset* lattices, where the elements of the lattices are sets of propositional formulas ordered by set inclusion. The reduction of the query-checking problem to multi-valued model-checking problem is described in [11].

## Acknowledgements

## References

[1] G. Bruns and P. Godefroid. "Model Checking Partial State Spaces with 3-Valued Temporal Logics". In *Proc. 11th Int. Conf. on Computer-Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 274–287, Trento, Italy, 1999. Springer.

[2] 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*, pages 168–182, University Park, PA, USA, Aug 2000. Springer.

[3] 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*, pages 450–463, Chicago, IL, USA, July 2000. Springer.

[4] 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).

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

[6] M. Chechik, B. Devereux, and A. Gurfinkel. "$\mathcal{X}$Chek: A Multi-Valued Model-Checker". In *Proc. 14th Int. Conf. on Computer-Aided Verification (CAV'02)*, LNCS, pages 505–509, Copenhagen, Denmark, July 2002. Springer.

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

[8] M. Chechik, A. Gurfinkel, B. Devereux, A. Lai, and S. Easterbrook. "Symbolic Data Structures for Multi-Valued Model-Checking". CSRG Tech Report 446, University of Toronto, Jan 2002.

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

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

[11] 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)*, pages 139–148, Charleston, SC, Nov 2002. ACM Press.

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

[13] 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*, pages 105–118, New York, NY, 1999. ACM.