χChek: A Multi-Valued Model-Checker

Marsha Chechik, Arie Gurfinkel, and Benet Devereux

Department of Computer Science, University of Toronto,
Toronto, ON M5S 3G4, Canada.
Email: {chechik,arie,benet}@cs.toronto.edu

1 Introduction

χChek is a multi-valued symbolic model-checker [CDE01a,CDEG01]. It is a generalization of an existing symbolic model-checking algorithm to an algorithm for a multi-valued extension of CTL (χCTL). Given a system and a χCTL property, χChek returns the *degree* to which the system satisfies the property. By multi-valued logic we mean a logic whose values form a *finite quasi-boolean distributive lattice*. The meet and join operations of the lattice are interpreted as the logical *and* and *or*, respectively. The negation is given by a lattice dual-automorphism with period 2, ensuring the preservation of involution of negation ($\neg\neg a = a$) and De Morgan laws. For example, a 3-valued logic of abstraction ($\mathbf{3}$), consisting of values *true* (T), *maybe* (M), and *false* (F), is given in Figure 1(a), where the negation operator is defined as: $\neg$T = F, $\neg$F = T, and $\neg$M = M.

Multi-valued logics have a wide range of uses in modeling. For example, the logic $\mathbf{3}$ allows for a natural representation of abstract or partial systems. In this case, the value *maybe* is used to represent state variables that are not known to be either *true* or *false*. Furthermore, additional values can be added to the logic to support better granularity, often useful for iterative refinement of partial systems. Although there are several techniques for verifying such systems [BG99], to our knowledge, χChek is the first symbolic model-checker that attacks this problem directly, without reducing it to several classical model-checking problems first.

Since a product of quasi-boolean logics is also quasi-boolean, these logics are the natural choice for reasoning about disagreements and inconsistencies between different viewpoints (also known as aspects, or features). With quasi-boolean logics, it is possible to combine different viewpoints into a single system without first resolving their inconsistencies or assigning priorities to them [EC01]. For example, if we are interested in combining two partial viewpoints, each specified using the logic $\mathbf{3}$, we can use the product logic $\mathbf{3x3}$ (see Figure 1(b)) to represent their composition. In this case, the logic value TF represents information specified to be *true* in the first viewpoint, and *false* in the second; MT represents information that is underspecified in the first viewpoint, and is *true* in the second, etc. Properties of such a composition can then be verified using χChek. Furthermore, counter-examples can help the analyst identify the *important* disagreements, that is, disagreements that affect the properties of interest. The results of this analysis can guide a negotiation between stakeholders to resolve the important disagreements first, leaving resolution of other disagreements until later stages of the design process.

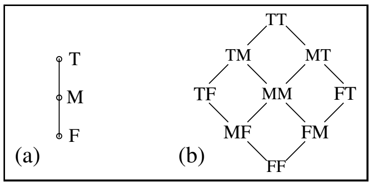

Fig. 1. Some multi-valued logics: (a) $\mathbf{3}$ – 3-valued logic of abstraction; (b) $\mathbf{3x3}$ – a product logic.

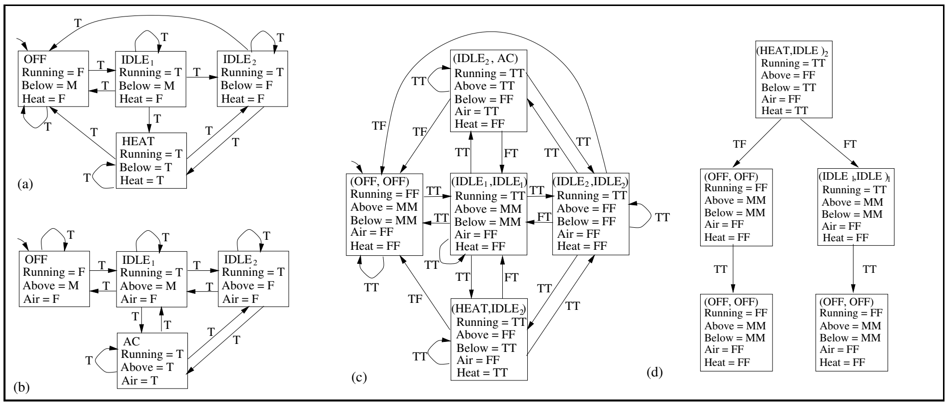

Fig. 2. Models of the thermostat. (a) Heater aspect; (b) AC aspect; (c) a combined model over logic $\mathbf{3x3}$; (d) witness for a temporal logic property.

Furthermore, χChek does not require that the logic values be interpreted as truth values. For example, the logic values can be interpreted as sets of propositional formulae, thus making χChek a solver for CTL query checking problems [BG01,GDC02].

Verification using χChek proceeds similarly to verification using a classical model-checker, with the most complicated part being the correct formalization of the properties. Although χChek does not provide any tools to address this problem, all of the standard techniques, such as property patterns of Dwyer et al [DAC99], can be used with it. This is possible because χCTL is syntactically equivalent to CTL, and semantically derived from CTL by replacing existential quantification by disjunction, and universal quantification by conjunction. This definition of χCTL ensures that χChek is equivalent to a classical model-checker when the classical boolean logic is used for the analysis [CDEG01].

2 Overview and Example

We illustrate the use of χChek 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 the final model of the thermostat.

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

Table 1. Verification results.

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

In the final step of our construction, we merge the two aspects to construct a monolithic 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 global (or shared) states. First, we identify the states OFF and $\texttt{IDLE}_1$ as global, thus requiring that they can only be merged with themselves. Second, we add an environmental constraint that $\texttt{Above} \wedge \texttt{Below}$ is not *true*, making the state $(\texttt{Heat}, \texttt{AC})$ unreachable in the composition.

As a logic of composition, we choose the logic $\mathbf{3x3}$, shown in Figure 1(b). 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 $(\texttt{OFF}, \texttt{OFF})$ 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 are equal to the value this variable has in the "host" aspect. For example, the value of Below in state $(\texttt{IDLE}_2, \texttt{AC})$ 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 $(\texttt{IDLE}_2, \texttt{IDLE}_2)$ and $(\texttt{IDLE}_1, \texttt{IDLE}_1)$ is FT because the transition between $\texttt{IDLE}_2$ and $\texttt{IDLE}_1$ 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, omitting FF transitions. The resulting composition is shown in Figure 2(c).

For the purpose of 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 in the state $(\texttt{HEAT}, \texttt{IDLE}_2)$, can it reach the state $(\texttt{OFF}, \texttt{OFF})$ in two steps? The formalization of these properties in χCTL is given in Table 1.

Finally, we use χChek to verify the properties. The results of the verification on the combined system are summarized in Figure 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. χChek helps us discover this problem by generating a witness, shown in Figure 2(d). A witness in the multi-valued case does not necessarily correspond to a single execution – it might instead be a tree, as in Figure 2(d). This witness shows that the property is satisfied in the Heater aspect because it is possible for the system to evolve into the `OFF` state via a single transition from the `HEAT` state, and then remain in the `OFF` state indefinitely. On the other hand, the AC aspect requires the system to first evolve into the $\mathtt{IDLE}_1$ state, and only then proceed to the `OFF` state. Moreover, since our counter-example generator is guaranteed to produce a single common execution if one exists, it follows that this disagreement is important, and additional negotiation is required. Further analysis shows that the source of the problem is our decision to make $\mathtt{IDLE}_1$ a global (or shared) state across the aspects.

3 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 later enables χChek to verify SMV models as well as abstractions and merges of these models.

The actual analysis is performed using different decision diagrams: MDDs and MBTDDs implemented as a custom decision diagram package [CDE$^+$01b], 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 $O(|h| \times |S| \times |\varphi|)$, where h is the height of the lattice used in the model, and S is its state space. Depending on the problem at hand, choosing the right type of a decision diagram may significantly improve the performance of the algorithm. A detailed comparison between different decision diagrams is available in [CGD$^+$02].

It is important to note that any multi-valued model-checking problem can be reduced to several classical model-checking problems. In fact, if a property φ does not contain negation, then model-checking of this property using χChek with BDDs is exactly equivalent to solving several classical problems. If φ does contain negation, then one must expand the state space by introducing a variable $\overline{a}$ for every atomic proposition a, and require the assertion $\overline{a} = \neg a$ to hold in every state [BG00]. Then, for every property with negation over the original model, there is an equivalent property without negation over the expanded model. Alternatively, a more memory-efficient solution can be achieved if all of the classical problems are solved at the same time. The description of this approach is available in [CGD+02].

Please see [CDEG01,CGD+02] for a more detailed description of the architecture of χChek and its application to several realistic case studies. The tool itself can be downloaded from `http://www.cs.toronto.edu/fm`.

References

[BG99] G. Bruns and P. Godefroid. "Model Checking Partial State Spaces with 3-Valued Temporal Logics". In *Proceedings of CAV'99*, volume 1633 of *LNCS*, pages 274–287, 1999.

[BG00] G. Bruns and P. Godefroid. "Generalized Model Checking: Reasoning about Partial State Spaces". In *Proceedings of CONCUR'00*, volume 877 of *LNCS*, pages 168–182, August 2000.

[BG01] G. Bruns and P. Godefroid. "Temporal Logic Query-Checking". In *Proceedings of 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 409–417, 2001.

[CDE01a] M. Chechik, B. Devereux, and S. Easterbrook. "Implementing a Multi-Valued Symbolic Model-Checker". In *Proceedings of TACAS'01*, volume 2031 of *LNCS*, pages 404–419. Springer, April 2001.

[CDE+01b] M. Chechik, B. Devereux, S. Easterbrook, A. Lai, and V. Petrovykh. "Efficient Multiple-Valued Model-Checking Using Lattice Representations". In *Proceedings of CONCUR'01*, volume 2154 of *LNCS*, pages 451–465. Springer, August 2001.

[CDEG01] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. "Multi-Valued Symbolic Model-Checking". CSRG Tech Report 448, University of Toronto, October 2001.

[CGD+02] 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, January 2002.

[DAC99] M. Dwyer, G. Avrunin, and J. Corbett. "Patterns in Property Specifications for Finite-State Verification". In *Proceedings of 21st International Conference on Software Engineering*, May 1999.

[EC01] S. Easterbrook and M. Chechik. "A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints". In *Proceedings of International Conference on Software Engineering (ICSE'01)*, pages 411–420, May 2001.

[GDC02] A. Gurfinkel, B. Devereux, and M. Chechik. "Model Exploration with Temporal Logic Query Checking". CSRG Technical Report 445, University of Toronto, Department of Computer Science, March 2002.