A Framework to Handle Horizontal Inconsistency in Software

Steve Easterbrook **Marsha Chechik**
Department of Computer Science
University of Toronto
6, King's College Rd
Toronto, Canada M5S 3H5
+1 416 978 3610
{sme,chechik}@cs.toronto.edu
February 27, 2001

During requirements elicitation, different stakeholders often hold different (and incompatible) views of how the proposed system should behave, resulting in inconsistencies between their descriptions. These inconsistencies are referred to as *horizontal*.

Viewpoints-based approaches have been proposed as a way of managing inconsistent and incomplete information gathered from multiple sources [8]. These approaches separate the descriptions provided by different stakeholders, and concentrate on identifying and resolving conflicts between them. A key advantage to the use of viewpoints is that inconsistencies between viewpoints can be tolerated [7].

Given an inconsistent set of viewpoints, we would like to know how the inconsistencies affect critical system properties. Some inconsistencies may be of little consequence, and do not need to be resolved, while others may be the result of fundamental disagreements about how a system should behave. For example, if the stakeholders descriptions are modeled as state machines, they may disagree about the details of some states and transitions, without affecting the values of global temporal properties. We cannot rely on reasoning about the properties of individual viewpoints and comparing the results, because these properties may change depending on how the viewpoints are combined. Worse still, we may not even be able to *express* global properties over individual viewpoints, because each viewpoint is only a partial model of the system.

Thus, we have the following goals:

- Enable a variety of merges of inconsistent viewpoints. The merges are determined by the roles the viewpoints play in the overall description of the system – they may describe different features of the same system, give different descriptions of the same functionality, or specify individual processes that need to be composed in parallel.
- Reason about disagreements automatically, using model-checking.
- Use counter-examples to discover inconsistencies that matter and help the negotiation process.

We can merge inconsistent viewpoints using classical logic, e.g., put a transition in the resulting model if either viewpoint includes it (non-determinism, or optimistic merge), or if both viewpoints include it (conjunction, or pessimistic merge). However, these merges are abstractions, and thus do not preserve certain global properties. Furthermore, merging state information classically is impractical – anything follows from $A \wedge \neg A$. Therefore, if we are interested in defining a number of interesting compositions in the presence of inconsistency, classical logic is clearly too limiting.

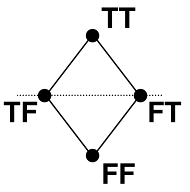
Instead, we propose the use of multi-valued logics. Multi-valued logics, where truth values are used to represent different levels of agreement, are paraconsistent logics, i.e., logics that permit some contradictions to be true, without resulting in trivilization. For example, if we keep the usual boolean values TRUE and FALSE to mean 'a unanimous yes' and 'a unanimous no', we can add any number of intermediate values to represent different kinds of disagreement. Examples include 'a majority said yes', '4 yeses and 1 no', 'nobody knows', 'the designated expert said no, everyone else said yes'. By adding these explicitly as values in the logic, we can reason about the level of agreement for any arbitrary proposition.

The choice of values to use in the logic depends on how we wish to combine information from individual viewpoints. There are a number of dimensions to this choice:

- Do we wish to preserve information about how many viewpoints gave each different response (i.e. so '4 trues, 1 false' is distinct from '3 trues, 2 falses')?
- Do we wish to preserve information about *who* said what (i.e. so that 'A said true, B said false' is distinct from 'A said false, B said true')?
- Do we wish to allow viewpoints to say 'don't know' for some propositions?
- Do we wish to designate some viewpoints as authorities, so that their answers count more?

Because it may be useful to explore different ways of combining information from multiple viewpoints during requirements modeling, we do not commit ourselves to any particular multi-valued logic. Rather, our framework uses a family of logics, and each viewpoint has a specific logic associated with it.

Quasi-Boolean logics have the characteristics necessary to effectively capture and reason about effects of horizontal



$\vee$	TT	TF	FT	FF
TT	TT	TT	TT	TT
TF	TT	TF	TT	TF
FT	TT	TT	FT	FT
FF	TT	TF	FT	FF
$\neg$	FF	FT	TF	TT

Figure 1: The logic **4val** as a lattice, together with the truth tables for disjunction, $\vee$, and negation, $\neg$.

inconsistency. Quasi-Boolean logics preserve many useful properties of classical logic, e.g. associativity, idempotency, distributivity, and De Morgan's laws. However laws relating to incompleteness and inconsistency, such as the law of excluded middle ($a \vee \neg a = \top$) and the law of non-contradiction ($a \wedge \neg a = \bot$), may be discarded.

We achieve this by defining each logic using a lattice of truth values, and define the logical operators in terms of lattice operations. Lattices are useful in this respect because they guarantee most of the properties we need. For example, a lattice by definition includes a unique least upper bound (a *join*) and a unique greatest lower bound (a *meet*) for each pair of elements. By defining disjunction as join, and conjunction as meet in the lattice, we ensure that a disjunction and a conjunction of each pair of values exists and is unique in the logic. Furthermore, these operations are commutative, associative and idempotent in a lattice. To see how the corresponding truth tables can be constructed from the lattice, consider the 4-valued logic **4val** shown in Figure 1. The truth table for disjunction was constructed by taking the join for each pair of values. The table for conjunction is formed similarly using the meet. By specifying the lattice, we avoid having to give the tables.

As well as conjunction and disjunction, we need to specify negation so that most of its expected properties hold: each element must have a negation which is a value in the lattice, such that $\neg\neg a = a$, and negation satisfies De Morgan's laws. The family of multi-valued logics that have these properties (but not necessarily the laws of excluded middle and non-contradiction) are called *quasi-boolean* logics [1]. Rather than providing a formal definition here[1], we note that symmetry of the lattice diagram across its horizontal axis is a sufficient condition for a quasi-boolean logic, where the negation of each element is defined to be its image through horizontal symmetry. Finally, we still have to choose a negation for values that fall on the axis of symmetry. For example, in Figure 1 we chose to make TF and FT negations of each other; we could equally have chosen to make them their own negations, which would have given us a different logic.

In summary, we restrict ourselves to quasi-boolean logics, and specify them as horizontally-symmetric lattices of truth values, together with a definition of negation. Note that classical 2-valued boolean logic is included in this family as a 2-valued lattice — we refer to it as **2val**.

[1] A complete treatment can be found in [5].

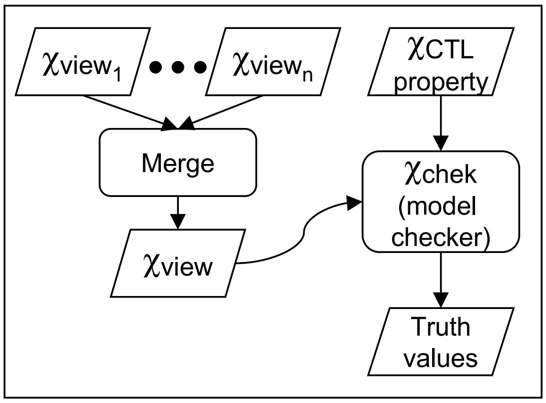


Figure 2: The process model.

Over the past year we have been developing χbel [6] – a framework for merging and reasoning about inconsistent state machine models using multi-valued logics. The framework provides the following components:

- A basic viewpoint model, or χview, which is a state machine model, extended for the multi-valued case.
- Interconnection primitives for defining how to combine χviews into a merged model. These handle differences in vocabulary across the χviews and support a range of different types of interaction between them.
- A set of merge templates, to guide the choice of multi-valued logic and interconnection primitives for different types of analysis.
- Multi-valued logics for reasoning over χviews.
- A symbolic multi-valued model checker, χchek [5, 2, 3], for verifying the temporal properties of χviews and for generating counter-examples.
- A strategy for using the counter-examples to discover disagreements.

Our analysis process is shown in Figure 2. We take a set of source χviews and merge them using a set of interconnection primitives and a merge template chosen by the analyst. The resulting merged χview can then be model checked against a set of properties expressed in χCTL, our multi-valued temporal logic, an extension of CTL. The model checker returns the value(s) from the logic that each property takes in the initial state(s). We use the same multi-valued state machine notation for both the source χviews and the merged χview. This enables us to run the model checker on individual χviews, as well as the merged ones, and to perform further merges on an already merged χview.

The status of this project is as follows:

- χchek is almost completely operational. Please contact us for a copy.
- A small library of merges and relationships between them and CTL properties has been defined.

- Current interest is in negotiation strategies, improvement of the library of merges, and better method guidance.
- Interested in merging at different levels of abstraction and in using belief models rather than multi-valued logics to support reasoning.
- Very interested in case studies.

ACKNOWLEDGEMENTS

The ideas presented in this paper were formed in collaboration with members of the University of Toronto formal methods reading group. Special thanks go to Benet Devereux, Victor Petrovykh, Albert Lai, Arie Gurfinkel, Christopher Thompson-Walsh and Ric Hehner.

REFERENCES

[1] L. Bolc and P. Borowik. *Many-Valued Logics*. Springer-Verlag, 1992.

[2] M. Chechik, B. Devereux, and S. Easterbrook. "Implementing a Multi-Valued Symbolic Model-Checker". In *Proceedings of TACAS'01*, April 2001.

[3] M. Chechik, B. Devereux, S. Easterbrook, A. Lai, and V. Petrovykh. "Efficient Multiple-Valued Model-Checking Using Lattice Representations". Submitted for publication, January 2001.

[4] M. Chechik, S. Easterbrook, and V. Petrovykh. "Model-Checking Over Multi-Valued Logics". In *Proceedings of FME'01*, March 2001.

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

[6] S. Easterbrook and B. Nuseibeh. "Using Viewpoints for Inconsistency Management". *BCS/IEE Software Engineering Journal*, pages 31–43, January 1996.

[7] A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. "Inconsistency Handling in Multi-Perspective Specifications". *IEEE Transactions on Software Engineering*, 20(8):569–578, August 1994.