Techniques for Abstraction and Symmetry in Model Checking: A Survey

Kathi Fisler
Rice University
kfisler@cs.rice.edu

April 29, 1999

Contents

1 Abstraction and Symmetry: An Overview **2**

2 What is an Abstraction Framework? **2**
2.1 Relating Abstract and Concrete Models 3

3 Existing Abstraction Frameworks **4**
3.1 Kurshan 4
3.2 Clarke, Grumberg, and Long 6
3.3 Dams, Gerth, and Grumberg 7
3.4 Cleaveland, Iyer, and Yankelevich 8
3.5 Kelb, Dams, and Gerth 8
3.6 Dams 9
3.7 VIS 10
3.8 Nitsche and Wolper 10
3.9 Ip and Dill 10
3.10 Lee, Pardo, Jang, Hachtel, and Somenzi 11
3.11 Pardo and Hachtel 11
3.12 Graf and Loiseaux 12

4 What is a Symmetry Framework? **12**

5 Existing Symmetry Frameworks **13**
5.1 Clarke, Filkorn, Enders, and Jha 13
5.2 Ip and Dill 14
5.3 McMillan and Aziz 14
5.4 Gyuris and Sistla 15
5.5 Emerson, Jha, and Peled 15

This document summarizes existing research on abstraction and symmetry for model checking. It provides a general overview to both areas, summaries of the literature in each area, and a list of references.

1 Abstraction and Symmetry: An Overview

Abstraction and symmetry are two techniques for state space reduction. Abstraction hides details about a system, while symmetry collapses parts of the state space that are structurally equivalent. Mathematically, symmetry is a special case of abstraction, as discussed in detail below. In practice, rather different methods are needed for using these two techniques. Symmetry can be handled automatically, while most abstraction frameworks require a degree of user guidance (usually in the form of suggesting what to abstract and how to abstract it). Symmetry has been much more successful for explicit state than for symbolic model checking. Abstraction frameworks are often discussed in the context of symbolic model checking, though we are not aware of any outright comparisons between explicit state and symbolic model checking with respect to abstraction.

The existing research in both areas is, for the most part, still immature. Experimental results are lacking for many frameworks, and comparative studies are practically nonexistent. Despite the theoretical ability to perform symmetry reduction automatically, the required intermediate information results in extremely large BDDs, making the practical application of these methods questionable for symbolic model checking. Several avenues of further research and experimentation therefore exist in both of these areas.

2 What is an Abstraction Framework?

An *abstraction framework* must provide three things:

1. A method for relating abstract and concrete models,
2. A logic for stating properties so that those properties preserved under abstraction can be characterized, and
3. A method (hopefully efficient) for obtaining abstract models.

Several variants exist for each of these parameters. Methods for relating models generally fall into one of the following four categories, listed in increasing order of generality. The definitions of and precise relationships between these items appear in section 2.1.

- Bisimulation (used mainly for symmetry)
- Homomorphism
- Galois Insertion
- Simulation (equivalent to a Galois connection)

Given a property logic, two types of property preservation are considered: *weak preservation* (everything true in the abstract model is true in the concrete model, but not vice versa) and *strong preservation* (same properties are true in both the abstract and concrete models). Methods for obtaining abstract models vary based on the amount of user-guidance required; some systems require the user to provide the abstract models, while other systems only require the user to provide abstraction functions.

Throughout these notes, C and A will denote concrete and abstract models, respectively, where a model is assumed to be some variant of state transition system (consisting of a set of states (S), a transition relation (R), and possibly a set of initial states (I) and a labelling function (L) from states to the propositions true in each state). In describing the various theories, we will assume that C and A are given, leaving aside issues of how each model was obtained.

2.1 Relating Abstract and Concrete Models

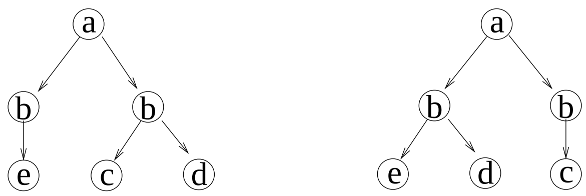
In general, when we say that A is an abstraction of C, we mean that the observable behaviors of A contain the observable behaviors of C relative to some *abstraction mapping* that converts behaviors of C into behaviors of A. Abstraction mappings can induce several types of abstractions [19], including:

- **Data abstractions:** Those that change the representation of data between models.
- **Behavioral abstractions:** Those that suppress behaviors between models, for example by hiding certain variables or certain modes of operation.
- **Temporal abstractions:** Those that change the time scale between models.

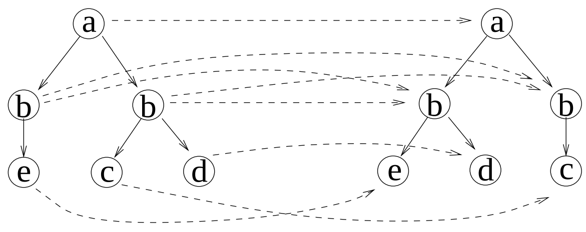
The most intuitive abstraction mappings are given as functions from the variables/data domains of the concrete model to the variables/data domains of the abstract model.[1] Such abstraction functions relate concrete and abstract models through *homomorphisms*. Given $C = \langle S_C, R_C \rangle$, $A = \langle S_A, R_A \rangle$, and $h : S_C \to S_A$, h is a homomorphism from C to A iff

$$\forall s, s' \in S_C \quad R_C(s, s') \to R_A(h(s), h(s'))$$

In the context of comparing observable behaviors, however, abstraction functions (as opposed to relations) can be overly restrictive. Consider the following pair of transition systems:



It is easy to see that the observable behaviors of these two systems are the same. However, no homomorphism exists that maps one to the other. If the abstraction mapping were relaxed to be a relation, however, the following mapping would suffice to align the observable behaviors:



It could be argued that this example does not look much like an abstraction, since the two systems differ only structurally. The example could have easily involved a change in data representation between the two systems to make it appear more "abstraction-like"; however, only the structural differences were shown for sake of clarity.

[1] If we take the view that a state in a transition system is defined by the values of the system's variables, such a mapping induces a mapping from concrete states to abstract states; this view is taken throughout these notes.

Relaxing abstraction mappings to allow relations as well as functions yields the class of relationships known as *simulations*. Formally, given $C = \langle S_C, R_C \rangle$, $A = \langle S_A, R_A \rangle$, and $h \subseteq S_C \times S_A$, h is a simulation relation from C to A iff

$$\forall s, s' \in S_C \; \forall t \in S_A \;\; R_C(s, s') \wedge h(s, t) \rightarrow \exists t' \in S_A \;\; R_A(t, t') \wedge h(s', t')$$

The mapping given in the above example meets this definition of a simulation relation. That h is allowed to be a relation is evident from the use of the existential quantifier in the consequent of the implication.

Both homomorphisms and simulations have been presented such that a mapping is only provided from the concrete model to the abstract model. In order to get from the abstract model back to the concrete model, the implicit assumption is that we would simply use the inverse relation of the given mapping. While this is reasonable, it is not necessary. Valid abstractions can be defined via two mappings, one from the concrete to the abstract and another from the abstract to the concrete, that are not inverses of one another. Such pairs of mappings are captured using *Galois connections*.

Using the framework of Galois connections requires that we have partial orderings on the sets of concrete and abstract states. Partial orderings on the abstract states can be motivated by their relative degrees of precision. For example, if the concrete states correspond to the integers, we could have abstract states leq_5 and leq_6, capturing numbers less than or equal to five and six, respectively. We think of leq_5 as more precise than leq_6 because it represents a smaller set of concrete values. Such notions of precision give rise to partial orders on the abstract states.

Motivating the ordering on concrete states is more difficult; Dams discusses several motivations in his thesis [8]. Basically, we can consider this ordering to be on *sets* of concrete states, rather than on the concrete states themselves. Under this interpretation, one set of states would be more precise than another if the former gives a more accurate description of those concrete states mapping to the same abstract state. This is not the only consideration we could take in ordering concrete states; it is, however, the main consideration that will be useful for purposes of defining abstractions.

Given the partial orderings, we can now define Galois connections. Let $\sqsubseteq$ and $\preceq$ be the partial order relations on S_C and S_A, respectively. Let $\alpha : S_C \rightarrow S_A$ and $\gamma : S_A \rightarrow S_C$. (α, γ) is a Galois connection from S_C to S_A iff for all $s_c \in S_C$ and $s_a \in S_A$, $\alpha(s_c) \preceq s_a \Leftrightarrow s_c \sqsubseteq \gamma(s_a)$. It follows from this definition that α and γ respect the relationships $\alpha \circ \gamma \circ \alpha = \alpha$ and $\gamma \circ \alpha \circ \gamma = \gamma$. This definition does not require α and γ to be inverses of one another; when they are, the pair (α, γ) is called a *Galois insertion*. Further details about the nature of Galois connections and insertions are beyond the scope of this writeup. However, it should be noted that using Galois insertions allows us to define relative precision between abstractions; this motivates their use in [7, 8].

3 Existing Abstraction Frameworks

This section summarizes several proposed abstraction frameworks. Each review provides a description of the framework and a brief summary detailing the instantiations of the above-listed parameters and the amount of user-guidance required by the framework.

3.1 Kurshan

Kurshan did some of the pioneering work in abstractions for model checking [17]. In his framework, both transition systems and properties are modelled as types of automata. Verification is performed by checking containment of the language of a system (S) in the language of a property (P):

$$\mathcal{L}(S) \subseteq \mathcal{L}(P)$$

This containment test is actually performed by an emptiness test on the intersection of the system language and the complement of the property language:

$$\mathcal{L}(S) \cap \overline{\mathcal{L}(P)} = \emptyset$$

Kurshan uses homomorphisms as the basis for abstractions. Given a concrete model, an abstract model, and a mapping from concrete to abstract states, his algorithms check whether the mapping yields a homomorphism between the two models. Once the homomorphism is verified, properties can be abstracted under the same homomorphism and verified against the abstract model. He gives results for both weak and strong preservation, as follows. Let C and A be concrete and abstract systems, respectively. Let P_C and P_A be statements of a property in the concrete and abstract systems, respectively. Let S' denote the complement of an automaton S. Assume P_A is deterministic and let h be a homomorphism. Kurshan's preservation theorem states that:

1. If $h(\mathcal{L}(P)) \subset \mathcal{L}(A)$, $h(\mathcal{L}(P'_C)) \subset \mathcal{L}(P'_A)$, then $\mathcal{L}(A) \subset \mathcal{L}(P_A) \Rightarrow \mathcal{L}(P) \subset \mathcal{L}(P_C)$.
2. If $h(\mathcal{L}(P)) = \mathcal{L}(A)$, $h(\mathcal{L}(P'_C)) = \mathcal{L}(P'_A)$, then $\mathcal{L}(A) \subset \mathcal{L}(P_A) \Leftrightarrow \mathcal{L}(P) \subset \mathcal{L}(P_C)$.[2]

The need for a user to supply both models plus the suggested homomorphism appears cumbersome. It is worth bearing in mind that Kurshan's general design methodology is top-down, so he assumes that both models have been produced as a natural part of the design process. It is also necessary that the user supply both models because Kurshan's framework allows strong fairness constraints to be expressed (see Section 3.8 for a brief discussion of the problems of abstraction in the context of strong fairness). In practice, having the user provide the abstract model also the advantage that the user is familiar with the abstract model and thus more likely to understand how it will behave.

The major drawback to Kurshan's approach is that the homomorphism checking must explore the state space of the concrete model. Abstractions are often used in verification because the original concrete models are too large to explore tractably. In such cases, the applicability of Kurshan's approach is limited. However, he does also prove results about compositional reasoning in relation to abstraction. Namely, if the concrete model is the product of several smaller models, then these smaller models may be abstracted individually under a single homomorphism; the cross-product of the abstracted components yields a valid abstraction of the original product. Formally, let $C = C_1 \times C_2 \times \ldots \times C_n$ and h be a homomorphism. Then $A = h(C_1) \times h(C_2) \times \ldots \times h(C_n)$ is a valid abstraction of C.

Bottom Line

This framework is worth considering if strong fairness is desired. If weak fairness is sufficient, there are several other frameworks that do not require the user to supply the abstract model or involve an exhaustive search of the concrete state space; these frameworks will likely be more practical.

[2] Actually, this is a somewhat stronger statement than that made in [17]; the precise theorem statement would require substantial background in Kurshan's theory which does not seem necessary for the current level of presentation.

3.2 Clarke, Grumberg, and Long

Although also based on homomorphisms, the framework of Clarke, Grumberg, and Long [5] differs from Kurshan's in two key ways: they are interested in branching time temporal logic and they attempt to generate abstract models automatically.

This framework explicitly includes initial states in transition systems. For this section then, we take $C = \langle S_C, I_C, R_C \rangle$ and $A = \langle S_A, I_A, R_A \rangle$ as our concrete and abstract models. Given a mapping function $h : S_C \to S_A$, they define A *approximates* C to be true when the following two conditions are met:

1. $\exists s_c (h(s_c) = s_a \wedge I_C(s_c))$ implies $I_A(s_a)$
2. $\exists s_{c_1} \exists s_{c_2} (h(s_{c_1}) = s_{a_1} \wedge h(s_{c_2}) = s_{a_2} \wedge R_C(s_{c_1}, s_{c_2}))$ implies $R_A(s_{a_1}, s_{a_2})$

This definition is equivalent to that for homomorphism; it has been presented here in its original format for consistency with the presentation of the rest of their framework. Whenever A approximates C, formulas from $\forall$CTL* are weakly preserved under abstraction.

Given a homomorphism, there is an abstract transition system that is minimal with respect to the above definition. This transition system is denoted $A_{\min}$, and is defined by replacing the implications in the above definition with bi-implications:

1. $\exists s_c (h(s_c) = s_a \wedge I_C(s_c))$ iff $I_{\min}(s_a)$
2. $\exists s_{c_1} \exists s_{c_2} (h(s_{c_1}) = s_{a_1} \wedge h(s_{c_2}) = s_{a_2} \wedge R_C(s_{c_1}, s_{c_2}))$ iff $R_{\min}(s_{a_1}, s_{a_2})$

Unfortunately, computing $A_{\min}$ requires first computing the full transition system. As an alternative, the authors propose a method for deriving an approximation to $A_{\min}$ from syntactic analysis of the text describing the model. First, they extract relational predicate expressions $\mathcal{I}$ and $\mathcal{R}$ from the program text to describe the concrete initial states and transition relation, respectively.

Constructing $\mathcal{I}$ and $\mathcal{R}$ requires that we have predicates, defined at the concrete level, corresponding to each operator in the language used to describe models. These predicates are assumed to be provided, either by the user or built-in to the system. Given any predicate ϕ over concrete values, the corresponding predicate $[\phi]$ in the abstract model can be computed via the equation

$$[\phi](\widehat{x_1}, \ldots, \widehat{x_n}) = \exists x_1, \ldots x_n (h(x_1) = \widehat{x_1} \wedge \cdots \wedge h(x_n) = \widehat{x_n} \wedge \phi(x_1, \ldots, x_n))$$

where h is the homomorphism and the hatted variables range over the respective abstract domains. $A_{\min}$ could be computed by evaluating $[\mathcal{I}]$ and $[\mathcal{R}]$. However, existential quantification is an expensive operation. Therefore, the authors propose distributing these quantifications over the subexpressions comprising the predicates being evaluated. As distributing existential quantification over conjunction weakens the expression, this distribution yields an approximate, rather than the exact, minimal abstract model. Models derived in this way approximate the original model, and hence weakly preserve $\forall$CTL* formulas.

For strong preservation of CTL* formulas, a notion of *exact* approximation is introduced. Exact approximations exist when the primitive relations in the relational expressions $\mathcal{I}$ and $\mathcal{R}$ are *congruent* under the homomorphisms. Homomorphisms induce equivalence relations on concrete values: if two values map to the same abstract value, then they are in the same equivalence class. Denoting congruence by $\sim$, the following equation defines when an expression ϕ is congruent under a homomorphism:

$$\forall d_1, \ldots, d_m, \forall e_1 \ldots e_m (\bigwedge_j d_j \sim e_j \to (\phi(d_1, \ldots, d_m) \Leftrightarrow \phi(e_1, \ldots, e_m)))$$

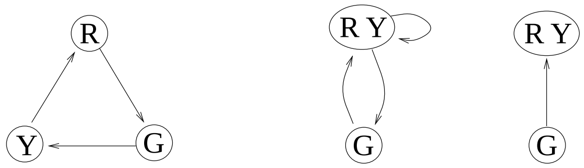
Bottom Line

Requiring the user to find the abstraction is not ideal. However, given such a mapping, the remainder of the ideas in this paper seem straightforward to implement. This is deemed a good starting point for implementations of abstraction techniques.

3.3 Dams, Gerth, and Grumberg

Clarke, Grumberg, and Long focused on weak preservation of $\forall$CTL*. Dams, Gerth, and Grumberg investigated frameworks for weak preservation of full CTL* formulas [7]; this work was presented in full detail and expanded upon in Dams' thesis [8]. This work follows the spirit of [5] in providing a method for producing approximate abstract models by syntactic analysis of the program text.

The main difference between this approach and that of [5] lies in the structure of the abstract transition relations. Dams, Gerth, and Grumberg use abstract transition relations with two transition relations: R_A^F and R_A^C, where F and C stand for "free" and "constrained", respectively. These pairs of transition relations are demonstrated via the following example. The transition system for a simple traffic light appears in the leftmost diagram, with the transition relations for R_A^F and R_A^C to its right, under the abstraction that maps the "R" and "Y" states to a single abstract state:



Intuitively, R_A^F corresponds to the type of transition relation generated in other approaches to abstraction: an edge exists from abstract state a to abstract state a' if there exists concrete state s mapping to a and concrete state s' mapping to a' such that there is a transition from s to s' in the original transition relation. Such transition relations preserve universal properties, but do not preserve existential properties because edges are introduced that suggest edges not present in the concrete model. For example, property $\mathsf{EF}(Y \wedge \mathsf{EX}G)$ (a yellow light can be immediately followed by a green light) is true of the abstract system, but not of the original system. R_A^C is designed to capture the transition relation with the goal of verifying existential properties: an edge exists from abstract state a to abstract state a' if for *all* concrete states s mapping to a there exists a concrete state s' mapping to a' such that there is a transition from s to s' in the original transition relation. The semantics of CTL* relative to these mixed transition systems uses R_A^F when the main operator is a universal path quantifier, and R_A^C when the main operator is an existential path quantifier. That these systems are weakly preserving instead of strongly preserving is seen by considering formula $\mathsf{EF}(R \wedge \mathsf{EX}G)$, which is true of the concrete model but is not true of the abstract model.

The paper also discusses issues such as optimality and relative precision of abstract models. Techniques are discussed for deriving abstract models from programs, in fashion similar to that discussed in [5]. The paper contains a few observations that may be useful for deriving abstractions directly from the program text if abstraction relative to existential property preservation is desired.

Bottom Line

The extensions to existential properties are unnecessary at the present time. The program-based abstractions used in this paper are much less clear than those used in the Clarke, Grumberg, and Long paper, and thereby would be more difficult to implement.

3.4 Cleaveland, Iyer, and Yankelevich

This work [6] bears many similarities to that of Dams, Gerth, and Grumberg. Abstract systems containing two transition relations are defined for strong preservation of CTL* formulas. The main difference between these two works appears to be that this paper uses homomorphisms to connect abstract and concrete transition systems, as opposed to Galois connections or insertions. No new insights are presented on how to actually locate good homomorphisms.

Bottom Line

As this work does not address the issue of computing abstract models, it will be of little use for implementing an abstraction framework.

3.5 Kelb, Dams, and Gerth

Kelb, Dams, and Gerth proposed a theory of compositional abstractions for symbolic mu-calculus model checking for parallel processes [16]. This work departs from other abstraction frameworks in that it does not propose using abstract data domains. Rather, it suggests that merely abstracting away details concerning the connections between different processes provides substantial reduction in the BDDs for the transition relation. This approach is inspired by the observation that process interconnection, not the individual process transition relations, is a significant cause of BDD explosion for the global transition relation.

An abstraction in this framework consists of a subset of the states local to some process. No indication is given as to how such subsets should be selected, or whether this selection is done manually or automatically. From this subset, two abstract versions of the transition relation are defined, one for interpreting existential properties and one for interpreting universal properties. The abstract model arising from these two transition relations weakly preserves all mu-calculus formulas.

The authors note that a naïve computation of the abstract transition relation for existential properties rarely yields adequate state space reduction. Accordingly, they propose a less naïve computation that restricts the existential abstract transition relation only at those points where abstraction has occurred; at points where no abstraction has occurred, a variant of the universal abstract transition relation is used. This version of the existential transition relation reportedly succeeded in verifying their example design whereas the naïve relation had failed.

Bottom Line

The overall idea of simply abstracting component connections may be worth keeping in mind. However, the work presented in this paper is too premature to recommend for implementation at this time. First, the paper is sloppy: there appear to be several inconsistencies in the definitions as stated. Next, although the paper reports substantial reductions with this technique on its single worked example, it is sketchy on details as to how the reductions were achieved; some comments, however, suggest that for safety properties, their abstractions amounted to pruning. Furthermore, there is no discussion regarding the size of the intermediate BDDs needed to compute the abstract transition relations.

3.6 Dams

This section discusses one chapter from Dams' thesis that was not covered by [7]. In this chapter (chapter 5), Dams turns to the issue of strong preservation of CTL* properties. All previously discussed works on strong preservation of properties required a strict relationship, such as bisimulation, between the abstract and concrete models. Dams explores another technique for producing strongly preserving abstract models relative to arbitrary sets of CTL* formulas.

This technique, which he calls *partition refinement*, is similar in spirit to deterministic finite automaton minimization based on bisimulation equivalence. In essence, for each pair of states we check whether there exists some formula of interest that is true in one state but not true in the other. If such a formula exists, the two states must lie in different equivalence classes under strong preservation. Once all pairs of states are considered, each equivalence class generates a state in the abstract model. Naturally, for sake of efficiency, the proposed algorithms do not consider each pair of states individually, despite achieving a similar end result.

Given a concrete model C and a set of formulas L to preserve, the partition refinement approach consists of the following steps:

1. Compute the *companion* of L, a set of formulas such that consistency of the abstract states with respect to $comp(L)$ guarantees the strong preservation of L. Consistency is defined as follows:

   > Let A be a set of abstract states and $\rho \subseteq C \times A$ a relation describing the connections between the abstract and concrete states. ρ is consistent with L in A iff for every $\phi \in L$ and every $a \in A$, $\forall c \in C\ [\rho(c,a) \Rightarrow c \models \phi]$ or $\forall c \in C\ [\rho(c,a) \Rightarrow c \not\models \phi]$.

2. Start with the single abstract state *true* and assume that all concrete states map to it. Derive the abstract transition system on the existing abstract state from the definition of the concrete model.

3. Iteratively choose some element ϕ from $comp(L)$, split each existing abstract state into those states for which ϕ is true and those for which ϕ is false. Re-derive the abstract transition system on the new set of abstract states.

Several termination conditions are possible for this algorithm. First, once all elements of $comp(L)$ have been used for splitting, the algorithm can terminate. As $comp(L)$ may be of infinite size, this method is not sufficient in general. Secondly, the abstract model computed during each iteration weakly preserves L; if the desired property verifies on an intermediate model, the algorithm may terminate. Finally, depending upon the nature of L a suitable stabilization condition may be found that indicates when to stop iterating (failure to split on one iteration is not sufficient, as the system may remain stable for splits on some formulas). For example, if $L = \forall$CTL, the algorithm can iteratively compute the simulation relation between the states of the concrete and abstract models, terminating when a single iteration does not produce any changes in the simulation relation. Stability conditions are also discussed for an arbitrary $\forall$CTL formula. Other subclasses of CTL* are not discussed explicitly, though the results are expected to extend to any such subclass, including individual formulas.

Bottom Line

The proposed algorithms appear to require extensive amounts of computation. Computing the intermediate abstract models after splitting requires computing the sets of concrete states for which the split-upon formula holds. This is essentially equivalent to performing repeated model-checks over the concrete transition relation. Even though methods are given for reducing the formulas to be split upon to some containing only disjunction and AX operators, the overhead still appears prohibitive. Dams has yet to implement these algorithms, so no experimental results are available to counteract these concerns.

3.7 VIS

The VIS system [3] from UC Berkeley supports a limited form of abstraction. Users can supply a file containing names of variables to be abstracted. The abstracted variables are then cut off from the variables that drive them in the underlying system so that they range freely over their ranges. There is no apparent facility for defining more abstract ranges for variables as seen in other frameworks.

Bottom Line

The utility of this abstraction mechanism is unclear. Mechanisms that allow a shift in variable ranges will likely be more useful for verification, as they provide more structured state space reductions.

3.8 Nitsche and Wolper

Nitsche and Wolper consider behavioral abstraction for LTL under fairness assumptions [20]. The abstraction mechanisms discussed so far are applicable in the context of weak fairness ("something happens infinitely often"), but not in the context of strong fairness ("infinitely often enabled implies infinitely often performed"). The problem is that, under strong fairness conditions, an abstraction may result in fewer, rather than more, behaviors relative to the concrete system. In this situation, preservation of formulas does not hold from the abstract to the concrete system.

This paper is a first attempt at answering the question: "when does truth under fairness of a property in the abstract system imply truth under fairness of a property in the concrete system?" Nitsche and Wolper show that *relative liveness* properties, a notion of properties based on language prefixes, are preserved under abstraction. Other papers give further details on relative liveness properties and their relationship to the traditional notion of properties [13].

3.9 Ip and Dill

Ip and Dill investigate another form of abstraction based on *reversible rules* [15]. Reversible rules are transitions such that the source state of the transition can be reconstructed from the transition and the target state of the transition. States that are not targets of reversible rules are called *progenitors*. The progenitors form the states of the abstract model, with edges between progenitors reflecting the existence of paths between them in the original model.

Ip and Dill claim that reversible rules can be identified from a program's text using traditional program analysis techniques (though they provide no details as to how this is done). As a result, this form of abstraction is completely automatic, unlike those forms that require the user to supply some sort of abstraction mapping. Deadlock checking, invariant checking, and stuttering-invariant CTL model checking are preserved under this abstraction.

No analysis is provided that compares this technique to other abstraction techniques. Ip and Dill do study the reductions in state space resulting from reversible rules, symmetry, and a combination of the two in the context of explicit state model checking for cache coherence protocols. Reversible rules give reductions ranging from 30 to 50 percent on their examples, which is much less substantial than those gained from symmetry alone on each example. The combination of symmetry and reversible rules yields substantial state space reductions (in the area of 90 percent or better on their examples).

Bottom Line

Abstractions based on reversible rules capture different types of reductions than the more traditional types of abstractions. The experimental results are insufficient to indicate whether reversible rules based abstractions are useful on a wide range of examples though. There is also no discussion of how to apply these results in a BDD framework. In short, this work is too premature to be recommended for adoption at this time.

3.10 Lee, Pardo, Jang, Hachtel, and Somenzi

This paper [18] considers abstractions for CTL model checking that are extremely similar to those of Kelb, Dams, and Gerth (Section 3.5). Abstractions do not involve changes in domains of values for variables; they involve only the tearing of connections between components of the system to be verified. This work differs from that of Kelb, Dams, and Gerth in providing a method for refining abstractions that are too coarse. This refinement method consists essentially of re-connecting variables that were torn by an earlier abstraction.

Bottom Line

Similarly to Kelb, Dams, and Gerth, this notion of tearing connections may be useful in practice, but the results are too preliminary to suggest wholeheartedly. It would seem that more useful abstractions could be obtained by reductions in variable domains, particularly for datapath intensive circuits.

3.11 Pardo and Hachtel

Pardo and Hachtel have considered abstraction techniques mu-calculus symbolic model checking [21]. As in the previous paper involving these authors, the focus is on automatic refinement of abstractions that have shown to be too coarse. Abstractions are based not on changing the domains of variables, but on attempts to reduce the size of the BDDs required to represent the state space.

Abstraction is performed by approximating the sets of states that satisfy the subformulas of the property to be checked. This work is unique in that it handles both existential and universal properties without the use of separate transition relations. Under- or over-approximations are taken for each subformula depending upon the number of negations in which the subformula is nested. The taking of approximations is guided by a user-defined limit on BDD-size. The algorithms attempt to calculate the full sets of states satisfying subformulas until the limits are reached, at which point they stop with the current approximations. Refinements are based on determining sets of states to either add or remove from the current approximations in order to improve the accuracy of the abstraction.

Bottom Line

At present, these techniques are extremely ad-hoc; they appear to amount to a collection of unguided hacks for trying to reduce BDD sizes. "Abstraction" is guided purely by when limits on BDD sizes are reached, rather than on any information about the underlying system being verified. In addition, these ideas have not yet been implemented, so there is no experimental analysis to support these techniques. Furthermore, the paper omits definitions of several functions used to compute refinements, so the precise nature of the refinement methodology is somewhat unclear.

3.12 Graf and Loiseaux

Graf and Loiseaux provide one of the only available papers describing the implementation of an abstraction system [11]. Their tool works over boolean programs of guarded commands. Abstractions are given as boolean expressions on abstract and concrete program variables; these expressions represent an abstraction relation between concrete and abstract states. The abstraction framework used here is based primarily on one proposed in [2]; this framework is a Galois-connection framework supposedly quite similar to that of [7].[3]

Bottom Line

The theoretical framework on which this tool is based does not provide any new insights over other works discussed in this writeup. Unfortunately, this paper does not contain enough information to be useful for understanding how to implement abstractions either. The description of how abstractions are stated is rather vague; even worse, the given example fails to provide the abstractions used to verify it, either informally or as part of the given code fragment.

4 What is a Symmetry Framework?

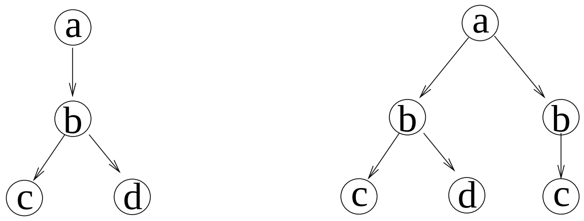
Recall that, in general, abstractions are indicated by simulation relations between models. Simulation was defined between two transition systems $M_1 = \langle S_1, R_1 \rangle$ and $M_2 = \langle S_2, R_2 \rangle$ as a relation $h \subseteq S_1 \times S_2$ satisfying the equation

$$\forall s, s' \in S_1 \; \forall t \in S_2 \;\; R_1(s, s') \wedge h(s, t) \rightarrow \exists t' \in S_2 \;\; R_2(t, t') \wedge h(s', t')$$

If, in addition, we require h to satisfy the equation

$$\forall s \in S_1 \; \forall t, t' \in S_2 \;\; R_2(t, t') \wedge h(s, t) \rightarrow \exists s' \in S_1 \;\; R_1(s, s') \wedge h(s', t')$$

then h is called a *bisimulation*. It is worth noting that bisimulation is a stronger condition than having a simulation from M_1 to M_2 and a simulation from M_2 to M_1, as demonstrated by the following pair of transition systems.



[3] This paper is not reviewed here because we were unable to locate a copy.

Symmetry exists between bisimilar parts of a transition system. Symmetry-based reductions could therefore be performed by computing bisimulation relations on the state space. This, however, would require representing the original transition relation in order to check the bisimulations. Several researchers have investigated alternative methods. All of these methods are based on some common mathematical notions from group theory.

- A *permutation* is a bijective mapping from a set to itself. For symmetry reductions, the mappings are on the states of a transition system.
- A permutation σ is a *symmetry* iff it preserves the transition relation, *i.e.*:

$$\forall s, s'(s, s') \in R \Rightarrow (\sigma(s), \sigma(s')) \in R$$

- A set of permutations G is a *symmetry group* iff every permutation in it is a symmetry, and it is closed under composition and inverse.
- Given a group G of permutations and a state s, the *orbit* of s, denoted $\theta(s)$, is defined as $\{s' \mid \exists \sigma \in G : \sigma(s) = s'\}$. Orbits define equivalence classes under permutation.

Intuitively, we can work with symmetries by computing the orbits, selecting a unique representative state from each orbit, computing the transition relation relative to those representatives, and then model-checking on this reduced model. The naïve algorithms for these tasks are straightforward. To compute the orbit of a state, iteratively apply the permutations until a fixpoint is reached. To model check, whenever a new state is reached, compute its representative, adding it to the list of nodes to be expanded if it is not already in the reachable state space. The existing symmetry frameworks are based upon these ideas and vary along the following parameters:

- How do we find the initial permutations?
- How do we select the representatives?
- How do we construct the mapping from states to representatives?
- BDD or explicit-state model checking over the representatives?

5 Existing Symmetry Frameworks

5.1 Clarke, Filkorn, Enders, and Jha

This paper presents the somewhat classical theory of symmetry based on calculating orbit relations using BDDs [4]. User-supplied permutations are used to generate a BDD for the *orbit relation*, which indicates for each pair of states whether or not they lie in the same orbit under the provided permutations. A new transition structure is generated from this relation, where the orbits provide the sets of states, the transition relation is lifted to the orbits in the obvious way, and the labelling is provided by the labelling of the representative states. Full CTL* is preserved in the symmetry-reduced model, provided the propositions true of each representative state are precisely those propositions true in each state in its orbit.

This paper also proves some complexity results. First, the orbit problem is shown to be as hard as graph isomorphism. Second, the authors prove an exponential lower bound (in the number of components or number of state variables in each component) on the size of the BDD needed to represent the orbit relation for *transitive* permutation groups. Let S be a set of size n. A permutation group G is transitive for S iff there exists an element $i \in S$ such that for all elements $j \in S$, there exists a permutation $\omega \in G$ such that $\omega(i) = j$. Examples of transitive groups include rotation groups as well as the full symmetric group.

Bottom Line

The theory presented in this paper is very nice. However, the exponential lower bounds for several BDDs crucial for symmetry reduction suggest that straightforward symmetry computations and BDDs are not a workable combination for intractable designs.

5.2 Ip and Dill

Ip and Dill are interested in explicit-state model checking [14]. They have augmented a model-description language with a data type through which users can identify potential symmetries. They describe syntactic checks that guarantee the soundness of these user-indicated symmetries and discuss how their techniques may be used for on-the-fly verification.

Bottom Line

This work applies only to verification of safety properties; liveness properties are not supported by this framework.

5.3 McMillan and Aziz

McMillan and Aziz extend symmetry results to netlist representations of designs [1] and further investigate lower bounds on BDD size and computations of representative functions. Regarding BDD size, they prove that the size of the BDD for the full symmetry group on $\{1, \ldots, n\}$ is exponential in n. This motivates their focus on explicit-state, rather than symbolic, model-checking.

In McMillan and Aziz's methodology, the user provides permutations as pairs (σ, τ), where σ is a permutation on the latch variables and τ a permutation on the input variables in a netlist. Such a permutation is *sound* iff it satisfies the equation

$$\sigma(\vec{F}(\vec{x}, \vec{u})) = \vec{F}(\sigma(\vec{x}), \tau(\vec{u}))$$

where $\vec{F}$ is the vector of next-state functions for the latches, $\vec{x}$ provides the latch variables, and $\vec{u}$ provides the input variables. This equation can be tested in BDDs for each permutation provided. After the permutations are checked, the reduced model is computed using the straightforward algorithm for using symmetry with explicit-state model checking.

McMillan and Aziz also address the issue of computing the canonicalization function. It follows from the NP-completeness of the orbit problem that canonicalization is likely also not polynomial time solvable (if it were, we could solve the orbit problem in polynomial time by canonicalizing two states and checking the equivalence of their representatives). McMillan and Aziz are interested in finding good heuristics for this computation.

First, they identify three cases in which canonicalization is computable in polynomial time:

1. The permutation group has a single generator.

2. All the permutations are transpositions (a transposition switches two elements, leaving the rest unchanged).

3. Assume we have a system with n identical components, each with m local state bits. Assume also that there is symmetry in two levels in the system, *i.e.* symmetry both within the individual components and between the components. Canonicalization is polynomial-time computable if we can define a set of permutations, each of which acts on the local variables of only one component, plus a set of permutations capturing the global symmetry that preserves the relative ordering of the state bits within each component.

Next, they describe a heuristic procedure for computing the canonicalization function that may result in multiple representatives. In the naïve algorithm for computing orbits, the set of permutations is iteratively applied to the current set of orbit elements until a fixpoint is reached. McMillan and Aziz propose a greedy algorithm that only keeps the lexographically largest state after each iteration. As an alternative, they also consider an algorithm that searches for the lexographically largest state after two iterations, but performs the next iteration on the state that reaches this largest state (look-2/step-1). Based on their experimental results, they conclude that the simpler, look-1 algorithm performs almost as well as look-2/step1 in much less time.

Bottom Line

This paper focuses on explicit-state methods for symmetry. In addition, a proposed method for checking the soundness of permutations using BDDs can potentially suffer exponential blow-up as it involves variable re-ordering. The method seemed better suited to situations where the properties were suspected to be correct, rather than to exploratory model checking.

5.4 Gyuris and Sistla

The paper presents an on-the-fly, explicit-state model checking algorithm that checks for correctness under weak fairness and exploits symmetry [12]; it is an extension of theoretical work by Emerson and Sistla [10]. This is the only known work on symmetry that considers fairness constraints. An implementation of a system based on the framework exists and experimental results are given comparing this framework to that proposed in [10].

Bottom Line

The main drawback to this work is that it only supports user-defined symmetries at this time. These symmetries identify reductions due to duplicated components, but miss several other types of more subtle symmetries that arise in practice. It is unclear what percentage of symmetries are missed with the restriction to user-defined symmetries, but this limitation is worth keeping in mind.

5.5 Emerson, Jha, and Peled

Emerson, Jha, and Peled consider combining partial order methods and symmetry [9]. The combination seems reasonable to explore, as the techniques are non-overlapping and their respective benefits have been substantiated with experimental evidence. The paper provides a fairly straightforward theory for combining the two techniques. It also provides algorithms for explicit state model checking using the combined techniques, one for preserving LTL-X (LTL without the "next" operator), and one for preserving CTL*-X (CTL* without the "next" operator). Preservation with the "next" operator is not considered as such preservation does not occur with partial order methods in general.

There has been no implementation of these ideas to date, and thus the paper provides no experimental evaluation of the reductions achieved.

Bottom Line

Partial order methods have proven successful for designs with an interleaving semantics. If such a semantics is part of the verification framework and partial order methods are considered, these algorithms might be worth considering.

References

[1] Adnan Aziz and Ken McMillan. Heuristic symmetry reduction for invariant verification. Submitted to DAC 97.

[2] A. Bouajjani, S. Bensalem, C. Loiseaux, and J. Sifakis. Property preserving simulations. In *Proc. of International Conference on Computer-Aided Verification (CAV)*, 1992.

[3] R.K. Brayton et al. VIS: A system for verification and synthesis. Technical Report UCB/ERL M95, Electronic Research Lab, University of California at Berkeley, 1995.

[4] E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, August 1996.

[5] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[6] Rance Cleaveland, Purush Iyer, and Daniel Yankelevich. Optimality in abstractions of model checking. In Alan Mycroft, editor, *Static Analysis*, pages 51–63. Springer-Verlag, 1995. Lecture Notes in Computer Science, number 983.

[7] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems. to appear in ACM-TOPLAS.

[8] Dennis Dams. *Abstract Interpretation and Partition Refinement for Model Checking.* PhD thesis, Technische Universiteit Eindhoven, 1996.

[9] E. Allen Emerson, Somesh Jha, and Doron Peled. Combining partial order and symmetry reductions. In *Proc. of TACAS, Tools and Algorithsm for the Construction and Analysis of Systems, LNCS 1217*, pages 19–34, 1997.

[10] E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, August 1996.

[11] Susanne Graf and Claire Loiseaux. A tool for symbolic program verification and abstraction. In *Workshop on Computer-Aided Verification, CAV93, Heraklion, Crete.* LNCS 697, Springer Verlag, 1993.

[12] Viktor Gyuris and A. Prasad Sistla. On-the-fly model checking under fairness that exploits symmetry. In *Proc. of International Conference on Computer-Aided Verification (CAV)*, pages 232–243, 1997.

[13] T. A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43:135–141, 1992.

[14] C. Norris Ip and David Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1/2):41–76, August 1996.

[15] C. Norris Ip and David L. Dill. State reduction using reversible rules. In *Proc. 33rd Design Automation Conference (DAC)*, 1996.

[16] Peter Kelb, Dennis Dams, and Rob Gerth. Efficient symbolic model checking of the full μ-calculus using compositional abstractions. Technical Report Computing Science Reports 95/31, Eindhoven University of Technology, The Netherlands, October 1995.

[17] Robert P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach.* Princeton University Press, 1994.

[18] Woohyuk Lee, Abelardo Pardo, Jae-Young Jang, Gary Hachtel, and Fabio Somenzi. Tearing based automatic abstraction for ctl model checking. In *Proc. IEEE International Conference on Computer-Aided Design*, pages 76–81, 1996.

[19] Thomas Frederick Melham. Formalizing abstraction mechanisms for hardware verification in higher order logic. Technical Report TR 201, University of Cambridge Computer Laboratory, August 1990.

[20] Ulrich Nitsche and Pierre Wolper. Relative liveness and behavior abstraction. In *Proc. Principles of Distributed Computing (PODC)*, 1997.

[21] Abelardo Pardo and Gary D. Hachtel. Automatic abstraction techniques for propositional μ-calculus model checking. In *Proc. of International Conference on Computer-Aided Verification (CAV)*, pages 12–23, 1997.