Structural Symmetry and Model Checking

Gurmeet Singh Manku[1] and Ramin Hojati[2] and Robert Brayton[3]

[1] IBM Almaden Research Center (manku@almaden.ibm.com)
[2] University of California at Berkeley and HDAC Inc. (hojati@hdac.com)
[3] University of California at Berkeley (brayton@ic.berkeley.edu)

Abstract. A fully automatic framework is presented for identifying symmetries in structural descriptions of digital circuits and CTL* formulas and using them in a model checker. The set of sub-formulas of a formula is partitioned into equivalence classes so that truth values for only one sub-formula in any class need be evaluated for model checking. Structural symmetries in net-list descriptions of digital circuits and CTL* formulas are formally defined and their relationship with the corresponding Kripke structures is described. A technique for automatic identification of structural symmetries is described that requires computation of the automorphism group of a suitable labeled directed graph. A novel fast algorithm for this problem is presented. Finally, experimental results are reported for BLIF-MV net-lists derived from Verilog.

1 Introduction

Temporal model checking algorithms [CES86,BCL+94] typically explore the states of a non-deterministic finite state machine that represents the system under scrutiny. A major bottleneck is the exponential number of states that need be explored. This is commonly known as *State Space Explosion*. Among the techniques being developed for countering this problem are partial order methods, abstraction, compositional approaches, and symmetry reductions. Symmetries abound in hardware circuits, distributed algorithms and concurrent programs.

Emerson and Sistla [ES96] and Clarke *et al* [CEFJ96] show how symmetries in Kripke structures and CTL* formulas allow the construction of a smaller sized *quotient* structure such that the formula need be verified only for the quotient. In both works, symmetries are specified by hand by the designer. Emerson and Sistla [ES95] have developed theory for using symmetries with fairness constraints. Gyuris and Sistla [GS97] have developed an on-the-fly model checker that utilizes symmetries under fairness. Emerson, Jha and Peled [EJP97] have combined partial orders and symmetries. Symmetries have also been shown to speedup transistor-level verification [PB97].

Ip and Dill [ID96] use symmetries for speeding up verification of safety properties using explicit techniques for designs specified in a guarded command language. They propose augmentation of the language itself by introducing a new data type with syntactic constraints for sets of fully symmetric variables called *scalarsets*. A major drawback of scalarsets is that important and standard specification languages such as Verilog and VHDL cannot be modified easily.

Our work is distinguished from previous work on several counts. First, we provide a framework for identifying symmetries *automatically*. Second, we formalize the notion

of *structural symmetries* in net-list descriptions, show how they relate to those in Kripke structures and present effective algorithms for automatically identifying them. Third, we show how *symmetries in the formula* itself can be used with or without quotient structures to expedite model checking.

2 Preliminaries

Kripke Structures: Let AP be a set of atomic propositions. A *Kripke structure* over AP is a triple $M = (S, R, K)$, where S is a finite set of *states*, $R \subseteq S \times S$ is a *transition relation* that is *total*, i.e. $(\forall s \in S)(\exists t \in S)((s,t) \in R)$, and $K : S \rightarrow 2^{AP}$ is a *labeling function*. Let states in S be encoded such that there is a 1-1 mapping from S into 2^L for some L. Then K is a multi-output boolean function $K : 2^L \rightarrow 2^{AP}$.

Temporal Logic CTL* is the set of strings $\mathcal{S}$ generated by the two productions $\mathcal{S} \rightarrow \langle AP \rangle \mid \neg\mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid E(\mathcal{P})$ and $\mathcal{P} \rightarrow \mathcal{S} \mid \neg\mathcal{P} \mid \mathcal{P} \vee \mathcal{P} \mid X\mathcal{P} \mid \mathcal{P}U\mathcal{P}$, where $\langle AP \rangle$ denotes any proposition $p \in AP$, $\mathcal{S}$ denotes a set of *state formulas*, and $\mathcal{P}$ denotes a set of *path formulas*. If $M = (S, R, K)$ is a Kripke structure, $(M, s \models f)$ denotes that the state formula f is true for state $s \in S$. Similarly, $(M, \psi \models g)$ denotes that path formula g is true for path ψ. See [CEFJ96] for a formal definition of $\models$ using this notation.

We say that two CTL* formulas are *logically equivalent* if their truth values are identical for every state in any Kripke structure. We say that two CTL* formulas are *structurally equivalent* if they also have isomorphic parse trees. Intuitively, the second formula is the same as the first one written in a structurally different way due to the commutativity of some operators.

Model Checking Problem: *Given a set of atomic propositions AP, a Kripke structure $M = (S, R, K)$, a CTL* formula f and a set of initial states $I \subseteq S$, does every state in I satisfy f?* Clarke, Emerson and Sistla [CES86] presented the first algorithm for CTL model checking using explicit state space exploration. A Binary Decision Diagrams based symbolic model checker that can handle more than 10^{120} states on some pipelined circuits has been described by Burch *et al* [BCL$^+$94].

Permutation Groups: A permutation π is a bijective mapping $\pi : S \rightarrow S$ defined over a finite non-empty set S. We denote the action of π on an element $s \in S$ by πs. We use $H \leq G$ to denote that H is a subgroup of G. We denote the intersection of G_1 and G_2 by $G_1 \cap G_2$, which itself is a group. For a set $T \subseteq S$, we define $\pi T = \{s \mid s = \pi t$ where $t \in T\}$. This overloads operator π but buys us notational convenience. For a set $X \subseteq S$, such that $\pi X = X$, we use $\pi_{<X>} : X \rightarrow X$ to denote the restriction of π to X.

Definition of $\bowtie$ Operator: Let G denote a permutation group over $S_1 \cup S_2$ such that $(\forall \pi \in G)((\pi S_1 = S_1) \wedge (\pi S_2 = S_2))$. Let H denote a permutation group over $S_2 \cup S_3$ similarly. Then $G \bowtie H$ is defined to be a permutation group over $S_1 \cup S_3$ such that $\pi \in G \bowtie H$ if and only if there exist $g \in G$ and $h \in H$ such that $(\forall s \in S_1)(gs = \pi s)$, $(\forall s \in S_3)(hs = \pi s)$ and $(\forall s \in S_2)(gs = hs)$.

3 Symmetric Sub-formulas

Let $M = (S, R, K)$ be a Kripke structure with 2^L states. Let $\pi : L \rightarrow L$ be a permutation. It induces a permutation $\Pi : 2^L \rightarrow 2^L$ naturally. Let π be such that Π is

an automorphism of the directed unlabeled graph (S, R). The set of all such π forms a group, which we denote by $Aut_M\ L$. Later, we consider a Kripke structure M having additional labels drawn from a set $X \supseteq AP$. The new labels can be looked upon as a mapping $K' : 2^L \rightarrow 2^X$. When $X = AP$, $K' = K$.

Consider a permutation $\pi : L \cup X \rightarrow L \cup X$ such that $(\pi L = L)$ and $(\pi_{<L>} \in Aut_M\ L)$ and $(\forall x \in 2^L)(\forall y \in 2^{AP})((K'(x) = y) \Leftrightarrow (K'(\pi x) = \pi y))$. The set of all such permutations π forms a group which we denote by $Aut_M\ L \cdot X$.

For $s \in S$ and $\pi \in Aut_M\ L \cdot AP$, let πs denote the state obtained by applying π to the encoding of s. For any path ψ in M, let $\pi\psi$ denote the path obtained by applying π to every state in ψ. For a CTL* formula f defined on AP, let πf denote the formula obtained by replacing every occurrence of $p \in AP$ by πp.

Theorem 1. *For a Kripke structure $M = (S, R, K)$ and a permutation $\pi \in Aut_M\ L \cdot AP$, $((M, s \models f) \Leftrightarrow (M, \pi s \models \pi f))$ and $((M, \psi \models g) \Leftrightarrow (M, \pi\psi \models \pi g))$ for any state $s \in S$, any path ψ in M, any CTL* state formula f and any CTL* path formula g.* □

Theorem 1 can be proved by induction using the identities $(\pi(\neg f) = \neg(\pi f))$, $(\pi(f \vee g) = \pi f \vee \pi g)$, $(\pi(Xf) = X(\pi f))$, $(\pi(Eg) = E(\pi g))$ and $(\pi(g_1 U g_2) = \pi g_1 U \pi g_2)$. A detailed proof can be found in [Man97]. For a Kripke structure M and CTL* formula f defined on AP, let SF denote the set of all sub-formulas of f, including any atomic propositions in AP that occur in f. Recall the definitions of logical and structural equivalence from Section 2. For a subgroup $G \leq Aut_M\ L \cdot AP$, we define a relation $\approx^G \subseteq SF \times SF$ as $(\forall f_1, f_2 \in SF)((f_1 \approx^G f_2) \Leftrightarrow (\exists \pi \in G)(\pi f_1$ and f_2 are logically equivalent)). We also define a relation $\approx^G_s$ the same way as $\approx^G$ but replacing logical equivalence by structural. The following theorem is immediate.

Theorem 2. *For $G \leq Aut_M\ L \cdot AP$, the relations $\approx^G$ and $\approx^G_s$ are equivalence relations, with $\approx^G$ inducing a partition coarser than that induced by $\approx^G_s$.* □

3.1 Applications

First, consider two sub-formulas g and h in the same equivalence class. Let $\pi \in Aut_M\ L \cdot AP$ be a witness that transforms h into g. If the truth value of h has been evaluated for all states in S, the truth value for g is immediately available. In a symbolic technique, the BDD for g can be computed from that for h by variable substitution corresponding to π. Second, having proved the correctness of a CTL* formula f, one can use Theorem 1 to generate new formulas whose truth value is already known by producing a non-trivial $\pi \in G$ and constructing πf. A model checker can present new formulas to a designer in a controlled fashion using an interactive user interface. Third, it will be clear that identification of symmetric sub-formulas contributed to savings on top of quotient structures that we describe in Section 4.

3.2 Computing Equivalence Classes

Given $G \leq Aut_M\ L \cdot AP$ and a CTL formula f, how do we find two sub-formulas g and h such that $g \approx^G h$?* This is a computationally hard problem even if f is a

simple boolean formula without path operators or temporal quantifiers [AT96]. If we replace $\approx^G$ by $\approx^G_s$, the problem is as hard as graph isomorphism [Man97]. We outline a technique that can identify symmetric sub-formulas if the symmetry in the specification is reflected in the formula as well, which is true in practice.

For a CTL* formula f, let SF denote the set of sub-formulas of f, including all atomic propositions that occur in f. Consider the group consisting of permutations $\pi : AP \to AP$ such that f and πf are structurally equivalent. Every permutation in this group implicitly defines a permutation on the set $AP \cup SF$. We denote this group by $Aut_f\ AP \cdot SF$. Let $G \leq Aut_M\ L \cdot AP$. Let $H \leq Aut_f\ AP \cdot SF$. Recall the definition of $\bowtie$ from Section 2. We see that the group $G \bowtie H$ is well defined. We define a relation $\approx^{G \bowtie H}_s \subseteq SF \times SF$ as $(\forall f_1, f_2 \in SF)((f_1 \approx^{G \bowtie H}_s f_2) \Leftrightarrow (\exists \pi \in G \bowtie H)(\pi f_1$ and f_2 are structurally equivalent)). This is an equivalence relation. In general, the partition induced by $\approx^{G \bowtie H}_s$ is finer than that induced by $\approx^G_s$ for $G = Aut_M\ L \cdot AP$.

In Section 5, we will show how $G \leq Aut_M\ L \cdot AP$, $H \leq Aut_f\ AP \cdot SF$ and $G \bowtie H$ can all be computed automatically. The representation for $G \bowtie H$ would allow us to easily identify the partitions induced by $\approx^{G \bowtie H}_s$ and produce witnesses that transform one sub-formula into another.

4 Quotient Structures

We now develop a theory of symmetries for Kripke structures, extending those developed by Clarke *et al* [CEFJ96] and Emerson and Sistla [ES96]. Let $M = (S, R, K)$ be a Kripke structure with 2^L states. Let $G \leq Aut_M\ L \cdot X$ for some set of labels $X \supseteq AP$. Let two states s and t in S be related if there exists $\pi \in G$ such that $\pi s = t$. This defines an equivalence relation, partitioning S into equivalent sets called *orbits*. We denote the orbit of a state $s \in S$ by $[\![s]\!]_G$. We pick a state from each orbit to obtain a set of representatives and define a function $\xi_G : S \to S$ such that each state is mapped to the representative of the orbit it belongs to. ξ_G is not unique. The results in this paper hold for any ξ_G. For a Kripke structure $M = (S, R, K)$ and $G \leq Aut_M\ L \cdot X$ for some set of labels X, the *quotient structure* is defined as $M_G = (S_G, R_G, K_G)$, where $S_G = \{[\![s]\!]_G \mid s \in S\}$, $R_G = \{([\![s]\!]_G, [\![t]\!]_G) \mid (s,t) \in R\}$ and $K_G([\![s]\!]_G) = K(\xi_G(s))$. The fundamental result in [CEFJ96] is captured by the following theorem:

Theorem 3. *[CEFJ96] For a Kripke structure $M = (S, R, K)$ and a group $G \leq Aut_M\ L \cdot AP$, if $(\forall \pi \in G)(\forall p \in AP)(\pi p = p)$, then for any CTL* formula f, it is true that $(\forall s \in S)((M, s \models f) \Leftrightarrow (M_G, [\![s]\!]_G \models f))$.* □

Application of Theorem 3 requires that the truth value of every atomic proposition be invariant under every permutation in G. In the extreme case, we could have $AP = L$, giving each state a unique label and making G trivial. Emerson and Sistla [ES96] present a generalization of Theorem 3. However, their theory is built for Kripke structures derived from systems of communicating isomorphic processes, the set of atomic propositions being the set of shared variables. In our terminology, it amounts to assuming $AP = L$ and a single initial state. We now develop a generalization of their result so that it is applicable to Kripke structures derived from net-list descriptions.

For a CTL* formula f, let MPS be the set of its maximal propositional sub-formulas. Let f_{MPS} be the multi-output boolean function $2^{AP} \rightarrow 2^{MPS}$. We define $Aut_f\ AP{\cdot}MPS = \{\pi : AP \cup MPS \rightarrow AP \cup MPS \mid \pi$ is a permutation, $\pi AP = AP$, $\pi MPS = MPS$, $(\forall y \in MPS)(\pi y = y)$ and $(\forall x \in 2^{AP})(f_{MPS}(x) = f_{MPS}(\pi x))\}$. This set forms a group. For $G \leq Aut_M\ L \cdot AP$ and $H \leq Aut_f\ AP \cdot MPS$, the group $G \bowtie H$ is well defined.

Theorem 4. *For $G \leq Aut_M\ L \cdot AP$ and $H \leq Aut_f\ AP \cdot MPS$, it is true that $(\forall s \in S)((M, s \models f) \Leftrightarrow (M_{G \bowtie H}, [\![s]\!]_{G \bowtie H} \models f))$*

Proof. The crux lies in showing that $G \bowtie H \leq Aut_M\ L \cdot X$, where $X = MPS$. Then replacing labels of M by labels corresponding to evaluations of sub-formulas in MPS allows a straightforward application of Theorem 3 to get the desired result. □

To construct the quotient, we require $G \bowtie H$, for which we present an automatic procedure in the next section. Once we have constructed $G \bowtie H$, how do we use it to expedite model checking? A detailed exposition can be found in [ES96,CEFJ96]. Briefly, we need to compute the canonical state function $\xi_{G \bowtie H}$ and modify the model checker so that it canonicalizes every state encountered during state space traversal. See [Man97] for a summary of known results for computing $\xi_{G \bowtie H}$. Theorem 4 can be further extended along the lines of *Auto f* in [ES96] by introducing an additional set of labels corresponding to all sub-formulas that have E, X or U as the topmost operator. Although we omit the theorem from this paper, we note that computation of the corresponding $G \bowtie H$ can still be automated.

5 Structural Symmetries

In Section 3.2, we saw how knowledge of groups $G \leq Aut_M\ L \cdot AP$ and $H \leq Aut_f\ AP \cdot SF$ would help us partition sub-formulas of a CTL* formula f into equivalence classes. In Section 4, we saw how knowledge of the same group G but a different $H \leq Aut_f\ AP \cdot MPS$ would allow us to construct quotient structures. In both the cases, we need to compute $G \bowtie H$. We now describe how G, H and $G \bowtie H$ can be computed automatically from net-lists of digital circuits and CTL* formulas, with no assistance from the designer. We have chosen BLIF-MV [B+91] as a representative structural hardware description language.

5.1 Characterizing a BLIF-MV Circuit

We model a BLIF-MV circuit as a five tuple $\mathcal{C} = \langle \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{T}, \mathcal{S} \rangle$, consisting of a set of primary input ports $\mathcal{I}$, a set of primary output ports $\mathcal{O}$, a set of latches $\mathcal{L}$, a set of tables $\mathcal{T}$ and a set of interconnection signals $\mathcal{S}$. Intuitively, $\mathcal{C}$ is a big black-box with I/O ports (primary inputs and outputs) consisting of smaller black boxes (tables and latches) whose I/O ports are interconnected with signals.

A table $T \in \mathcal{T}$ has input ports $i_1^T, i_2^T, \ldots$ and output ports $o_1^T, o_2^T, \ldots$. With each output o_i^T, we associate a function f_i^T that takes the ordered tuple $\langle i_1^T, i_2^T, \ldots \rangle$ as its

argument. In general, f_i^T is non-deterministic, as allowed by BLIF-MV. See the figure below for a BLIF-MV table description. A latch $L \in \mathcal{L}$ has two input ports i_1^L, i_2^L and one output port o_1^L. The second input port specifies the initial value for the latch.

Let $P_{sink} = P_{in}^T \cup P_{in}^L \cup \mathcal{O}$, where $P_{in}^T = \bigcup_{T \in \mathcal{T}} \{i_1^T, i_2^T, \ldots\}$, $P_{in}^L = \bigcup_{L \in \mathcal{L}} \{i_1^L\}$. Let $P_{source} = P_{out}^T \cup P_{out}^L \cup \mathcal{I}$, where $P_{out}^T = \bigcup_{T \in \mathcal{T}} \{o_1^T, o_2^T, \ldots\}$ and $P_{out}^L = \bigcup_{L \in \mathcal{L}} \{o_1^L\}$. Thus P_{sink} is the set of primary outputs and input ports of tables and latches, except those for initial values for latches. And P_{source} is the set of all primary inputs and all output ports of tables and latches.

```
.table a b c -> carry
.default 0
1 1 - 1
1 - 1 1
- 1 1 1
```

Each port is associated with a domain. Let $dom(p)$ denote the domain of any port $p \in P_{sink} \cup P_{source}$. For $o_j^T \in P_{out}^T$, let the function $f_j^T(i_1^T, i_2^T, \ldots)$ be the boolean function specified in its table that corresponds to the output produced at o_j^T. This function takes an ordered list of input ports as its arguments. It could be non-deterministic. The interconnection signals $\mathcal{S}$ simply define a relation $S_{ext} \subseteq P_{source} \times P_{sink}$. Also define $S_{int} = \bigcup_{T \in \mathcal{T}} (\{i_1^T, i_2^T, \ldots\} \times \{o_1^T, o_2^T, \ldots\}) \bigcup_{L \in \mathcal{L}} (i_1^L, o_1^L)$. Thus S_{int} captures the internal dependencies of input and output ports within a latch or a table. And S_{ext} captures the *external* dependencies between primary inputs, primary outputs and I/O ports of tables and latches.

A *structural symmetry* of $\mathcal{C}$ is an automorphism $\pi : P_{sink} \cup P_{source} \rightarrow P_{sink} \cup P_{source}$ of the directed unlabeled graph $(P_{sink} \cup P_{source}, S_{int} \cup S_{ext})$ that satisfies the following constraints: (a) $(\forall \mathcal{X} \in \{P_{in}^T, P_{out}^T, P_{in}^L, P_{out}^L, \mathcal{I}, \mathcal{O}\})(\pi\mathcal{X} = \mathcal{X})$, (b) $(\forall p \in P_{sink} \cup P_{source})(dom(p) = dom(\pi p))$, and (c) $(\forall o_j^T \in P_{out}^T)(f_j^T(i_1^T, i_2^T, \ldots) = f_{j'}^{T'}(i_{1'}^{T'}, i_{2'}^{T'}, \ldots))$ where $(o_{j'}^{T'} = \pi o_j^T)$, $(i_{1'}^{T'} = \pi i_1^T)$, $(i_{2'}^{T'} = \pi i_2^T)$, ... It follows from the first two conditions that vertices corresponding to a table get mapped to vertices of another table with the same number of I/O ports such that their domains match. Condition (c) stipulates that even the functionality of the two tables should match modulo π. It may be verified that the set of structural symmetries forms a group.

How are structural symmetries in $\mathcal{C}$ related to symmetries in some Kripke structure M? For a circuit $\mathcal{C}$, there exists a Kripke structure $M = (S, R, K)$ with 2^L states and the set of atomic propositions AP. The set L corresponds to the latches. The set AP corresponds to outputs in $\mathcal{O}$. The function K represents the boolean predicate on latches that generate outputs. We assume that the outputs in $\mathcal{C}$ do not depend on the inputs i.e. $\mathcal{C}$ defines a Moore machine. However, we note that the basic ideas developed in this paper can be extended to Mealy machines also. For a structural symmetry π, let $\pi_{\mathcal{C}} : L \cup AP \rightarrow L \cup AP$ denote the permutation naturally induced by π. Let $Aut_{\mathcal{C}}\ L \cdot AP$ denote the set of all such permutations. $Aut_{\mathcal{C}}\ L \cdot AP$ forms a group.

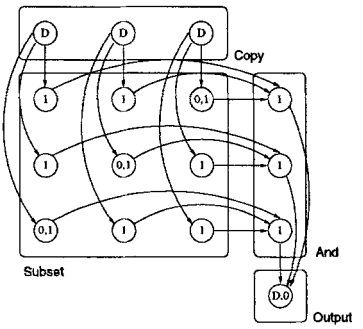
Theorem 5. $Aut_{\mathcal{C}}\ L \cdot AP \leq Aut_M\ L \cdot AP$. □

Although we used BLIF-MV terminology to formalize the notion of structural symmetries, we believe that our definition is general enough to be applicable to gate level descriptions like those expressible in EDIF, Verilog and VHDL.

5.2 Graphs for BLIF-MV Circuits

One problem with the definition of structural symmetries in the previous section is that the third condition cannot be expressed in purely graph theoretic terms. However, we can augment the graph so that there is a 1-1 correspondence between structural symmetries and automorphisms of the graph. This allows us to leverage results from computational group theory developed for identifying graph automorphisms.

First, label each vertex in $P_{sink} \cup P_{source}$ with its domain. Next, substitute each subgraph corresponding to a table by a graph similar to the one shown in the adjoining figure. The new nodes are *internal* to the table and are labeled with the corresponding table entries.



Let A_C denote the labeled directed graph so constructed. For an automorphism π of A_C, let $\pi_A : L \cup AP \rightarrow L \cup AP$ denote the 1-1 mapping naturally defined by π. Let $Aut_C\ L \cdot AP$ denote the set of all such permutations. It can be verified that Theorem 5 still holds. A detailed proof can be found in [Man97], which also shows how multiple-output tables, the "=" construct [B+91], pseudo inputs and other special cases can be handled. The size of the graph is linear in the size of the *flattened* BLIF-MV description.

Here are two interesting theoretical questions: First, is every group possible? Let $G \subseteq Aut_M\ L \cdot AP$. Computation of the canonical state requires $G_{<L>}$, the restriction of G to L, as input. Is there any group $G_{<L>}$ that does not correspond to any BLIF-MV circuit? If so, we can focus on the remaining groups to solve the canonical state problem. However, the answer is negative [Man97]. Second, how hard is it to identify scalarsets? A scalarset is an automorphism of the graph A_C such that the automorphism can be written as a product of disjoint transpositions. Note that A_C is not an arbitrary directed graph. It has been derived from a valid BLIF-MV circuit. See [Man97] for a simple proof that the problem is as hard as graph isomorphism.

5.3 Graphs for CTL* Formulas

To compute $H \leq Aut_f\ AP \cdot SF$, draw the parse tree for the formula f. Label each internal node with the operator it represents. The leaf nodes correspond to propositions in AP. For each internal node labeled *Until*, introduce two new nodes labeled *Left* and *Right*. Replace the edge between *Until* and its left operand by two edges: one from *Until* to *Left* and one from *Left* to the left operand. Replace the edge between *Until* and its right operand similarly. Collapse all leaf nodes representing tha same $p \in AP$ into a single node. Label these nodes with a common color, say *White*. Introduce a new set of nodes, one for every $p \in AP$, labeled identically with a new color, say *Black*. Draw an edge from a *White* node to a *Black* node if they correspond to the same atomic proposition. Let A_f denote the graph we constructed. It is clear that the nodes of A_f, except those labeled *Left* or *Right*, are in 1-1 correspondence with elements of $AP \cup SF$. For every automorphism of the graph A, let π denote its restriction to nodes

corresponding to $AP \cup SF$. The set of all such π forms a group, which we denote by $Aut_{A_f}\ AP \cdot SF$.

Theorem 6. $Aut_{A_f}\ AP \cdot SF \leq Aut_f\ AP \cdot SF$. □

To compute $H \leq Aut_f\ AP \cdot MPS$, first identify MPS, the set of maximal propositional sub-formulas of f. For each $g \in MPS$, construct its parse tree. Label each internal node with the operator it corresponds to. Collapse all leaf nodes which correspond to the same $p \in AP$ into a single node. Label them identically with a new color, say *White*. Introduce a new node for every proposition $p \in AP$, all labeled identically with a new color, say *Black*. Draw an edge from every *White* node to a *Black* node if they correspond to the same atomic proposition p. Re-label each root node corresponding to some $g \in MPS$, with a distinct new color. Let $\tilde{A}_f$ denote the graph we constructed. Let $\pi : AP \cup MPS \rightarrow AP \cup MPS$ denote the permutation corresponding to the restriction of some automorphism of $\tilde{A}_f$ to vertices corresponding to AP and MPS. The labels of $\tilde{A}_f$ ensure that $\pi AP = AP$ and $(\forall g \in MPS)(\pi g = g)$. The set of all such π forms a group, which we denote by $Aut_{\tilde{A}_f}\ AP \cdot MPS$.

Theorem 7. $Aut_{\tilde{A}_f}\ AP \cdot MPS \leq Aut_f\ AP \cdot MPS$. □

5.4 Computing $G \bowtie H$

One approach is to compute the groups G and H separately and then compute $G \bowtie H$ using group-theoretic algorithms. G need be computed only once for a given circuit. However, computing group intersections is as hard as graph isomorphism [Hof80], though polynomial time algorithms do exist for special cases.

A simpler approach is to join the two graphs corresponding to G and H together by drawing an edge between every pair of vertices that correspond to the same $p \in AP$ in both the graphs. The key to correctness lies in the fact that the sets of labels in the two graphs, except for the vertices corresponding to AP, are mutually exclusive.

5.5 The Big Picture

Given a BLIF-MV circuit $\mathcal{C}$, a CTL* formula f and a set of initial states I, we first compute sets of symmetric sub-formulas of f, as defined in Section 3, by constructing the graphs $A_{\mathcal{C}}$ and A_f, described in Section 5.2 and Section 5.3 respectively, joining them as described in Section 5.4 and solving the graph automorphism problem for the resulting graph. The data structure for representing graph automorphisms allows identification of partitions of sub-formulas of f easily. We then compute $H \leq Aut_f\ AP \cdot MPS$ by constructing the graph $\tilde{A}_f$ described in Section 5.3, joining it with $A_{\mathcal{C}}$ as described in Section 5.4 and solving the graph automorphism problem for the resulting graph. This would give us generators for the group $G \bowtie H$, from which we compute $\xi_{G \bowtie H}$. Finally, we feed the sets of symmetric sub-formulas and the function $\xi_{G \bowtie H}$ to a modified model checker that canonicalizes states during state space traversal and uses Theorem 2 to avoid computing truth values for all sub-formulas. After having evaluated the truth value of f for all initial states, the modified model checker can start offering new formulas to the designer, whose truth value can easily be deduced, as described in Section 3.1.

6 Computing Automorphisms

Let us first breeze through a set of definitions. Consider a directed labeled graph $A = (V, E)$ with n vertices and m edges. A **bipartition** P defined over V is a set of ordered pairs $\cup_{1 \le i \le k}\{(V_i, W_i)\}$, where (a) $\cup_{1 \le i \le k} V_i = \cup_{1 \le i \le k} W_i = V$, (b) $(\forall i.1 \le i \le k)(|V_i| = |W_i| \neq 0)$, and (c) $(\forall j.1 \le i < j \le k)(V_i \cap V_j = W_i \cap W_j = \phi)$. The set of edges of a graph or its labeling function play no role in the definition. A bipartition P is a **unipartition** if $(\forall i.1 \le i \le k)(V_i = W_i)$. It is simply a partition of the set of vertices V into disjoint non-empty sets. A bipartition $Q = \cup_{1 \le i \le q}\{V_i^Q, W_i^Q\}$ is a **refinement** of another bipartition $P = \cup_{1 \le i \le p}\{V_i^P, W_i^P\}$ if they are defined over the same set of vertices V and $(\forall i.1 \le i \le q)(\forall j.1 \le j \le p)((V_i^Q \cap V_j^P = \phi) \vee (V_i^Q \subseteq V_j^P \wedge W_i^Q \subseteq W_j^P))$. We denote this relationship by $Q \preccurlyeq P$. We also say that P is coarser than Q and that Q is finer than P. The relation $\preccurlyeq$ is reflexive and transitive. Two bipartitions $P = \cup_{1 \le i \le p}\{(V_i^P, W_i^P)\}$ and $Q = \cup_{1 \le i \le q}\{(V_i^Q, W_i^Q)\}$ are **compatible** if $(\forall i.1 \le i \le p)(\forall j.1 \le j \le q)(|V_i^P \cap V_j^Q| = |W_i^P \cap W_j^Q|)$. The **intersection** of two compatible bipartitions P and Q is defined as $P \curlywedge Q = \cup_{1 \le i < p, 1 \le j \le q}\{(V_i^Q \cap V_j^P, W_i^Q \cap W_j^P)\} - \{(\phi, \phi)\}$, which itself is a bipartition. Let $A = (V, E)$ be a directed labeled graph with labeling function c. A bipartition P is an **automorphism** of A if (a) $(\forall i.1 \le i \le k)(|V_i| = |W_i| = 1)$, (b) $(\forall v, w \in V)(\forall i.1 \le i \le k)((v \in V_i \wedge w \in W_i) \Rightarrow (c(v) = c(w)))$, and (c) $(\forall v, v', w, w' \in V)(\forall i.1 \le i \le k)(\forall j.1 \le j \le k)((v, v') \in E \Leftrightarrow (w, w') \in E)$. The set of all automorphisms of A forms a group. We denote it by $Aut\ A$. A bipartition P is **consistent** with an automorphism if there exists an automorphism P' of A such that $P' \preccurlyeq P$. For notational convenience, we will denote both a vertex $v \in V$ and a singleton set $\{v\}$ by simply v. This allows us to write a set like $\{(\{u\}, \{v\})\}$ as (u, v). The following lemmas are immediate:

Lemma 1. *If P, Q and R are bipartitions of A such that $P \preccurlyeq Q$ and $P \preccurlyeq R$, then Q and R are compatible and $P \preccurlyeq Q \curlywedge R$.*

Lemma 2. *Let U_{max} be a unipartition such that two vertices of A lie in the same set iff they have the same label. Then, U_{max} is consistent with every automorphism of A.*

Lemma 3. *Let U_{min} be a unipartition such that two vertices u and v lie in the same set if and only if $(\exists \pi \in Aut\ A)(\pi u = v)$. Then, U_{min} is consistent with every automorphism of A and is the finest such unipartition.*

Lemma 4. *If P is a bipartition such that $P \in Aut\ A$ and $(u, v) \in P$, then $P \preccurlyeq \{(succ(u), succ(v)), (V - succ(u), V - succ(v))\}$, where $succ(x) = \{y \mid (x, y) \in E\}$.*

Lemma 5. *If P is a bipartition such that $P \in Aut\ A$ and $(u, v) \in P$, then $P \preccurlyeq \{(pred(u), pred(v)), (V - pred(u), V - pred(v))\}$, where $pred(x) = \{y \mid (y, x) \in E\}$.*

We tackle the following problem: *Given a bipartition P for a directed labeled graph $A = (V, E)$, produce an automorphism of A consistent with P, if one exists.*

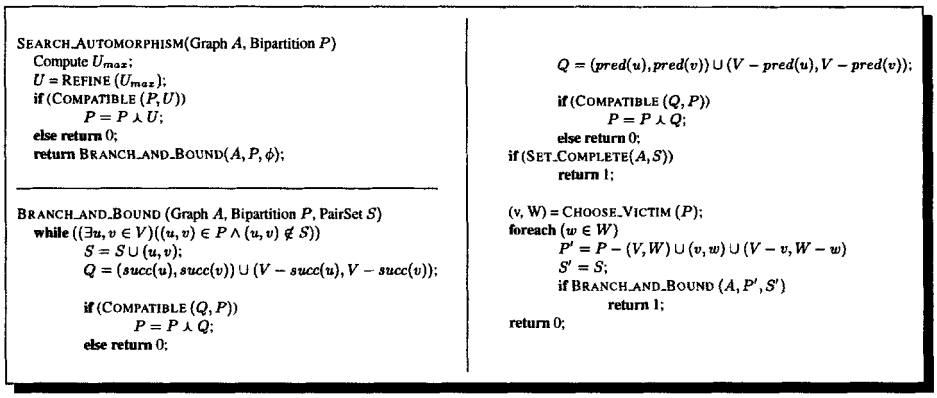


Fig. 1. Algorithm for finding an automorphism, given graph A and bipartition P.

6.1 Branch and Bound Algorithm

Pseudo-code for the algorithm is given in Figure 1. We start with U_{max}, as defined in Lemma 2, since U_{max} is consistent with every automorphism of A. Ideally, we should start with U_{min}, as it is the finest such partition. However, computing U_{min} itself is as hard as graph isomorphism [vL90]. Therefore, we compute an approximation U such that $U_{min} \preccurlyeq U \preccurlyeq U_{max}$ using REFINE, which we describe in detail in Section 6.2. U is consistent with every automorphism of A. Having computed U, we check whether P and U are compatible. If not, then from Lemma 1, P is not consistent with any automorphism of A; the algorithm terminates. If P and U are compatible, we compute their intersection $P \curlywedge U$. From Lemma 1, any automorphism consistent with P and U has to be consistent with $P \curlywedge U$. Finally, we invoke BRANCH_AND_BOUND.

The Bounding Step is the *while* loop in BRANCH_AND_BOUND is the bounding step. From Lemma 4, we conclude that if $(u, v) \in P$ and if an automorphism is consistent with P, then it has to be consistent with $Q = \{(succ(u), succ(v)), (V - succ(u), V - succ(v))\}$ as well. From Lemma 1, we conclude that P and Q must be compatible and that the automorphism must be consistent with $P \curlywedge Q$ as well. A similar argument holds for $Q = \{(pred(u), pred(v)), (V - pred(u), V - pred(v))\}$ also. If P and Q are non-compatible, BRANCH_AND_BOUND terminates.

The bounding step also helps to refine P by computing $P \curlywedge Q$, which in turn might generate new singleton pairs $(u, v) \in P$. Intuitively, the *implications* of mapping u to v are getting *propagated*. The set S remembers such pairs (u, v), thereby avoiding duplicate work. The *while* loop terminates when no such pairs remain. At this point, all pairs in P which have size one, lie in S. The function SET_COMPLETE checks whether all vertices in V have found their way into S. If so, we have discovered an automorphism. Otherwise, it is time to branch.

The Branching Step: CHOOSE_VICTIM first selects a pair (V, W) in P such that $|V| \neq 1$ using some heuristic. It then selects a vertex $v \in V$ using another heuristic and returns (v, W). The choice of v and W is important for at least two reasons. First, small

sized W implies fewer branches to explore. Second, branches that lead to dead ends need be avoided. Our implementation is not fancy: we simply choose the smallest sized W available, breaking ties arbitrarily; our choice of $v \in V$ is also arbitrary. Having chosen v and W, we try to discover $w \in W$ such that v maps to w in some automorphism of A. To this end, we compute $P' = P - (V, W) \cup (v, w) \cup (V - v, W - w)$ and invoke BRANCH_AND_BOUND. Clearly, if all choices of w fail, there is no automorphism consistent with P and the function terminates unsuccessfully.

Lemma 6. *For a directed graph $A = (V, E)$, a bipartition $P = \cup_{1 \le i \le n}\{(V_i, W_i)\}$ is an automorphism of A if and only if (a) $(\forall i.1 \le i \le n)(|V_i| = |W_i| = 1)$, (b) $P \preccurlyeq U_{max}$, and (c) $(\forall i.1 \le i \le n)(\forall v, w \in V)((v \in V_i \wedge w \in W_i) \Rightarrow (P \preccurlyeq \{(succ(v), succ(w)), (V - succ(v), V - succ(w))\}$* □

Proof of Correctness: Condition (a) is verified by SET_COMPLETE before termination. Condition (b) is true because SEARCH_AUTOMORPHISM computes $P \curlywedge U$ where $U \preccurlyeq U_{max}$. Condition (c) is checked for each vertex pair in the *while* loop. The entire algorithm simply verifies Condition 3 for vertex pairs generated by the branching step.

Time Complexity: Testing compatibility and computing intersection of two bipartitions require $O(n)$ time. Computing U_{max} is trivial. From Figure 1, it might appear that each level of recursion is required to store its own copy of P and S. However, this can be obviated by remembering set boundaries at each recursion level and quickly merging subsets when backtracking. Our implementation uses only nine arrays of size n, apart from the usual adjacency lists for edges. If we never backtrack and if the size of sets returned by CHOOSE_VICTIM is bounded by a constant, our implementation runs in $O(m + n)$ time.

6.2 Refinement

REFINE computes a unipartition U such that $U_{min} \preccurlyeq U \preccurlyeq U_{max}$. Why is refinement useful? First, it might generate singleton pairs whose implications can be propagated immediately in the bounding step. Second, by shrinking the sizes of pairs of sets, fewer branches may have to be explored later. Ideally, if a graph has no non-trivial automorphisms, all sets in U should be singleton.

A unipartition U can also be looked upon as a function that computes the same value for two vertices if they lie in the same set. Some such functions that satisfy $U_{min} \preccurlyeq U$ are easy to compute. The intersection of two such functions $U_1 \curlywedge U_2$ is also guaranteed to be at least as coarse as U_{min}. Such functions are called *vertex invariants* [FH$^+$83]. Some vertex invariants that can be computed in $O(m + n)$ time are the in-degree and out-degree of vertices, the set of degrees of vertices incident at a vertex and the set of degrees of vertices which a vertex is incident upon. See [Man97] for references to articles that describe other vertex invariants that are more expensive to compute.

An important trick is to treat a unipartition U as a labeling function and use it to refine itself. For a vertex v, let U' compute the set of labels of vertices incident upon v. Then U' is a vertex invariant such that $U_{min} \preccurlyeq U'$ [FH$^+$83]. U can be refined by computing $U \curlywedge U'$ in $O(m + n)$ time repeatedly. At most $n - 1$ iterations are required. In practice, a few iterations suffice.

6.3 The Automorphism Group

It is possible to produce *all* the automorphisms by continuing the search even after discovering the first one. Since their total number could be exponential in n, we need a succinct representation of $Aut\ A$. A detailed description of an algorithm for computing $Aut\ A$, that draws ideas from computational group theory and uses the algorithm in Figure 1 as a backbone, is given in [Man97]. We omit its discussion from this paper as it is yet to be implemented.

We initially experimented with a software package called GAP [Gap], which offers a graph automorphism program called *nauti* [McK90] based on one of the earliest such programs written by McKay [McK81]. It is natural to ask: Why write another graph automorphism program? Existing packages are general purpose and carry around a lot of baggage. We found GAP to be slow. We can exploit a lot of structure in the graphs we construct. For a detailed description of several other motivating reasons see [Man97].

7 Experimental Results

We implemented the algorithm in Figure 1 to convince ourselves that our modeling of the circuit is sufficient to allow discovering symmetries. As it stands, it is useful when a circuit verifier suspects that certain symmetries exist in the circuit at hand. She can ratify it by providing a bipartition using her intuition and running our algorithm.

Example	n	m	C	C_1	C_2	C_f	*iter*
ctlp20	4920	6740	15	34	51	246	7
ping-pong	288	378	11	25	39	144	7
z4ml	527	929	5	14	19	108	4
4-arbit	3158	4000	19	52	105	3110	60

Example	*back-track*	*maxset*	*numchoices*
ctlp20	0	20	19
ping-pong	0	2	1
z4ml	8	4	21

Starting with a Verilog description, we obtain a BLIF-MV description using a compiler called `vl2mv` written by Cheng [CYB93]. The BLIF-MV description is *flattened* using a standard VIS command. The flattened circuit along with a bipartition is fed to our program which first generates a suitable labeled directed graph, then refines the labels and finally runs the branch and bound algorithm. We identified symmetries in all the examples in the second table. `ctlp20` solves the dining philosophers problem for 20 philosophers. It has a cyclic group. `ping-pong` has a fully symmetric system of size 2. `z4ml` is a combinational circuit whose inputs constitute three sets of fully symmetric variables. We tabulate n, m and C, the number of vertices, edges and colors respectively, in the initial graph. Since refinement impacts the running time of the algorithm greatly, we also tabulate the number of colors after successive refinement steps. C_1 denotes the number of colors after the in-degree and out-degrees have been used as vertex invariants. C_2 denotes the number of colors after the set of in-degrees of fan out vertices and the set of out-degrees of fan out vertices have been used as vertex invariants. C_f denotes the final number of colors after iterative refinement. The number of iterations is listed under *iter*. We also list the number of times our branch and bound algorithm had to backtrack, the number of times CHOOSE_VICTIM was invoked and the maximum size of the set returned by this routine. Our algorithm runs in linear time if we never backtrack and if the size of sets returned by CHOOSE_VICTIM is bounded. The table shows that the two conditions are almost satisfied.

References

[AT96] M. AGRAWAL AND T. THIERAUF. The Boolean Isomorphism Problem. In *Proc. Symp. on Foundations of Computer Science*, pp. 422–430, October 1996.

[B+91] R. K. BRAYTON ET AL. BLIF-MV: An Interchange Format for Design Verification and Synthesis. Technical Report UCB/ERL M91/97, UC Berkeley, November 1991.

[BCL+94] J. R. BURCH, E. M. CLARKE, D. E. LONG, K. L. MCMILLAN, AND D. L. DILL. Symbolic Model Checking for Sequential Circuit Verification. *IEEE Tran. on Comp. Aided Design of Integrated Circuits and Sys.*, 13(4):401–424, April 1994.

[CEFJ96] E. M. CLARKE, R. ENDERS, T. FILKORN, AND S. JHA. Exploiting Symmetry in Temporal Logic Model Checking. *Formal Meth. in Sys. Design*, 9(1/2):77–104, 1996.

[CES86] E. M. CLARKE, E. A. EMERSON, AND A. P. SISTLA. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CYB93] S.-T. CHENG, G. YORK, AND R. K. BRAYTON. VL2MV: A Compiler from Verilog to BLIF-MV, October 1993.

[EJP97] E. A. EMERSON, S. JHA, AND D. PELED. Combining Partial Order and Symmetry Reductions. In *Proc. TACAS 97*, pp. 19–34, April 1997.

[ES95] E. A. EMERSON AND A. P. SISTLA. Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic Approach. In *Proc. CAV 95*, pp. 309–324, July 1995.

[ES96] E. A. EMERSON AND A. P. SISTLA. Symmetry and Model Checking. *Formal Meth. in Sys. Design*, 9(1/2):105–131, 1996.

[FH+83] G. FOWLER, R. HARALICK, ET AL. Efficient Graph Automorphism by Vertex Partitioning. *Aritificial Intelligence*, 21:245–269, 1983.

[Gap] GAP: Groups, Algorithms and Programs, Version 3, Release 4. Available via ftp from `ftp.math.rwth-aachen.de`, directory `/pub/gap`.

[GS97] V. GYURIS AND A. P. SISTLA. On-the-Fly Model Checking under Fairness that Exploits Symmetry. In *Proc. CAV 97, Haifa, Israel, June 1997*, pp. 232–243, 1997.

[Hof80] C. M. HOFFMAN. On the Complexity of Intersecting Permutation Groups and its Relationship with Graph Isomorphism. Technical Report 4/80, Institüt for Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, 1980.

[ID96] C. N. IP AND D. L. DILL. Better Verification Through Symmetry. *Formal Meth. in Sys. Design*, 9(1/2):41–76, 1996.

[Man97] GURMEET SINGH MANKU. Structural Symmetries and Model Checking. Master's thesis UCB/ERL M97/92, University of California at Berkeley, 1997. Available as `http://www-cad.eecs.berkeley.edu/~manku/papers/ms.ps.gz`.

[McK81] B. D. MCKAY. Practical Graph Isomorphism. In *Proc. Tenth Manitoba Conf. on Numerical Math. and Computing, Winnepeg, 1980, vol 1*, pp. 45–87, 1981.

[McK90] B. D. MCKAY. Nauty Users Guide (Version 1.5). Technical Report TR-CS-90-02, Computer Science Department, Australian National University, Australia, 1990.

[PB97] M. PANDEY AND E. BRYANT. Exploiting Symmetry when Verifying Transistor-Level Circuits by Symbolic Trajectory Evaluation. In *Proc. CAV 97, Haifa, Israel, June 1997*, pp. 244–255, 1997.

[vL90] J. VAN LEEUWEN. Graph Algorithms. In *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, pp. 525–631. Elsevier Science, 1990.