# A New Heuristic for Bad Cycle Detection Using BDDs

R. H. Hardin[1] R. P. Kurshan[1]
S. K. Shukla[2] M. Y. Vardi[3]

**Abstract.** We describe a new heuristic for detecting bad cycles (reachable cycles that are not confined within one or another designated sets of model states), a fundamental operation for model-checking algorithms. It is a variation on a standard implementation of the Emerson-Lei algorithm, which our experimental data suggests can result in a significant speed-up for verification runs that pass. We conclude that this heuristic can be used to advantage on "mature" designs for which the anticipated result of the verification is *pass*.

## 1 Introduction

It is well known that the model-checking problem for the linear-time temporal logic LTL, the branching time temporal logic CTL, and $\omega$-automaton specifications are all solvable in time that is linear in the size of the model [CES86, LP85, VW86]. In most existing approaches, the model-checking problem is reduced to the graph problem of finding a "bad" cycle, *i.e.*, a reachable cycle that is not contained in one or more designated set of nodes [CES86, CVWY91, Kur94, LP85]. Since this graph problem has a well known linear-time solution using depth-first search [Kur94], the linear-time complexity of the model-checking problems follows.

In this paper, we concentrate on the bad cycle detection problem. Our experimental data derives from our implementation of the titled heuristic in the verification tool COSPAN [HHK96, Kur94]. COSPAN applies to a model-checking paradigm in which both the program and the specification are represented by $\omega$-automata. Fairness is captured by automata acceptance conditions, and model-checking consists of checking that the language of the program is contained in the language of its specification (in terms of their respective representations as automata). As with the other model-checking paradigms cited above, COSPAN performs this check by searching a derived reachable state space for "bad" – or more specifically, unfair – cycles, captured as reachable cycles not contained in

[1] Bell Labs, Murray Hill, NJ 07974, {rhh,k}@research.bell-labs.com
[2] Computer Science Department, State University of New York at Albany, NY 12222 sandeep@cs.albany.edu
[3] Department of Computer Science, Rice University, Houston, TX 77005-1892, vardi@cs.rice.edu, http://www.cs.rice.edu/~vardi. Supported in part by NSF grant CCR-9628400. Part of this work was done in Bell Laboratories during the DIMACS Special Year on Logic and Algorithms.

one or more specified sets of states [Kur94]. In COSPAN, the derived state space is formed from the product of the automata representing the program and the complement of the specification, and the specified sets of states are the *cycle sets* defined as part of the respective automata acceptance structures. The language containment check is verified if and only if there are no bad cycles in this derived state space (hence the name).

Model checking algorithms can be classified broadly according to the method of state reachability employed: *explicit* state enumeration, in which states are generated as explicit data addresses, or *implicit* (or "symbolic") state enumeration, in which states are represented implicitly as a solution of an equation. An obvious apparent advantage of implicit over explicit is that the size of the model state space that can be verified is not intrinsically limited by a linear function of the available computational space: a syntactically small equation can define a very large set of states. Unfortunately, in the worst case, the reachability problem in both paradigms is PSPACE-complete in the number of program variables. Nonetheless, implicit state enumeration has found many applications in industrial-scale verificational problems, on account of its potential to search very large state spaces [BCM+92].

The most successful implementation of implicit state enumeration currently is in terms of binary decision diagrams (BDDs) [BCM+92, McM93, TBK91, TBK95]. Thus, many modern verification tools, for example SMV [McM93] and COSPAN [HHK96], employ implicit state enumeration implemented with BDDs. (COSPAN also employs explicit state enumeration that in some circumstances is more efficient, and can be used effectively in concert with implicit state enumeration.)

In model-checking based upon explicit state enumeration, cycle detection may be done in time linear in the size of the program state space, by using depth-first search to find the strongly connected components of a graph derived from the product of the automata representing the program and the specification [Kur94]. All reachable cycles are contained in the strongly connected components of the graph (which are cycles themselves), and and it is enough to check that no strongly connected component is a bad cycle.

In BDD-based model-checking, a depth-first search is incompatible with the routines used to efficiently manipulate the BDDs. Instead, the cycle detection algorithms in BDD-based algorithms [McM93, TBK91, TBK95] generally employ some form of the Emerson-Lei algorithm for model checking [EL86]. These algorithms have time complexity quadratic in the size of the underlying state space because the Emerson-Lei algorithm employs a double nesting of fixed-point computations. Each inner fixed-point computation involves a number of iterations that is linear in the diameter of the state space, and the number of times the inner fixed-point is computed is linear in the size of the state space. Thus, even for small-size BDDs ($O(\log n)$), the time complexity is $O(n^2 \log^2 n)$. In principle, one may think it is possible in implicit state enumeration to detect bad cycles more efficiently by means of a transitive-closure computation, which requires only a linear number of steps to find bad cycles, following a logarithmic number of BDD-

manipulation iterations to compute the transitive closure of a state space of size $n$, using successive path doubling. For small BDDs, the time complexity of the transitive-closure algorithm is polylogarithmic. In practice, however, the BDDs that define the transitive-closure become rather large, and empirical results have demonstrated that whatever time may be saved in a cycle-detection algorithm employing a transitive-closure subroutine is more than lost in manipulating the larger-size BDDs required to represent it [HTBK93, TBK91, TBK95]. In most applications, the Emerson-Lei algorithm advantageously trades the quadratic increase in algorithmic steps for a decrease in the number of steps required to manipulate the associated BDDs, resulting in fewer steps over-all (and requiring less space than the transitive-closure algorithm, as well). Thus, current implementations based upon the Emerson-Lei algorithm not only require less space than an implementation based upon transitive closure, but are empirically much faster as well.

In practice, the Emerson-Lei algorithm is often quite slow and is acceptable only for lack of a better algorithm. It is not unusual to find examples in which bad-cycle detection takes 2-10 times longer than reachability. It is known that the Emerson-Lei formula cannot be expressed as an alternation-free fixed-point formula [EL86], (which would have yielded a linear-time algorithm for the implicit state cycle detection also [Cle93]). It is an open question whether there is an algorithm for implicit-state cycle detection that has an over-all time complexity better than Emerson-Lei, when manipulation of the underlying data-structure (say, BDDs) is taken into account. We do not solve this problem in this paper; rather we focus on heuristics to improve the performance of the Emerson-Lei algorithm for cycle detection.

We present a heuristic and experimental evidence based upon running 10K randomly generated models, that this heuristic can significantly speed up bad cycle detection in the case that no bad cycle in fact is present. When there is no bad cycle, which implies that the system satisfies its specification, our heuristic outperforms the Emerson-Lei algorithm in almost all the longer-running models we tried, sometimes very significantly (performance measured in terms of running time). On the other hand, when there is a bad cycle, which means that the property being checked is not satisfied by the system, the normal implementation performs generally better. (The new heuristic and the normal implementation require comparable BDD sizes and space.) This consistent comparative performance suggests a guided use of model-checking in the following fashion. If the user has a reasonable expectation that the verification will result in a positive answer, use the new heuristic. If there is a reason to believe otherwise, use the existing algorithm. Often the user has a reasonable expectation of the outcome of verification: if the design is "green", it likely has many bugs and failure is anticipated; if the design already has been verified and a small change is made to the model, it may be anticipated that the verification most likely will pass. In the latter case, the new heuristic is recommended. A particular example of this latter case is regression verification [HKM+96], for which the choice of the new heuristic may be automated. Moreover, in any case, both algorithms may be run concurrently on separate processors to yield an average speedup over all.

## 2 Previous work

For perspective, we state the two basic algorithms for bad cycle detection in the context of implicit state enumeration: transitive closure and the Emerson-Lei algorithm. As stated in the Introduction, experimental evidence [HTBK93] suggests that transitive closure not only results in a larger memory requirement, but also in an empirically slower bad cycle check, on account of the added overhead of manipulating larger-sized BDD's. It is hard to find an example in which the transitive-closure algorithm outperforms the Emerson-Lei algorithm.

### 2.1 Bad Cycle Detection Based on Transitive Closure

Let $R(x,y)$ denote the Boolean function representing the transitive closure of the transition relation $T(x,y)$ determining the set of transitions from a state $x$ to a state $y$. Then $(R(x,y) \wedge R(y,x))$ implicitly defines the set of cycles involving both $x$ and $y$. Let $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ denote the *cycle sets* of states in which cycles are allowed. For each $\mathcal{C}_i$, a set $NC_{\mathcal{C}_i}$ is computed as follows:

$$NC_{\mathcal{C}_i}(x) = \exists y((R(x,y) \wedge R(y,x)) \wedge \overline{\mathcal{C}_i(y)})$$

A state $x$ belongs to $NC_{\mathcal{C}_i}$ if and only if there is a cycle involving $x$ that is not entirely contained in $\mathcal{C}_i$.

**Theorem 1.** [TBK91, TBK95] *Every reachable cycle is contained in at least one of the cycle sets $\mathcal{C}_i$ iff the set $NC$ is empty, where $NC = \cap_{1 \leq i \leq n} NC_{\mathcal{C}_i}$.*

The transitive-closure relation can be computed using a number of iterations that is logarithmic in the size of the state space, using successive path-doubling. Nevertheless, in practice the transitive-closure computation is very expensive and it was shown experimentally that even with various implementations of the transitive-closure computation, transitive-closure-based algorithms were much slower than the algorithms based on the Emerson-Lei formula described in the next subsection [TBK91, TBK95].

### 2.2 Bad Cycle Detection Based on The Emerson-Lei Formula

In [EL86] Emerson and Lei introduced a $\mu$-calculus formula of alternation depth 1, which expressed an FCTL fairness constraint. In [TBK91, TBK95] this formula was used to find bad cycles in the context of checking language containment. The main idea is to compute the set $NC_\mu$ of reachable states from which set $NC$ can be reached. Hence, emptiness of $NC$ is equivalent to the emptiness of $NC_\mu$.

$NC_\mu$ is obtained as the greatest fixed point of the following function $F$:

$$F(c)(x) = c(x) \wedge \bigwedge_{i=1}^{n} \exists y(T(x,y) \wedge lfp(G_i^c)(y))$$

Evaluation of $F$ requires the computation of $n$ least fixed points (lfp), one per function of $G_i^c$, for each iteration of $F$. For a given set $c$, $G_i^c$ is defined as follows:

$$G_i^c(d)(x) = c(x) \wedge (\overline{C_i}(x) \vee \exists y(T(x,y) \wedge d(y)))$$

The following theorem is central to the correctness of the methods based on the Emerson-Lei formula.

**Theorem 2.** [TBK95] *$NC_\mu$ is the set of reachable states from which $NC$ can be reached.*

It is clear from the alternation of the fixed points that the complexity of these algorithms will be quadratic in the size of the underlying structure (BDD nodes, and in the worse case, states). Moreover, it was shown that this fixed point formula cannot be expressed in the alternation-free fragment of the $\mu$-calculus [EL86]. Thus, it does not seem that a linear-time algorithm will be possible based on this formalism. Instead, we propose a heuristic that is based upon the idea of computing quickly the set of *bad states*: states lying on some bad cycle, by successive approximation. Our heuristic strategy can be described as a *catch-them-young* strategy as explained in the next section.

## 3 New Heuristics

In this section we describe our *catch-them-young* heuristic for faster refinement of the set of bad states. The central idea is to use combined forward and backward reachability to find out if there is a $C_i$ that contains the set of all bad states relative to $\{C_1, \ldots, C_{i-1}\}$. If that is the case for some $i$, then there is no bad cycle.

### 3.1 Description of the Basic Heuristic and Correctness Proof

Let $S$ be the state space of the system and $\{C_1, C_2, ..., C_n\}$ be the set of cycle sets. We may assume that all relations are restricted to the set of reachable states. Now construct the BDDs for the complements of cycle sets and call them $F_1, F_2, ..., F_n$. So $F_i = S \setminus C_i$. A bad cycle must pass through all the $F_i$'s.

We now describe the successive steps for constructing the set $B$ of *bad states* (states lying on some bad cycle). $B$ will be the set limit of its monotonically decreasing successive approximations $B^\#$. In any iteration, $B^\#$ is the set of states through which some bad cycle potentially may pass. Let

$$^*F_i = \{x \mid \text{there is a path from state } x \text{ to some state in } F_i\} ,$$

$$F_i^* = \{x \mid \text{there is a state in } F_i \text{ from which there is a path to } x\} .$$

These sets can be computed using standard backward and forward reachability (respectively), using a linear number of iterations. Let $F_i^\# = {}^*F_i \cap F_i^*$, $F^\# = \bigcap_{i=1}^n F_i^\#$.

If we now denote by $f_i$, $f_i^*$, $^*f_i$, and $f^\#$ the characteristic functions of $F_i$, $F_i^*$, $^*F_i$, and $F^\#$, respectively, then $f^\#$ can be computed as follows:

$f\# := true$
for all $i$ do
$f_i := f_i \wedge b$
$f^\# := f^\# \wedge f_i^* \wedge^* f_i$

Here $b$ is the characteristic function of $B^\#$, which is initially *true*. Note that if there is a cycle through $x$ that visits some state in $F_i$, then $x \in F_i^\#$. Hence all the states on a bad cycle, must be in $F^\#$. Our first approximation to the set of bad states is thus given as $B^\# = F^\#$.

**Lemma 3.** *$F^\#$ contains all the bad states.*

Note, however, that $F^\#$ is only an approximation to $B$, as there may be some states in $F^\#$ that are not on any bad cycle. Hence, we have now to refine the set $B^\#$ to discard at least some of those states until either we can cover $B^\#$ by one of the $C_i$'s, at which point we can stop since no bad cycle can exist, or the approximation reaches a fixed point where no more states can be thrown out of $B^\#$.

At this point we apply our *catch-them-young* strategy. The strategy is based on the observation in Lemma 4, which helps us to identify *early on* some states that provably do not belong to the set of bad states.

**Lemma 4.** *If $s \in B$ then there must be a predecessor $s'$ of $s$ in the state reachability graph such that $s' \in B$. Similarly, if $s \in B$ then there must be a successor $s'$ of $s$ in the state reachability graph such that $s' \in B$.*

Due to Lemma 4, we may identify those states that have none of their predecessors and none of their successors in $B^\#$ to be the ones that cannot be bad states. So we throw these states out of the set $B^\#$. Note that this is possible because of the monotonicity of our approximation (refinement) and Lemma 4. Hence the next step is to change the characteristic function of $B^\#$ to a new one as follows:

$$B^\#(x) := B^\#(x) \wedge (\exists y(T(x,y) \wedge B^\#(y))) \wedge (\exists y(T(y,x) \wedge B^\#(y))).$$

If this step changes $B^\#$ then we keep deleting more such states repeating the above BDD manipulation until a fixed point is reached.

Let $b$, $b^+$ and ${}^+b$ denote the characteristic functions of $B^\#$, the set of successor of states in $B^\#$, and the set of predecessors of states in $B^\#$. Then the next approximation $B^\#$ can be computed as follows.

$b := f\#$
while $b$ changes do
$b := b \wedge b^+$
$b := b \wedge^+ b$

Now we have a better approximation of $B$. However, at this point we throw out from each $F_i$, the states that have already been known not to be bad states. This is accomplished by $F_i := F_i \cap B^{\#}$.

Then we again compute $F^{\#}$. If there are bad cycles, this set will contain all the states on those cycles. It may contain, however, also some good states. Thus, we need to repeat the calculation of $B^{\#}$ as described above, starting with $B^{\#} := F^{\#}$. By repeating this approximation of $B$ several times, when we reach a fixed point, we check if there is some $C_i$ that fully covers $B^{\#}$. If there is one, then we know that there is no bad cycle, and if there is none, then there is a bad cycle.

The correctness of the algorithm depends on the following proposition.

**Proposition 5**

- *If there are no bad cycles, then the fixed point of $B^{\#}$ is empty.*
- *If there are bad cycles, then the fixed point of $B^{\#}$ contains a strongly connected component with states from each $F_i$.*

Since the sequence of successive approximations to $B$ is monotonic, the operations on $B^{\#}$ are iterated a number of times that is only linear in the size of the state space. Note, however, that this does not eliminated the doubly nested loop of the Emerson-Lei algorithm; at best this might speed up the convergence of the algorithm.

### 3.2 Implementational Improvements by transformational design

The heuristic approach explained in the previous section gives rise to the following implementation.

1. b := S
2. $f\# := true$
   for all $i$ do
   $f_i := f_i \wedge b$
   $f^{\#} := f^{\#} \wedge f_i^{*} \wedge^{*} f_i$
3. $b := f\#$
   while $b$ changes do
   $b := b \wedge b^{+}$
   $b := b \wedge^{+} b$
4. If $b$ is changed then go to step 2 else go to step 5;
5. If $b = \emptyset$ then there is no bad cycle else $b$ contains bad cycle.

The correctness of this implementation follows from the correctness argument of the previous subsection. However, we did a few more correctness preserving transformation on this implementation to optimize the performance.

Note that in the above implementation, we have created a BDD for $b$, and a BDD for $f\#$, but eventually, $b$ is assigned the same BDD as $f\#$. Hence, we

can do the computations on $b$ in step 2, rather than creating an intermediate representation $f\#$. This speeds up the implementation quite a bit.

We get the final version of our heuristic by using a simple technique in David Long's implementation in COSPAN of the Emerson-Lei algorithm: if at any point of our successive refinement of $B^{\#}$, we find an $F_i$ such that $B^{\#} \subseteq F_i$ and that is not the only $F_i$ that is left, we can drop this $F_i$ from the further steps of the algorithm. The reason is that since the refinement of $B^{\#}$ is monotonic, after $B^{\#} \subseteq F_i$ holds, any further refinement of $B^{\#}$ will still be a subset of $F_i$. This means that at this point, it is guaranteed that if there is any bad cycle then it certainly passes through $F_i$. Hence in the computation of bad states, $\mathcal{C}_i$ is an irrelevant cycle set.

### 3.3 Experimental Results and their Interpretations

In order to test the performance of the new heuristic against the standard Emerson-Lei algorithm, we randomly generated 10K small models and ran them in COSPAN, using the two algorithms in succession. The random models were generated as one to seven processes which follow each other in identical graphs, one process making a transition at a time. The identical graphs contain from 1 to 17 nodes, and are randomly connected with a fanout of 1 to 3. The number of processes, the fanout at a node, and the number of nodes all were chosen uniformly. Each model had a passing version, where each strongly connected component was exactly matched by cycle sets or canceled by *recur edges* [Kur94] (another part, together with cycle sets, of the automaton acceptance structure for the type of automata used with COSPAN), and a failing version, where one of these cycle sets or recur edges was omitted. The generated data shows that whereas the ordinary Emerson-Lei algorithm is generally faster than the new heuristic on models that fail, the new heuristic tends to be better on models that pass (see Figure 1). For longer passing runs, when the Emerson-Lei algorithm runs slowly, the new heuristic can be as much as 10× faster, whereas for the top third longest runs, the Emerson-Lei algorithm was almost never faster than the heuristic. For passing runs, the Emerson-Lei algorithm outperformed the new heuristic by a factor of 3× or more in fewer than .001 of the models, and almost all of those were among the shorter runs. The new heuristic outperformed the Emerson-Lei algorithm by a factor of 3× or more in .04 of the models, mostly among the longer runs. The new heuristic thus seems to be much less likely to get into whatever difficulty is causing long runs, for passing runs. Overall, as an arithmetic average, the heuristic ran 18 percent faster for passing runs, and 36 percent slower for failing runs.

The size of the models, measured in number of bdd nodes, was essentially unchanged as a function of algorithm or pass/fail.

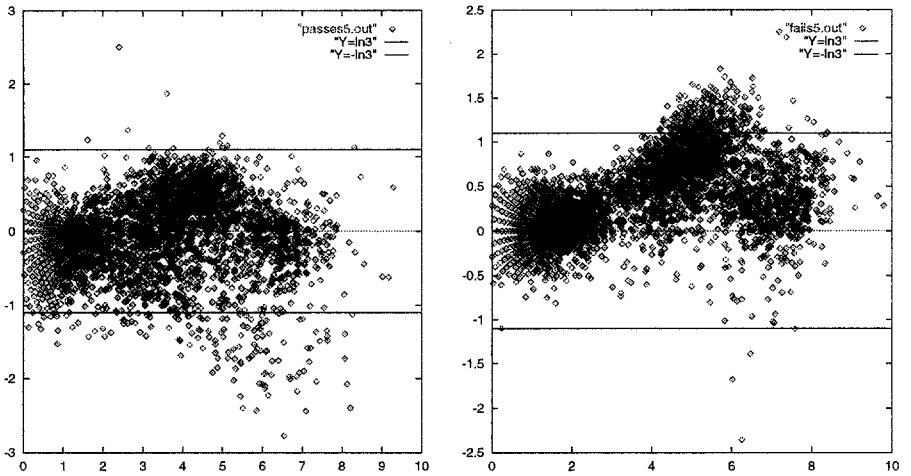


**Fig. 1.** Plotted points are of the form $(ln(r/r_0), ln(t_1/t_0))$, one point for each randomly generated model, where $t_1$ is the run time for the new heuristic and $t_0$ is the run time for the ordinary Emerson-Lei algorithm. $r$ is $sqrt(t_1^2 + t_0^2)$ and represents a sort of average run time, and finally $r_0$ is 0.1 seconds. The horizontal lines show run time ratios of 3 and 1/3. The "fish tails" at small $r$ indicate the timing granularity of .01 seconds. After eliminating from the plots runs with times reported as "0 seconds" or with $r$ less than $r_0$, 3929 models passed and 5724 models failed.

We also ran the new heuristic on some longer-running hardware and software design models. While the number of models was too small to have experimental significance, the results were consistent with the randomly generated models, with one possible exception: there was some indication in these design-model runs that the incremental benefit of the new heuristic may be directly related to the number of cycle sets in the model.

## 4 Conclusion and Future work

We have described a variation of the Emerson-Lei algorithm applied to "bad" cycle detection. It uses a combination of backward and forward reachability limited to a successively restricted subset of the reachable states, and successively eliminates irrelevant cycle sets. This heuristic has been implemented in COSPAN, and run on approximately 10K randomly generated models. The results of these runs suggest that the new heuristic can be much faster than the Emerson-Lei algorithm on models that pass the verification, and essentially never slower on the longer-running models. On the other hand, the Emerson-Lei algorithm seems generally faster for models that fail. Therefore, it is recommended that this new

heuristic be applied to models which are expected to pass verification, for example, models which have already been verified but need to be reverified on account of a small change in the model. If two computers are available, the two algorithms could be run in parallel, decreasing the expected termination time.

It is not known how well this heuristic does on models which derive from rationally generated (non-random) designs, although some preliminary experiments suggest that the heuristic will be beneficial on real designs as well, especially those with many cycle sets (fairness constraints). We plan to do further experimentation using commercial models. More investigation is needed to understand why this heuristic is skewed in favor of models which pass, something which is not understood by the authors.

This heuristic supports the general model-checking strategy of creating different algorithms for different special cases. This is useful as long as the applicable case can be predicted in advance, and is especially useful when the choice of heuristic can be automated. The heuristic presented here joins a number of such special-case algorithms which have been implemented to considerable advantage in COSPAN. In the case of this heuristic, it is a natural choice for regression verification [HKM+96] in which the verification is anticipated to pass. We hope there will be more work in the direction of special-purpose heuristics such as this one.

## References


[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: $10^{20}$ states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verifications of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[Cle93] R. Cleaveland. A linear-time model-checking algorithm for the alternation-free modal $\mu$-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[CVWY91] C. Courcoubetis, M. Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Lecture Notes in Computer Science*, 531:233–245, 1991.

[EL86] E. A. Emerson and C. L. Lei. Efficient model checking in fragments of the propositional modal mu-calculus. In *Proceedings of LICS 1986*, pages 267–278, 1986.

[HHK96] R. H. Hardin, Z. Har'El, and R. P. Kurshan. COSPAN. *Lecture Notes in Computer Science*, 1102:423–427, 1996.

[HKM+96] R. H. Hardin, R. P. Kurshan, K. L. McMillan, J. A. Reeds, and N. J. A. Sloane. Efficient regression verification. *IEE Proc. WODES'96*, pages 147–150, 1996.

[HTBK93] R. Hojati, H. J. Touati, R. K. Brayton, and R. P. Kurshan. Efficient $\omega$-regular language containment. In *Proceedings of CAV 93, LNCS 663*, pages 396–409, 1993.

[Kur94] R. P. Kurshan. *Computer Aided Verification of Coordinating processes : An Automata Theoretic Approach.* Princeton University Press, 1994.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specifications. In *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 97–107. ACM, ACM, January 1985.

[McM93] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[TBK91] H. J. Touati, R. K. Brayton, and R. P. Kurshan. Testing language containment for omega-automata using BDD'S. In *Proceedings of The 1991 International Workshop on Formal Methods in VLSI Design*, January 1991.

[TBK95] H. J. Touati, R. K. Brayton, and R. P. Kurshan. Testing language containment for $\omega$-automata using BDD's. *Information and Computation*, 118(1):101–109, April 1995.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.