Analysis of Symbolic SCC Hull Algorithms*

Fabio Somenzi[1], Kavita Ravi[2], and Roderick Bloem[3]**

[1] University of Colorado at Boulder
Fabio@Colorado.EDU
[2] Cadence Design Systems
kravi@cadence.com
[3] Graz University of Technology
Roderick.Bloem@tugraz.at

Abstract. The Generalized SCC Hull (GSH) algorithm of [11] can be instantiated to obtain many symbolic algorithms for the detection of fair cycles in a graph. We present a modified GSH with improved convergence properties, and we use it to study—both in theory and experimentally—the performance of various algorithms employed in symbolic model checkers. In particular, we show that the algorithm of Emerson and Lei [4] has optimal complexity among those that can be derived from GSH. We also propose an early termination check that allows the Lockstep algorithm [1] to detect the existence of a fair cycle before an entire SCC has been examined. Our experimental evaluation confirms that no one method dominates the others, and identifies some of the factors that impact run times besides those accounted for by the theoretical analysis.

1 Introduction

Cycle detection algorithms are at the heart of symbolic model checkers. For common specification mechanisms like fair CTL, LTL, and ω-automata, deciding the satisfaction of a property entails determining whether an infinite path can be found along which certain fairness constraints are satisfied infinitely often [3,9]. The existence of a fair cycle—one that goes through a set of states intersecting all fairness constraints—provides an affirmative answer.

Given its central role, fair cycle detection has received considerable attention over the years. In recent times, both new algorithms [15,1] based on SCC enumeration, and variants of the classical algorithm of Emerson and Lei [4] that computes a hull of all fair SCCs [7,6,8,5] have been proposed for symbolic model checking [10]. A first contribution to the classification and comparison of these different algorithms was offered in [11]. There, it was shown that most algorithms that compute hulls of the fair SCCs can be regarded as instantiations of a Generic SCC Hull (GSH) algorithm. In this paper, we improve the GSH algorithm and we use it to derive several bounds on the performance of classes of instantiations of GSH. These classes subsume all popular SCC-hull algorithms. Among our results, we prove that the algorithm of Emerson and Lei

* This work was supported in part by SRC contract 2001-TJ-920 and NSF grant CCR-99-71195.
** This work was done while this author was with the University of Colorado at Boulder.

M.D. Aagaard and J.W. O'Leary (Eds.): FMCAD 2002, LNCS 2517, pp. 88–105, 2002.

is optimal in terms of worst-case number of image and preimage computations among the SCC-hull algorithms.

Besides providing an excellent framework for the analysis of SCC-hull algorithms, GSH is flexible and efficient in practice. Our implementation of the Emerson and Lei algorithm as an instantiation of GSH is actually slightly more efficient than the traditional one it replaced. We used the new implementation of GSH to compare several different SCC hull algorithms, and to compare them to a modified Lockstep [1] algorithm, enhanced by an early termination check that allows it to detect the existence of a fair cycle even before an entire SCC has been examined.

Our experiments confirm in essence the observations of [11,5] that there is no clear winner among the competing algorithms. They also shed some light on what factors affect the relative performance of different techniques, and point to directions for possible improvement.

The paper by Fisler et al. [5] also compares different SCC hull algorithms with respect to their complexity. Its bounds are in terms of the number of nodes of the graph. For the large graphs encountered in symbolic model checking, we argue that our analysis, which is in terms of quantities like the diameter of the graph and the height of the SCC quotient graph, provides more useful bounds. On the other hand, we do not discuss maximum gaps between algorithms.

2 Preliminaries

Symbolic graph algorithms operate on the characteristic functions of sets. Besides the usual Boolean connectives, they employ as basic operations the computation of the predecessors (EX) and successors (EY) of a set of states[1]. From these operators, others can be derived; in particular, we shall use:

$$\begin{aligned} \mathsf{E}\, q\, \mathsf{U}\, p &= \mu Z \,.\, p \vee (q \wedge \mathsf{EX}\, Z) & \mathsf{E}\, q\, \mathsf{S}\, p &= \mu Z \,.\, p \vee (q \wedge \mathsf{EY}\, Z) \\ \mathsf{EG}\, p &= \nu Z \,.\, p \wedge \mathsf{EX}\, Z & \mathsf{EH}\, p &= \nu Z \,.\, p \wedge \mathsf{EY}\, Z \\ \mathsf{EF}\, p &= \mathsf{E}\, \mathsf{true}\, \mathsf{U}\, p & \mathsf{EP}\, p &= \mathsf{E}\, \mathsf{true}\, \mathsf{S}\, p \ , \end{aligned}$$

where $\mu Z \,.\, \tau$ denotes the least fixpoint of τ and $\nu Z \,.\, \tau$ denotes its greatest fixpoint.

Given a graph $G(V, E)$ and a set $C = \{c_0, \ldots, c_{m-1}\} \subseteq 2^V$ of (Büchi) fairness constraints, a *fair cycle* is a cycle that intersects each c_i. Several algorithms have been proposed that check for the existence of a fair cycle by computing an *SCC hull* [11]; that is, a set of states that is empty if no fair cycle exists in G, and otherwise contains all fair SCCs. A *fair SCC* is a maximal nontrivial strongly connected component of the graph that intersects all fairness constraints, and hence contains a fair cycle. (A trivial SCC has one node and no arcs).

A set of vertices is *SCC-closed* if every SCC is either contained in it or disjoint from it. The *SCC quotient graph* of G is obtained by collapsing each SCC to a single node. This graph is acyclic, and defines a partial order of the SCCs. The *transitive (irreflexive)*

[1] Throughout this paper, the interpretation of a temporal logic formula is the characteristic function of a set of states. We do not distinguish a set from its characteristic function.

closure of a graph is obtained by adding arcs from a node to all nodes to which it has paths of length greater than 1.

Let L be a finite lattice and $l, l' \in L$ generic elements of L. A function $\tau : L \to L$ is *monotonic* if $l \leq l'$ implies $\tau(l) \leq \tau(l')$; τ is *downward* if $\tau(l) \leq l$. A monotonic function over a finite lattice has both a least and a greatest fixpoint [13].

Given a family $T = \{\tau_1, \ldots, \tau_k\}$ of monotonic functions over L, we are interested in their greatest common fixpoint. We summarize here the results from [12] that we need. For σ a finite sequence over T, let τ_σ be the function $L \to L$ obtained by composing all the functions in σ in the order specified by the sequence. We say that σ is *1-closed* if for $i = 1, \ldots, k$ we have $\tau_i(\tau_\sigma(1)) = \tau_\sigma(1)$, in other words, if $\tau_\sigma(1)$ is a common fixpoint of $\tau_1, \ldots, \tau_k$. Let σ be a 1-closed sequence and $\Gamma = \bigcap_{\tau \in T} \tau$ be the pointwise greatest lower bound of all the functions in T. Then

$$\tau_\sigma(1) = \nu Z \,.\, \Gamma(Z) \ . \tag{1}$$

From this result it is easily seen that τ_σ is the greatest common fixpoint of $\tau_1, \ldots, \tau_n$. An infinite sequence over T is *fair* if each $\tau \in T$ appears in the sequence infinitely often. If the functions in T are downward, then all fair sequences have a finite prefix that is 1-closed. Also, for a 1-closed sequence σ, $\tau_\sigma(1) \leq l$, then $\tau_\sigma(l) = \tau_\sigma(1)$. The results presented above imply that we can compute this common fixpoint by iteratively applying all functions, as long as the sequence is fair, i.e., as long as we keep revisiting every function.

A set of states X is *forward-closed* [11] if $X = \mathsf{EP}\, X$; X is *backward-closed* if $X = \mathsf{EF}\, X$. If X is forward-closed, then $X \geq \mathsf{EY}\, X$ and $\mathsf{EY}\, X$ is forward-closed. If X is backward-closed, then $X \geq \mathsf{EX}\, X$ and $\mathsf{EX}\, X$ is backward-closed.

3 The Generic SCC Hull Algorithm

In this section, we shall describe the Generic SCC Hull algorithm. We shall first describe how an SCC hull can be computed. Then, we shall introduce the algorithm and constraints on the schedule so that only potentially useful operations can be chosen. Finally, we shall look at existing algorithms as GSH schedules.

3.1 Computing an SCC Hull

Let $G = (V, E)$ be a graph and $C = \{c_0, \ldots, c_{m-1}\} \subseteq 2^S$ a set of Büchi fairness constraints. Let

$$T_P = \{\mathsf{ES}_0, \ldots, \mathsf{ES}_{m-1}, \lambda Z.\, Z \wedge \mathsf{EY}\, Z\}$$

be a set of monotonic downward past-tense (or forward) operators over S, where ES_i stands for $\lambda Z.\ \mathsf{E}\, Z\, \mathsf{S}(Z \wedge c_i)$. Likewise,

$$T_F = \{\mathsf{EU}_0, \ldots, \mathsf{EU}_{m-1}, \lambda Z.\, Z \wedge \mathsf{EX}\, Z\}$$

is a set of monotonic downward future-tense (or backward) operators with $\mathsf{EU}_i = \lambda Z.\, \mathsf{E}\, Z\, \mathsf{U}(Z \wedge c_i)$. Let $T_B = T_P \cup T_F$. (P stands for "Past," F stands for "Future," and B stands for "Both.")

Theorem 1. *Let ϕ, π, and β be finite 1-closed sequences over T_F, T_P, and T_B, respectively. Let F (P) be the set of nodes of G with a path to (from) a cycle fair with respect to C. Let $B = F \cap P$. Then*

$$\tau_\beta(1) = B \ , \qquad \tau_\phi(1) = F \ , \qquad \textit{and} \qquad \tau_\pi(1) = P \ .$$

Proof. We prove that $\tau_\phi(1) = F$. The proofs for $\tau_\pi(1) = P$ and $\tau_\beta(1) = B$ are similar.

It follows from the results in Section 2 that $\tau_\phi(1)$ is the greatest common fixpoint of the functions in T_F. We shall now prove that F has the same property.

We shall first prove that F is a common fixpoint. It is clear that $F \supseteq \mathsf{E}\, F\, \mathsf{U}(F \wedge c_i)$. For the other direction, suppose $v \in F$. Then v has a path to a fair cycle, and this path and the cycle are by definition contained within F. Hence, there is a path from v to a node in c_i that is wholly contained within F.

It is also clear that $F \supseteq F \wedge \mathsf{EX}\, F$. For the other direction, note that if a node has a path to a fair cycle, then it has a successor with a path to a fair cycle.

We shall now prove that F is the greatest common fixpoint of T_F. Suppose F' is a common fixpoint of T_F, and let $v \in F'$. From $F' = F' \wedge \mathsf{EX}\, F'$ it follows that v has a successor $v' \in F'$. This successor has a path to some $v_0 \in c_0 \wedge F'$ because $F' = \mathsf{E}\, F'\, \mathsf{U}(F' \wedge c_0)$.

The path from v to v_0 is a non-trivial path connecting v to a state in v_0. With similar process, we can extend this path contained in F' so that it goes through states $v_1, \ldots, v_i, \ldots$ with $v_i \in v_{i \bmod m}$. Since F' is finite, it is possible to extract from this path a cycle that touches all fairness constraints. Hence, v can reach a fair cycle, and thus $F' \subseteq F$. □

As a result of this theorem, and the fact that all functions in T_F are downward, any fair sequence of functions from T_F has a finite prefix ϕ that computes the set of states that can reach a fair cycle. We can compute this set by applying all functions until we have reached a common fixpoint. In a similar manner, we can compute P or B.

In our applications, we need to detect the existence of a fair cycle, i.e., we can stop as soon as we know that F, P or B are empty; for counterexample generation we need to detect the cycle, by either isolating the fair SCC or knowing whether it is minimal or maximal. The following corollary allows us to halt the computation of B as soon as we have reached either a common fixpoint of T_F, or a common fixpoint of T_P.

Corollary 1. *If σ is a sequence over T_B that is 1-closed with respect to T_F (that is, $\tau(\tau_\sigma(1)) = \tau_\sigma(1)$ for all $\tau \in T_F$), then $B \leq \tau_\sigma(1) \leq F$. If σ is 1-closed with respect to T_P, then $B \leq \tau_\sigma(1) \leq P$.*

Proof. We consider only the case in which σ is 1-closed with respect to T_F. The other case is proved similarly. Since closedness with respect to T_B implies closedness with respect to T_F, if β is a 1-closed sequence over T_B, we have

$$B = \tau_\beta(1) \leq \tau_\sigma(1) \ .$$

Let ϕ be a 1-closed sequence over T_F. Then

$$\tau_\sigma(1) = \tau_{(\sigma,\phi)}(1) \leq \tau_\phi(1) = F \ ,$$

where the first equality comes from the closedness of σ with respect to T_F. □

In the proof of Theorem 1, we have assumed that all functions were downward. In practice, for schedules that apply past-tense operators only, we can use $\lambda Z.\ \mathsf{EY}\ Z$ instead of $\lambda Z.\ Z \wedge \mathsf{EY}\ Z$ because $\mathsf{EP}\ S_0$ is forward-closed. For future-tense schedules, we can use $\lambda Z.\ \mathsf{EX}\ Z$ instead of $\lambda Z.\ Z \wedge \mathsf{EX}\ Z$, because Z restricted to the reachable states is backward-closed. Therefore, the simplified operators are *contextually* downward. In the following, we assume that contextual downwardness is exploited when possible, and we denote by EX and EY the simplest forms of the operators allowed by the schedule.

3.2 Limiting the Choice of the Operator

Corollary 1 leads to the Generalized SCC Hull (GSH) algorithm of Fig. 1 (which improves the one in [11]). The algorithm functions by keeping a fixpoint iterate Z (and its previous value ζ). It iteratively picks and applies an operator τ. We say the operator makes *progress* if it changes Z. The algorithm keeps a list of disabled functions γ. Function PICK is responsible for the scheduling. Note that a schedule need not be fixed in advance. The function CONVERGED ensures convergence, by disabling operators that can not make progress. Its decisions depend on whether Z changes, the last operator, and the operators that will not yield progress; they are discussed below.

Theorem 2. *If a past-tense (future-tense) operator is an* ES *(*EU*) or makes no progress, then it can not make progress when applied again unless in the meanwhile a different past-tense (future-tense) operator has made progress.*

Proof. We shall prove the theorem for past-tense operators. The case for future-tense operators is dual. Suppose EY causes no progress. Then all the minimal SCCs are nontrivial. Application of EX may not remove any of them. Application of EU_i may remove some of them. Let s be a minimal nontrivial SCC that is removed by EU_i. Then s is not fair, and it cannot reach the i-th fair set. Hence, EU_i removes all the successors of s together with s. Therefore, no trivial SCC becomes minimal as a result of the application of EU_i. Thus, EY will not cause progress if applied again.

Suppose ES_i is applied. Then all the minimal SCCs of the resulting graph intersect the i-th fair set. As above, if EU_j ($j \neq i$) removes s, then it removes also all its successors. Hence, all minimal SCCs continue to intersect the i-th fair set. Likewise, (repeated) application of EX may eliminate s, but since EX eliminates terminal SCCs, when s is eliminated it has no successors and no new minimal SCCs are thus created. Hence, applying ES_i again will not cause progress.

We have shown that if a past-tense operator τ is an ES or fails to make progress, no intervening future-tense operator can change the graph in such a way that application of τ will make progress. □

Theorem 3. *For a given sequence of operators that GSH may pick, there is a graph G such that all operators that are not in γ at the j-th iteration of the sequence make progress in G restricted to the iterate Z at iteration j.*

Proof (sketch). The main observation is that the operators may only remove minimal or maximal SCCs. Given a sequence of pairs—each pair consisting of an operator and a Boolean denoting whether the operator made progress—one can construct a graph G that satisfies the following two properties, and their counterparts for future-tense operators.

```
global T_F, T_P

GSH(G, S_0) { // graph, initial states
    Z := EP S_0; // fixpoint iterate
    γ := ∅; // disabled operator set
    do {
        ζ := Z;
        τ := PICK((T_F ∪ T_P) \ γ);
        Z := τ(Z);
        (done, γ) := CONVERGED((Z ≠ ζ), τ, γ);
    } until (done);
    return Z;
}

CONVERGED(progress, τ, γ) {
    if (progress) {
        // enable operators of the same tense as τ ...
        if (τ ∈ T_F) γ := γ \ T_F; else γ := γ \ T_P;
        // ... except τ itself if it is an EU or an ES
        if (τ ∉ {EX, EY}) γ := γ ∪ {τ};
        return (false,γ);
    } else { // no progress, disable τ
        γ := γ ∪ {τ};
        return ((T_F ⊆ γ ∨ T_P ⊆ γ) , γ);
    }
}
```

Fig. 1. Improved GSH algorithm

1. If the j-th operator made progress, then the subgraph of G defined by Z_j has a minimal trivial SCC, so that EY will make progress if applied.
2. If there has been a successful application of a past-tense operator since the last ES_i, then the subgraph of G defined by Z_j has a minimal SCC that has no intersection with c_i, so that ES_i will make progress if applied.

The graph can be constructed inductively as a linear graph with some self loops, by starting with a graph consisting of a single node, and working back through the sequence of operations, making the graph consistent with the observations. □

Theorem 2 leads to the improved convergence test of Fig. 1, while Theorem 3 proves that it is optimal in the sense that if decisions are only based on the picked operator and whether it made progress, the set of disabled operations could never be any larger.

Theorem 2 also allows us to improve algorithm that use both tenses [7,6] as follows. If during a pass no progress is obtained by application of the past-tense (future-tense) operators, these operators are no longer applied. This improvement is for the case in which we insist on having both initial and terminal SCCs fair. Otherwise, the criterion of Fig. 1 suffices. Theorems 2 and 2 are not exploited in the original GSH [11].

Theorem 4. *The* GSH *algorithm returns the empty set if no fair cycle exists. If a fair cycle exists, it returns a set containing all states on a fair cycle.*

Proof. The algorithm terminates because of the result in Section 2 that any fair sequence has a finite 1-closed prefix, and because CONVERGED does not disable too many operations, as proven in Theorem 2. Correctness follows from Theorem 1. □

It should be noted that either the minimal SCCs or the maximal SCCs of a non-empty hull produced by GSH are fair [11].

3.3 Existing Algorithms as GSH Schedules

Efficient version of popular SCC-hull algorithms can be described as specializations of GSH by giving their schedules. In particular:

$$\begin{aligned}
\text{EL [4]} &: \mathsf{EU}_0, \mathsf{EX}, \ldots, \mathsf{EU}_{m-1}, \mathsf{EX}, \mathsf{EU}_0, \mathsf{EX}, \ldots \\
\text{EL2 [7,5]} &: \mathsf{EU}_0, \mathsf{EU}_1, \ldots, \mathsf{EU}_{m-1}, \mathsf{EG}, \mathsf{EU}_0, \mathsf{EU}_1, \ldots \\
\text{HH [7,6]} &: \mathsf{EU}_0, \mathsf{ES}_0, \ldots, \mathsf{EU}_{n-1}, \mathsf{ES}_{m-1}, \mathsf{EX}, \mathsf{EY}, \mathsf{EX}\mathsf{EY}, \ldots, \\
&\quad \mathsf{EU}_0, \mathsf{ES}_0, \ldots
\end{aligned}$$

Here, EG represents applying EX to convergence, which implies that the EL2 schedule is not fixed in advance, but rather depends on the termination of EG.

Note that the termination condition of GSH is more refined than that of the algorithms above. In particular, even with the EL schedule, GSH may detect convergence earlier than the standard EL algorithm, as shown in the following example.

Example 1. Suppose we are using algorithm EL to compute the set of states reaching a cycle intersecting a single Büchi fairness constraint F. Let Z be the fixpoint iterate. We start with Z_0 equal to the set of all reachable states and compute

$$\begin{aligned}
Z_1 &= \mathsf{E}\, Z_0\, \mathsf{U}(Z_0 \wedge F) \\
Z_2 &= Z_1 \wedge \mathsf{EX}\, Z_1 \ .
\end{aligned}$$

Suppose $Z_1 \neq Z_0$, and $Z_2 = Z_1$. EL will perform another iteration because $Z_2 \neq Z_0$, computing

$$\begin{aligned}
Z_3 &= \mathsf{E}\, Z_2\, \mathsf{U}(Z_2 \wedge F) \\
Z_4 &= Z_3 \wedge \mathsf{EX}\, Z_3 \ .
\end{aligned}$$

If GSH picks operators so as to mimic EL, it also computes Z_1 and Z_2, but then it stops, because all future tense operators are disabled. Since Z_1 is a fixpoint,

$$Z_1 = \mathsf{E}\, Z_1\, \mathsf{U}(Z_1 \wedge F) \ ,$$

and from $Z_2 = Z_1$ it follows that Z_2 is the set of states that can reach a fair cycle. □

Accelerated convergence is obviously desirable, and in practice implementing EL as a GSH schedule improves performance slightly. The main virtue of the GSH approach, however, lies in providing a common framework for the implementation and study of SCC-hull algorithms, as shown in the next section.

Table 1. Labels for nodes

label	meaning
0	has self loop, belongs to no fairness constraints
$n > 0$	has self loop, belongs to all fairness constraints except the nth
x	trivial SCC, belongs to no fairness constraints
o	trivial SCC, belongs to all fairness constraints
F	has self loop, belongs to all fairness constraints

4 Complexity of GSH

Following [1], we measure the cost of GSH in *steps*, the number of EX and EY operations applied to a nonempty set. We express the cost in terms of the number of fairness constraints $|C|$, the diameter of the graph d, the height of (i.e., the length of the longest paths in) the SCC quotient graph h, and the number of SCCs (N) and nontrivial SCCs (N'). Since d, h, and N are often much smaller than n, this analysis provides more practical bounds than using the number of states. In [1, Theorem 1] it was shown that EL takes $\Theta(|C|dh)$ steps. In this section we extend this result to account for the flexibility in the scheduling of operators that characterizes the GSH algorithm. Throughout the analysis, we shall analyze worst-case behavior: we pick graphs that are hard for the algorithms. In Sections 4.1 and 4.2, we shall look at how many steps are needed if the scheduler chooses operations badly. In Section 4.3, we shall study how many steps are needed if the scheduler makes optimal decisions.

The conventions used to label the nodes in Figs. 2–5 are shown in Table 1. To avoid clutter, the arcs controlling the diameter are not shown, but rather described in the captions.

4.1 Bounds for Unrestricted Schedules

Theorem 5. *GSH takes $O(|C|dN)$ steps.*

Proof. Let $t = |T_B| = 2|C| + 2$ be the number of operators applied by GSH. Clearly, $O(t) = O(|C|)$. We must have progress at least once every t iterations, because otherwise all operators have been applied without progress and the algorithm terminates.

Each operator's application cost is $O(d)$. Hence, we do $O(|C|d)$ work in between two advances toward the fixpoint. The number of times we make progress is $O(N)$. Hence, the desired bound.

To show that the number of times we make progress is $O(N)$ we argue as follows. The initial Z is SCC-closed, because it is either V or the reachable subset of V. When any of the operators in T_B is applied, SCC-closedness is preserved. In particular, this holds for EU and ES because Z is (inductively) SCC-closed. Indeed, if $v \in V$ has a path to $Z' \subseteq Z$ all in Z, then the SCC of v is contained in Z. Hence, each v' in the same SCC has a path to Z' all in Z. The result is therefore SCC-closed, and the set of dropped states, which is the difference of two SCC-closed sets, is also SCC-closed.

In summary, when there is progress, Z loses an integral number of unfair SCCs that is greater than or equal to 1. Thus, progress cannot occur more than N times. □

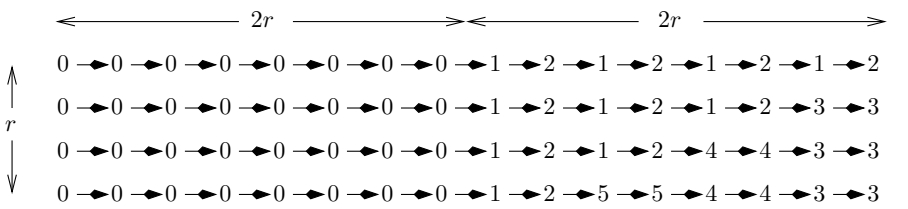

Fig. 2. Graph showing that GSH is $\Omega(|C|dN)$. Not shown are the arcs from any node with a label different from 0 to all the nodes to its right

The bound of Theorem 5 is tight in the following sense.

Theorem 6. *There exist a family of graphs and a corresponding family of schedules such that GSH takes $\Omega(|C|dN)$ steps.*

Proof. Consider the family of graphs parameterized by r that is exemplified by Fig. 2. A graph in the family has r rows, each of which consists of $4r$ nontrivial SCCs. Hence, there are $N = 4r^2$ SCCs, and $|C| = r + 1$ acceptance conditions; the height of the SCC graph is $4r$, and the diameter is $d = 2r + 1$.

Let $U = \{\mathsf{EU}_i \,|1 \leq i \leq |C|\}$. We consider the following schedule.

- All elements of $U \setminus \{\mathsf{EU}_3\}$ in decreasing index order r times, followed by EU_3.
- All elements of $U \setminus \{\mathsf{EU}_4\}$ in decreasing index order $r - 1$ times, followed by EU_4.
- ...
- All elements of $U \setminus \{\mathsf{EU}_{|C|}\}$ in decreasing index order twice, followed by $\mathsf{EU}_{|C|}$.
- All elements of $U \setminus \{\mathsf{EU}_{|C|}\}$ in decreasing index order once.

We now count the steps. The first series of subsequences takes $h/4O(|C|d)$ steps. The second series takes $(h/4 - 1)O(|C|d)$ steps and so on. The total number of steps is therefore $\Omega(|C|dh^2)$, which is also $\Omega(|C|dN)$. □

4.2 Bounds for Restricted Schedules

If we strengthen the assumption about PICK, we can prove an $O(|C|dh + N - N')$ bound. ($N - N'$ is the number of trivial SCCs.) The additional assumption is that the computation is performed in passes. We shall show that this bound is tight for EL2, but not for EL.

Definition 1. *A* pass *is a sequence over T_B that satisfies the constraints imposed by GSH, and such that:*

1. *No EU_i or ES_i appears more than once.*
2. *Either all operators in T_F or all operators in T_P appear.*

Having thus divided the computation in passes, we can use reasoning similar to the one of [1, Theorem 1].

Table 2. Schedules and tenses. The algorithms are classified according to the mix of operators (EX of EY and EU or ES). Within these categories, they differ by tense

	EL	EL2
future-tense	[4]	[7,5]
past-tense		[7,8]
both tenses		[7,6]

Theorem 7. *If the operator schedule can be divided in passes, GSH takes* $O(|C|dh + N - N')$ *steps.*

Proof. A pass in which all EU operators and at least one EX have been applied once removes all the terminal unfair SCCs present at the beginning of the pass. Likewise, a pass in which all ES operators and at least one EY have been applied once removes all the minimal unfair SCCs present at the beginning of the pass. Then, by induction on h, we can prove that we cannot have more than h passes of either type, for a total of $2h$ passes.

Each pass may contain more than one EX or EY. We charge their cost separately, and we argue that the total cost of the successful applications is $O(N - N')$, because each extra EXs or EYs removes a trivial SCC. The cost of the unsuccessful applications is dominated by the cost of the EUs and ESs, which is $O(|C|d)$. □

The algorithms of Table 2 all satisfy the restricted scheduling policy[2], and are therefore $O(|C|dh + N - N')$. $N - N'$ is the linear penalty discussed in [5]. Though this penalty does not alter the complexity bounds in terms of the number of states n, it cannot be ignored when the bounds are given in terms of $|C|$, d, and h.

Consider the following family $G_{r,s,f}$ of graphs. Here, r is the number of rows, $0 < s < 2r$ determines the diameter, and f is the number of fairness conditions. (Shown in Fig. 3 is $G_{3,2,2}$.) For this family of graphs, $d = s + 2$, $|C| = f$, and $h = 4r - 1$.

We consider the EL2 schedule. The future-tense version of EL2 applies EU_1 through EU_f followed by EG until convergence.

The first application of EU_1 through EU_f removes the f rightmost nontrivial SCCs of each row. The successive EG removes what is left of the first row, and the rightmost trivial SCC of all the other rows. The second round of EU's removes again the f rightmost nontrivial SCCs of each surviving row. EG then removes the second row entirely and the rightmost trivial SCC of each other row.

We need a total of r passes to converge. Each pass costs $|C|(s + 3) + 2r + 2$. The $|C|(s + 3)$ term is for the EUs and the $2r + 2$ term is for the EG. Hence, the total cost of EL2 is $(|C|(s + 3) + 2r + 2)r$ which is not $O(|C|dh) = O(|C|sr)$.

So, even though EL2 may beat EL on specific examples, EL's $O(|C|dh)$ bound is better. For EL2 we have the following lower bound, which is a special case of our previous observation about schedules that can be divided into passes.

[2] GSH can implement a simplification of Kesten's algorithm that disregards the issues of Streett emptiness.

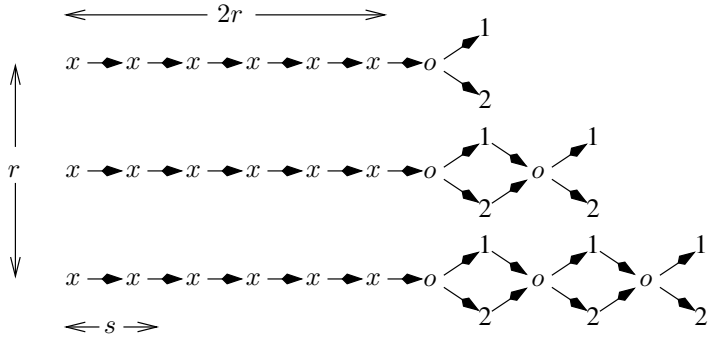

Fig. 3. Graph $G_{3,2,2}$ showing that EL2 is $\Theta(|C|dh + N - N')$. Not shown are the arcs from each node of type o or n to every node to its right on the same row, and from every node of type x, but the first s, to each x to its right and to the first o node on the row

Theorem 8. *Algorithm GSH with schedule EL2 runs in $\Theta(|C|dh + N - N')$ steps.*

Proof. EL2 is $O(|C|dh+N-N')$ thanks to Theorem 7. To show that EL2 is $\Omega(|C|dh+N-N')$ we resort to the family of graphs $G_{r,s,f}$ we have used to show that EL2 is not $O(|C|dh)$. We counted $(|C|(s+3)+2r)r$ steps for EL2. Since $N-N' = 5r^2/2+r/2$, $|C|dh + N - N' = |C|(s+2)(4r-1) + 5r^2/2 + r/2$, so $(|C|(s+3)+2r)r$ is $\Omega(|C|dh + N - N')$. □

A similar analysis can be carried out for the variant of EL2 that uses both tenses (HH). In particular, for the upper bound Theorem 7 applies. For the lower bound, one can take the family of graphs we have used for EL2, add a fair SCC at the beginning of each row, and then mirror each row to the left. On the other hand, every schedule divided in passes in which the cost of applying EX and EY in each pass is dominated by the cost of applying EU and ES operators shares the (optimal) bounds of EL.

4.3 Bounds for Optimal Schedules

Theorem 6 is concerned with how badly things can go if the schedule is not well matched to the graph at hand. It is also interesting to consider what an optimal schedule can do. To this purpose, we provide GSH with an oracle that computes an optimal schedule, and we call the resulting method OSH.

Theorem 9. *OSH takes $\Theta(|C|dh)$ steps.*

Proof. For the upper bound we rely on [1, Theorem 1], which shows that EL is $O(|C|dh)$. For the lower bound, we use the example of Fig. 4. In the graph shown, $|C| = 3$. The diameter is determined by the number of x nodes. Assume there are at least as many o nodes as there are x nodes.[3]

[3] This assumption guarantees that the number of o nodes, which determines the number of "rounds" of EUs or ESs, is $\Omega(h)$.

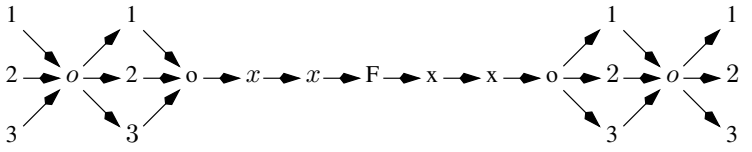

Fig. 4. Graph showing that OSH is $\Omega(|C|dh)$. The o and n nodes have arcs forward to the other o and n nodes on the same side of F

OSH takes $\Omega(|C|dh)$ steps on this family of graphs. The cost of an EU or ES does not change until some x nodes are removed. At that point, the optimal schedule simply removes the remaining exposed x nodes to reach convergence. Hence, in this case, a unidirectional schedule is optimal. Suppose we use a future-tense schedule to fix ideas.

Initially, we can only make progress by applying one EU. After that, we need to apply all remaining EUs before the rightmost o is exposed. At that point we can only make progress by applying EX. Therefore, we need to apply all EUs and one EX $\Omega(h)$ times. (Here is where we use the assumption of the number of o nodes.) The number of EUs is thus $\Omega(|C|h)$ and their cost is $\Omega(|C|dh)$. □

A consequence of Theorem 9 is that EL is an optimal schedule. This optimality has its limits: it depends on our choice of measures, and there are graphs on which other schedules need fewer steps.

Corollary 2. *If cost is measured in terms of steps, and expressed in terms of $|C|$, d, h, N, and N', there is no schedule of GSH that has better worst-case asymptotic complexity than EL.*

4.4 Bidirectional vs. Unidirectional Schedules

We conclude our analysis of GSH with a discussion of the advantages and disadvantages of schedules that use all four types of operators relative to those schedules that use only past-tense operators, or only future-tense operators.

The proof of Theorem 7 suggests that trimming from both sides may take more steps than trimming from one side only, because if we work only from one side, we need at most h passes instead of $2h$. Occasionally, though, working from both sides will speed up things, especially when there are no fair SCCs. (As noted in [6].) One reason is that a search in one direction can reduce the diameter of the graph, which helps the search in the other direction. The following example illustrates this point.

Example 2. Consider the family of graphs exemplified in Fig. 5. The arcs out of the x nodes are all to the direct neighbor to the right, while the remaining nodes form a complete acyclic subgraph. In the example graph, $d = 12$, $|C| = 3$.

OSH first applies an EH at a cost of $d/2$; it then applies $d/2$ EUs. Each EU costs 2 steps; so, the total cost is $\Theta(d)$ steps for EUs and $O(d)$ steps total.

Any purely future-tense or purely past-tense algorithm needs to apply $d/2$ EUs, too, but this time every one costs $\Omega(d)$ steps, giving a quadratic behavior. Note that an EG does not help. □

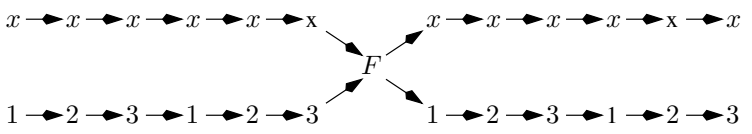

Fig. 5. Graph illustrating the possible advantages of bidirectional schedules

The preceding example shows that some bidirectional schedules may outperform all unidirectional ones. Obviously, there are even more cases in which bidirectional schedules outperform schedules in one direction, but not in the other.

5 Early Termination in Lockstep

Lockstep is a symbolic cycle detection algorithm based on SCC enumeration [1]. Given a seed v, the SCC containing v is computed as the intersection of the set $\beta(v)$ of states that can reach v nontrivially and the set $\phi(v)$ of states that are nontrivially reachable from v. If there are short cycles involving the seed state v, the intersection of $\beta(v)$ and $\phi(v)$ may be non-empty well before the two sets have been computed in their entirety. This suggests an early termination criterion for the algorithm.

Theorem 10. *Let $F(v)$ and $B(v)$ be the subsets of $\phi(v)$ and $\beta(v)$ computed by Lockstep at some iteration. Let $I(v) = F(v) \cap B(v)$ and $U(v) = F(v) \cup B(v)$. If $I(v)$ has a non-null intersection with all fair sets, then $U(v)$ contains a fair cycle.*

Proof. Every state in $B(v)$ has a non-trivial path to v entirely contained in $B(v)$. Every state in $F(v)$ has a non-trivial path from v entirely contained in $F(v)$. Hence, every state in $I(v)$ has non-trivial paths to and from v entirely contained in $U(v)$. Therefore, every state in $I(v)$ is connected to every other state of $I(v)$ by a non-trivial path entirely in $U(v)$. Since $I(v)$ contains representatives from all fair sets, one can trace a fair cycle in $U(v)$. □

Once a fair set intersects $I(v)$, it will continue to intersect it in all successive iterations of lockstep. Furthermore, at each iteration, one can stop testing intersections as soon as one fair set is found that does not intersect $I(v)$. Hence, if there are $|C|$ fair sets and convergence requires s steps, the number of intersection checks is $O(|C| + s)$. The overhead for this early termination check is $O(s)$ intersections (for $I(v)$) and $O(s)$ intersection checks, because the original Lockstep performs $O(|C|)$ intersection checks on the maximal SCC.

Early termination imposes a simple change to counterexample generation: The fair sets are intersected with $I(v)$. This intersection guarantees that the path connecting the fair sets can always be closed.

6 Experiments

In this section we present preliminary results obtained with implementations of GSH and Lockstep in VIS 1.4 [2]. The CPU times were measured on an IBM IntelliStation running Linux with a 1.7 GHz Pentium 4 CPU and 1 GB of RAM.

The experiments involved three types of schedules for GSH: EL, EL2, and a random schedule, which applies about the same fraction of EXs as EL. We experimented with different levels in the use of don't care conditions in the computation of the fixpoints. Our experiments involved both language emptiness checks, and CTL model checking problems. For the language emptiness experiments, we considered both future-tense and past-tense schedules, and we also ran the enhanced version of Lockstep described in Section 5.

All experiments included reachability analysis and used fixed BDD variable orders to minimize noise. We present three summary tables, for the three classes of experiments. The parameters we vary are the algorithm/schedule, the tense (future or past) for GSH schedules, and the degree of exploitation of don't care conditions in the EX and EY computations.

For all GSH schedules, a "low DC level" means that the reachable states are used to simplify the transition relation, that the fixpoint computation for the fair states is started from $Z = \mathsf{true}$, and that no frontiers are used in the EU and ES computations. A "medium DC level" means that in addition to simplifying the transition relation, frontiers are used. A "high DC level," would simplify the argument to each EX computation with respect to the reachable states. (This simplification is not possible for EY computations.) One such scheme that we tried did not produce significant improvements over "medium DC level." We have not yet implemented the technique described in [14].

The columns of the tables have the following meaning. *Total* is the total time (in seconds) for all experiments. For timed-out experiments, the limit of 1800s is taken. *Gmt* is the geometric mean of the running time of all experiments. A *win* is a case in which a method is at least 2% and at least 0.1s faster than any other method. A *tie* is a case in which there was no win, and the method was either fastest or less than 2% slower than the fastest method. *T/O* is the number of experiments that timed out. *Steps* is the total number of steps (EXs and EYs) performed, and *gms* is the geometric mean of the number of steps. All experiments for which at least one method took 0s are excluded from the computation of geometric mean time, wins, and ties.

Table 3 compares different GSH schedules for CTL model checking experiments; Table 4 compares GSH schedules and Lockstep on language emptiness problems; and Table 5 shows the results of LTL and CTL model checking for families of parameterized models. The properties used for these experiments required cycle detection for both CTL and LTL model checking.

The data in the tables supports the following remarks.

- No type of schedule dominates the others, even though on individual models there are sometimes large differences. On average, EL-type schedules are the fastest for the parameterized models, while EL2 is the best for the non-parameterized ones.
- The complexity bounds of Section 4 are in terms of number of steps. While within a homogeneous group of experiments (same tense and DC level) the schedule performing fewer steps is often the fastest, it is obvious that the cost of a step is not

Table 3. Summary of CTL model checking experiments for 39 models

schedule	DC level	total	gmt	wins	ties	T/O	steps	gms
EL	low	10307	7.7	1	20	4	78734	139
EL2	low	8002	6.3	0	23	3	31153	126
random	low	10028	8.5	2	20	4	64420	148
EL	medium	9606	7.5	0	14	5	78622	137
EL2	medium	9128	6.7	1	17	4	31013	123
random	medium	11284	9.0	1	16	6	64059	147

Table 4. Summary of language emptiness experiments for 59 models

schedule	tense	DC level	total	gmt	wins	ties	T/O	steps	gms
EL	future	low	15389	23.1	0	13	7	37252	64
EL2	future	low	10273	16.4	0	13	4	15182	57
random	future	low	15167	22.7	0	13	6	31500	63
EL	future	medium	12367	19.6	1	13	4	38412	68
EL2	future	medium	8342	14.8	1	13	1	16033	59
random	future	medium	11875	18.9	1	14	2	32840	67
EL	past	low	11440	18.3	0	13	4	118711	101
EL2	past	low	5999	9.3	0	17	2	19894	71
random	past	low	10666	17.4	0	13	4	105842	98
EL	past	medium	6759	14.2	1	14	0	153188	105
EL2	past	medium	3362	8.4	3	16	0	20692	72
random	past	medium	5771	13.4	0	15	0	116904	101
lockstep	both	medium	10613	22.2	7	12	3	140673	86

constant. Figure 6, for instance, shows the relation between number of steps and CPU time for the EL schedule with low DC level applied to the parameterized families of models. It is readily seen that for most families, the computation time grows much faster than the number of steps. Also, the past-tense schedules of Table 5 perform many more steps than the corresponding future-tense schedules. However, the majority of them are due to just one model, and are very cheap.

– In our experiments, the tense did not affect in a significant way the comparison between different types of schedules (e.g., EL vs. EL2).
– Past-tense schedules did usually better than future-tense schedules.[4] However, the advantage of past-tense schedules may depend on several factors. These include different quantification schedules for EX and EY, different diameters for a graph and its reverse, the positions of the fair SCCs in the graphs, as well as various BDD-related factors like the fact that some fixed variable orders are saved at the end of reachability analysis runs with dynamic reordering, and the hard-to-predict effects of the BDD computed table. In addition, our current implementation applies the same don't care techniques for past and future schedules. All these reasons may

[4] In Table 5, the best results are for CTL model checking, but it is not possible to compare those future-tense schedules to the others because the LTL models have more state variables.

Table 5. Summary of model checking experiments for 57 models from 11 parameterized families

logic	schedule	tense	DC level	total	gmt	wins	ties	T/O	steps	gms
LTL	EL	future	low	7746	6.8	0	10	3	606502	587
LTL	EL2	future	low	13419	10.7	0	10	5	626093	1061
LTL	random	future	low	12614	10.5	0	8	5	685794	1053
LTL	EL	future	medium	7503	6.7	0	9	3	606400	587
LTL	EL2	future	medium	14009	10.5	0	8	7	595326	1007
LTL	random	future	medium	13896	10.3	0	8	7	656409	996
LTL	EL	past	low	8422	6.0	0	17	3	2873990	1220
LTL	EL2	past	low	8573	5.9	0	18	4	2749041	1164
LTL	random	past	low	7236	5.9	0	14	2	2865320	1237
LTL	EL	past	medium	8587	6.2	0	17	4	2816871	1188
LTL	EL2	past	medium	8588	6.0	0	18	4	2714631	1139
LTL	random	past	medium	7908	6.0	0	15	2	2857436	1222
LTL	lockstep	both	medium	26597	30.2	0	8	12	3362850	2603
CTL	EL	future	low	2699	2.8	7	26	1	396251	429
CTL	EL2	future	low	10076	4.9	4	23	5	370797	720
CTL	random	future	low	11525	4.9	0	23	6	483961	676
CTL	EL	future	medium	2969	3.2	0	29	1	396218	428
CTL	EL2	future	medium	10928	5.5	0	27	5	342588	703
CTL	random	future	medium	11725	5.5	0	27	6	463425	667

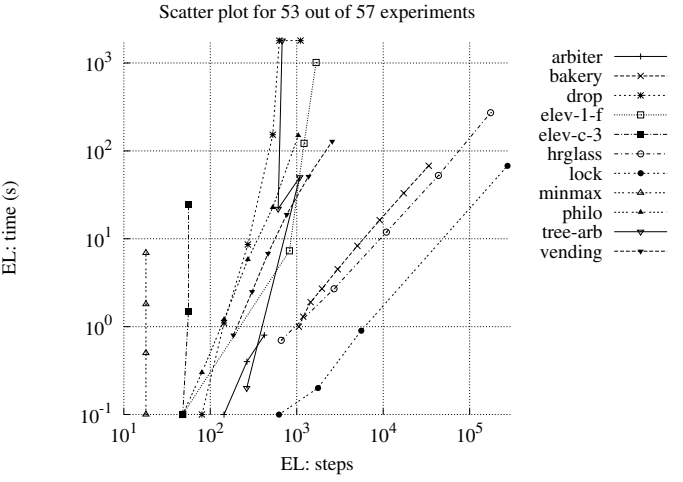

Fig. 6. CPU time as a function of the number of steps

explain the differences between our results and those of [11] with regard to tenses. It should also be mentioned that future tense schedules may be applied also without preliminary reachability analysis. For past-tense schedules, one has then to prove reachability of the fair SCCs.

– For all the experiments that complete within 1800 s, the number of steps does not depend on the DC level. However, in case of timeout, the number of steps until the timeout is counted. This explains small differences in the numbers of steps between methods that differ only in the use of don't cares.

Early termination for Lockstep is effective. The results with early termination are uniformly better or equal to those without.[5] Compared to GSH, Lockstep loses in most cases, but has the largest number of wins for both non-parameterized and parameterized language emptiness experiments. (That is, not counting the CTL experiments in Table 5.)

7 Conclusions

We have presented an improved Generic SCC Hull algorithm, and we have proved several bounds on the performance of classes of algorithms that can be cast as particular operator schedules for GSH. We have proved in particular, that when complexity is measured in steps (EX and EY computations) and it is given as a function of the number of fairness constraints $|C|$, the diameter of the graph d, the height of the SCC quotient graph h, and the number of total (nontrivial) SCCs N (N'), then algorithm EL is optimal ($\Theta(|C|dh)$) among those that can be simulated by GSH. Variants like EL2, on the other hand, are not optimal in that sense. (They are $\Theta(|C|dh + N - N')$.)

Of course, on a particular graph, EL2 may outperform EL for at least two reasons: On the one hand, the theoretical bounds are for worst-case performance. On the other hand, the cost of individual steps can vary widely. This implies that the theoretical analysis should be accompanied by an experimental evaluation.

We have performed such an assessment, conducting experiments with several competing algorithms on a large set of designs. We have found that no GSH schedule dominates the others. Also, Lockstep is slower on average than GSH, but it produces the best results in quite a few cases. On individual experiments the ranges of CPU times for the various schedules may cover three orders of magnitude, which suggests that having more than one method at one's disposal may allow more model checking problems to be solved.

References

[1] R. Bloem, H. N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 37–54. Springer-Verlag, November 2000. LNCS 1954.

[2] R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eighth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.

[3] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[5] We also found that the best performance is achieved by not trimming [11] the initial set of states (the reachable states). The results shown for Lockstep are for the algorithm that does not trim the initial set.

[4] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the First Annual Symposium of Logic in Computer Science*, pages 267–278, June 1986.

[5] K. Fisler, R. Fraer, G. Kamhi, M. Vardi, and Z. Yang. Is there a best symbolic cycle-detection algorithm? In T. Margaria and W. Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 420–434. Springer-Verlag, April 2001. LNCS 2031.

[6] R. H. Hardin, R. P. Kurshan, S. K. Shukla, and M. Y. Vardi. A new heuristic for bad cycle detection using BDDs. In O. Grumberg, editor, *Ninth Conference on Computer Aided Verification (CAV'97)*, pages 268–278. Springer-Verlag, Berlin, 1997. LNCS 1254.

[7] R. Hojati, H. Touati, R. P. Kurshan, and R. K. Brayton. Efficient ω-regular language containment. In *Computer Aided Verification*, pages 371–382, Montréal, Canada, June 1992.

[8] Y. Kesten, A. Pnueli, and L.-o. Raviv. Algorithmic verification of linear temporal logic specifications. In *International Colloquium on Automata, Languages, and Programming (ICALP-98)*, pages 1–16, Berlin, 1998. Springer. LNCS 1443.

[9] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, Princeton, NJ, 1994.

[10] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 1994.

[11] K. Ravi, R. Bloem, and F. Somenzi. A comparative study of symbolic algorithms for the computation of fair cycles. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 143–160. Springer-Verlag, November 2000. LNCS 1954.

[12] F. Somenzi. Symbolic state exploration. *Electronic Notes in Theoretical Computer Science*, 23, 1999. http://www.elsevier.nl/locate/entcs/volume23.html.

[13] A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[14] C. Wang, R. Bloem, G. D. Hachtel, K. Ravi, and F. Somenzi. Divide and compose: SCC refinement for language emptiness. In *International Conference on Concurrency Theory (CONCUR01)*, pages 456–471, Berlin, August 2001. Springer-Verlag. LNCS 2154.

[15] A. Xie and P. A. Beerel. Implicit enumeration of strongly connected components. In *Proceedings of the International Conference on Computer-Aided Design*, pages 37–40, San Jose, CA, November 1999.