An Algorithm for Strongly Connected Component Analysis in $n \log n$ Symbolic Steps*

Roderick Bloem, Harold N. Gabow, and Fabio Somenzi

University of Colorado at Boulder
{Roderick.Bloem@,hal@cs.,Fabio@}Colorado.EDU

Abstract. We present a symbolic algorithm for strongly connected component decomposition. The algorithm performs $\Theta(n \log n)$ image and preimage computations in the worst case, where n is the number of nodes in the graph. This is an improvement over the previously known quadratic bound. The algorithm can be used to decide emptiness of Büchi automata with the same complexity bound, improving Emerson and Lei's quadratic bound, and emptiness of Streett automata, with a similar bound in terms of nodes. It also leads to an improved procedure for the generation of nonemptiness witnesses.

1 Introduction

Tarjan's classical algorithm [Tar72] for finding strongly connected components (SCCs) is not feasible for very large graphs. Even though its worst-case complexity is linear, it has to consider and label each node individually: it is an *explicit* algorithm. For many graphs encountered in applications such as model checking, time and space do not suffice to consider individual nodes.

By contrast, symbolic graph algorithms deal with sets of nodes, represented by their characteristic functions. Although the worst-case time complexity of symbolic algorithms is typically worse than that of corresponding explicit algorithms, they perform well as heuristics, so that many large problems can only be tackled symbolically. The first use of symbolic graph algorithms known to these authors is due to McMillan [McM87], in the context of model checking [McM94]. As set representation he used Binary Decision Diagrams [Bry86], a compact, canonical form that allows for efficient set manipulation.

The most important operations in symbolic graph algorithms are the set-theoretic operations union, intersection, and complementation and, above all, *image* and *preimage*. The latter two are collectively referred to as *steps*. This paper uses the number of steps as the main measure of efficiency.

Although the number of steps does not determine the time-complexity of a symbolic algorithm, it is an important measure of difficulty. As noted in [HTKB92, HKSV97, BRS99], algorithms that take a quadratic number of steps, such as the Emerson-Lei algorithm for Büchi emptiness [EL86], often take much

* This work was supported in part by SRC contract 98-DJ-620 and NSF grant CCR-99-71195.

more time than algorithms that take a linear number of steps, such as the CTL model checking algorithm [CES86]. It is therefore important to keep the number of steps that an algorithm takes low.

The subject of this paper is a symbolic algorithm for strongly connected component decomposition with applications to the Büchi and Streett emptiness problems.

The first symbolic algorithm for SCC decomposition is due to Touati et al. [TBK95]. It computes the transitive closure of a transition relation using *iterative squaring*. This computation is expensive [HTKB92], and the algorithm is limited to rather simple graphs. A more practical symbolic algorithm for SCC decomposition has been presented by Xie and Beerel [XB99]. When measured in terms of nodes, this algorithm takes $\Theta(n^2)$ steps in the worst case. We present an algorithm that is similar to that of Xie and Beerel but takes $\Theta(n \log n)$ steps in the worst case.

Our algorithm depends critically on *lockstep search*, the idea of performing backward and forward searches in the graph simultaneously. Lockstep search is used in [ES81] to update the connected components in an undirected graph after an edge has been deleted, and in [HT96] to decide Streett emptiness. Although these papers contain a complexity analysis that is similar to ours, they use lockstep search in a different fashion: to recompute (strongly) connected components from the two nodes of an edge that has been deleted. Both algorithms are explicit, and the complexity bounds that Henzinger and Telle prove have little relation to our bounds.

An algorithm for SCC decomposition can be used to decide emptiness of a Büchi automaton: the language of an automaton is nonempty if there is a nontrivial SCC that contains an accepting state. Emptiness checks for Büchi automata are in common use in symbolic LTL model checkers such as McMillan's SMV and in fair-CTL model checkers such as SMV and VIS [McM94, McM99, B$^+$96]. The algorithm can easily be extended to deal with L-automata, which are used in Cospan [Kur94, HHK96].

All known symbolic algorithms for deciding Büchi emptiness, including the original Emerson-Lei algorithm [EL86, HKSV97, KPR98], take $\Theta(n^2)$ steps in the worst case. It has been an open problem whether this bound can be improved [HKSV97]. Our algorithm presents an improvement to $\Theta(n \log n)$.

In this paper we shall also show that the complexity of the Emerson-Lei algorithm can be bounded by $\Theta(|\mathcal{F}|dh)$, where $\mathcal{F}$ is the set of acceptance conditions, d is the diameter of the graph and h is the length of the longest path in the SCC quotient graph. The complexity of the algorithm presented here cannot be bounded as a function of only d and h, and is hence incomparable to that of the Emerson-Lei algorithm. We shall discuss problems on which it performs worse. Our algorithm, like that of Xie and Beerel, potentially enumerates each nontrivial SCC, which may be expensive if the ratio of nontrivial SCCs to nodes is high. In that sense it is "less symbolic" than the other algorithms for Büchi emptiness.

Emerson and Lei [EL87] and Kurshan [Kur94] propose explicit algorithms for deciding emptiness of Streett automata. We combine their approach with our SCC-decomposition algorithm to yield a symbolic algorithm for Streett emptiness that takes $\Theta(n(\log n + p))$ steps in the worst case, where p is the number of non-Büchi fairness conditions. This improves the $\Theta(pn^2)$ bound of the symbolic algorithm for Streett emptiness of Kesten et al. [KPR98]. We also show that the new algorithm leads to a procedure for the generation of nonemptiness witnesses that improves over the ones of [CGMZ95, HSBK93, KPR98].

In this paper, we focus on the algorithm and its asymptotic complexity. An analysis of its performance on practical verification examples and a comparison with other algorithms are given in [RBS].

The flow of the paper is as follows. In Section 2 we present the basic definitions. In Section 3 we present an analysis of the Emerson-Lei algorithm. In Section 4 we describe the Lockstep algorithm, whose complexity we analyze in Section 5. In Section 6 we present a simple optimization and its analysis. In Section 7 we discuss an application of Lockstep to the generation of witnesses, and we conclude with Section 8.

2 Preliminaries

In this section we discuss the basic notions pertaining to graphs, symbolic algorithms, and Streett and Büchi automata. In the following, $\lg x$ is the base 2 logarithm of x, and $\log x$ is the logarithm of x where the base is an unspecified constant greater than 1, with the stipulation that $\log x = 1$ for $x \leq 1$.

2.1 Graphs and Strongly Connected Components

A (directed) *graph* is a pair $G = (V, E)$, where V is a finite set of nodes and $E \subseteq V \times V$ is a set of edges. The number of nodes of the graph is denoted by $n(G)$. We say that $(v_0, \ldots, v_k) \in V^+$ is a *path* from v_0 to v_k of length k if for all i we have $(v_i, v_{i+1}) \in E$. A path is *nontrivial* if its length is not 0, and *simple* if all its nodes are distinct. The distance from v to w is the length of a shortest path from v to w, if one exists. The *diameter* $d(G)$ of a nonempty graph G is the largest distance between any two nodes between which the distance is defined. A *cycle* is a nontrivial path for which $v_0 = v_k$. The graph $H = (W, F)$ is an *induced subgraph* of G if $W \subseteq V$ and $F = E \cap (W \times W)$.

A *Strongly Connected Component (SCC)* of G is a maximal set $C \subseteq V$ such that for all $v, w \in C$ there is a path from v to w. An SCC C is *nontrivial* if for all $v, w \in C$ there is a nontrivial path from v to w. For $v \in V$, we define $\mathrm{SCC}(v)$ to be the SCC that contains v. The number of SCCs (nontrivial SCCs) of G is denoted by $N(G)$ ($N'(G)$). The SCC-quotient graph of (V, E) is a graph (V', E') with $V' = \{\mathrm{SCC}(v) \mid v \in V\}$ and $E' = \{(C, C') \mid C \neq C' \text{ and } \exists v \in C, v' \in C' : (v, v') \in E\}$.

Since the SCC graph is acyclic, it induces a partial order, and when we speak of a 'maximal' or 'minimal' SCC, we refer to this order. Specifically, a

Table 1. Glossary of symbols used in complexity analysis

d : diameter	h : height of the SCC quotient graph
N : number of SCCs	N' : number of nontrivial SCCs
n : number of nodes	p : number of non-Büchi fairness constraints

minimal (maximal) SCC has no incoming (outgoing) edges. The height of the SCC quotient graph (the length of its longest path) is denoted by $h(G)$. A subset $S \subseteq V$ is *SCC-closed* if for every SCC C, we have either $C \subseteq S$ or $C \cap S = \emptyset$. The intersection and union of two SCC-closed sets are SCC-closed, and so is the complement with respect to V of an SCC-closed set.

When using $n(G)$, $d(G)$, $h(G)$, $N(G)$, and $N'(G)$, we shall omit G if the graph is understood from the context. See Table 1 for a glossary of the parameters used in this paper.

Let H be a subgraph of G induced by a set of vertices that is SCC-closed. No parameter of Table 1 is larger in H than in G, with the exception of $d(H)$, which can be arbitrarily larger than $d(G)$. We now define a property that guarantees $d(H) \leq d(G)$.

A set $S \subseteq V$ is *path-closed* if any path of G that connects two vertices of S has all its vertices in S. In this case we also call the subgraph induced by S path-closed. Let S be path-closed and let H be its induced subgraph. We use these two simple facts.

1. $d(H) \leq d(G)$.
2. For $T \subseteq S$ both T and $S \setminus T$ are path-closed if no edge goes from T to $S \setminus T$ or no edge goes from $S \setminus T$ to T.

Note that in Property 2, an edge of G may go from T to $V \setminus S$ in the first case, and from $V \setminus S$ to T in the second case.

2.2 Symbolic Algorithms

We represent sets of nodes symbolically by their characteristic function. A symbolic algorithm can manipulate sets using union, intersection, complementation, image and preimage. It can also test a set for emptiness and pick an element from a set V using $\text{pick}(V)$. (The way in which the element is picked is immaterial to our exposition.) The image and preimage computations, also known as EY and EX, respectively, are defined as follows:

$$\begin{aligned} \text{img}_G(S) &= \{v' \in V \mid \exists v \in S : (v, v') \in E\}, \\ \text{preimg}_G(S) &= \{v' \in V \mid \exists v \in S : (v', v) \in E\}. \end{aligned}$$

We shall drop the subscript G if it is clear from the context.

The set of nodes that are reachable from a node v is denoted by $\mathsf{EP}\, v$. It can be computed as the least fixpoint of the function $\lambda T.\, \{v\} \cup \text{img}(T)$. Similarly, the set of nodes that can reach v is the least fixpoint of the function $\lambda T.\, \{v\} \cup \text{preimg}(T)$, and is denoted by $\mathsf{EF}\, v$ [CES86].

In the remainder we shall refer to the computation of the image or preimage of a nonempty set as a *step* (computing the (pre-)image of an empty set requires no work). We shall measure the complexity of a symbolic algorithm as the number of steps that it takes. This choice is motivated by the larger cost normally incurred by image and preimage computations when compared to other operations performed on sets of states like union and intersection. The simplification afforded by concentrating on a few important operations compensates the occasional inaccuracy.

The algorithms we analyze in the following do not modify the transition relation of the automata on which they operate. Keeping the transition relation constant allows for a fairer comparison between different algorithms.

The characteristic functions manipulated by symbolic algorithms are in practice boolean functions that depend on variables ranging over the function domain. To represent a set of n states one then needs a set of at least $\lceil \lg n \rceil$ boolean variables. To represent the transition relation of a graph, twice as many variables are used. To write a formula that says "there is a node z such that x is related to z and z is related to y" (as in the transitive closure computation) one needs three sets of variables. The cost of symbolic computations strongly depends on the number of variables used [HTKB92], which explains why image and preimage are typically the most expensive operations. The algorithms examined in this paper all use the same number of variables.

2.3 Streett and Büchi Automata

A *Streett automaton* over an alphabet A is a tuple $\mathcal{A} = ((V, E), v_0, L, \mathcal{F})$ where (V, E) is a graph, $v_0 \in V$ is the initial node, $L : E \to A$ is the edge-labeling function, and $\mathcal{F} = \{(L_1, U_1), (L_2, U_2), \ldots, (L_k, U_k)\} \subseteq 2^V \times 2^V$ is the acceptance condition (a set of pairs of sets of nodes) [Str82]. For technical reasons we also require that there is a path from v_0 to every node in V. The language of $\mathcal{A}$ is nonempty if (V, E) contains a *fair cycle*: a cycle D such that for all pairs $(L, U) \in \mathcal{F}$, we have that $L \cap D \neq \emptyset$ implies $U \cap D \neq \emptyset$. We write $p(\mathcal{A})$, or simply p, for the number of distinct pairs $(L, U) \in \mathcal{F}$ for which $L \neq V$. We shall use this quantity (which is typically much smaller than n) in the complexity analysis.

A generalized *Büchi automaton* is a Streett automaton $((V, E), v_0, L, \mathcal{F})$ such that for all pairs $(L, U) \in \mathcal{F}$ we have $L = V$ [Büc62]. The language of a Büchi automaton is nonempty if there is a cycle that contains at least one state in U_i for every i. The emptiness condition for Büchi automata can easily be stated in terms of SCCs: the language is nonempty if the automaton contains an SCC C that is *fair*: it is nontrivial and $C \cap U_i \neq \emptyset$, for every i. Note that for Büchi automata p is 0.

The nonemptiness check for Streett automata can also be based on the identification of the SCCs of the automaton graph. However, if an SCC C intersects L_i, but not U_i, it is still possible for it to contain a fair cycle, provided such a cycle does not contain the states in L_i. This can be checked by subtracting the states in L_i from C, and recursively analyzing the SCCs of the resulting subgraph.

3 Complexity of the Emerson-Lei Algorithm

Because of the practical importance of the Emerson-Lei algorithm for Büchi emptiness, we shall analyze its complexity in more detail. To our knowledge, the only known complexity bound is the obvious $\Theta(|\mathcal{F}|n^2)$ bound. In this section we sharpen the bound to $\Theta(|\mathcal{F}|dh)$. In Section 5, we shall contrast the bounds achieved by our algorithm with this bound.

Let $((V, E), v_0, L, \mathcal{F})$ be a Büchi automaton. The Emerson-Lei algorithm computes the set of states F that have a path to a fair cycle; the language of the automaton is empty iff $F = \emptyset$. The algorithm is characterized by the following μ-calculus formula. (See [CGP99] for an overview of the μ-calculus.)

$$F' = \nu Y. \bigcap_{(V,U)\in\mathcal{F}} \operatorname{preimg}\big(\mu Z.\, Y \cap (U \cup \operatorname{preimg}(Z))\big).$$

Evaluation of this formula requires two nested loops. The inner loop identifies the states from which there is a path that is contained in Y and reaches a state in U. It finds these states by starting with the states in $U \cap Y$ and extending the paths backward. Then, the predecessors of these states are computed by preimg to eliminate states that are not on cycles. This procedure is repeated once for every acceptance condition. We call the computation of the intersection of these $|\mathcal{F}|$ fixpoints a *pass*. The outer loop repeatedly performs passes until convergence is achieved.

Theorem 1. *Algorithm Emerson-Lei performs $\Theta(|\mathcal{F}|dh)$ steps in the worst case.*

Proof. We say that node s has distance δ to set S if the shortest distance from s to any node $s' \in S$ is δ.

Suppose the algorithm is called on a graph $G = (V, E)$. If we consider the subgraph induced by Y, then at iteration i of the inner loop, the iterate acquires those states that are at distance i from the set $U \cap Y$. It is easy to see that each iterate Y is path-closed—in fact we always have $\operatorname{preimg}(Y) \subseteq Y$. Hence $d(Y) \leq d$. This limits the number of steps taken in the inner loop to $\mathcal{O}(d)$, and hence the number of steps in a pass to $\mathcal{O}(|\mathcal{F}|d)$.

It remains to be shown that the outer loop performs $\mathcal{O}(h)$ iterations. It is easy to see that throughout the execution of the algorithm, the subgraph induced by the iterate Y consists of a collection of SCCs of G. Each pass deletes every unfair SCC that was maximal at the start of the pass, and the algorithm halts at the end of a pass that began with no maximal unfair SCCs. Hence, it suffices to prove that an unfair SCC C that is maximal at the start of the ith pass is the first vertex of a simple path of length at least $i - 1$ in the SCC quotient graph. We do this by induction on i, as follows.

Suppose that the unfair SCC C is maximal at the start of the $i + 1$st pass. Then C was not maximal at the start of the ith pass. Hence it precedes an SCC

C' that was maximal at the start of the ith pass and was deleted. Clearly C' is unfair, and so by induction C' contains the first node of a path of length at least $i-1$. This gives the desired path of length at least i for C.

These three observations yield an $\mathcal{O}(|\mathcal{F}|dh)$ bound. We now prove that this bound is tight. We first give an example where $|\mathcal{F}|$ can be any integer greater than 1. We will then treat the case $|\mathcal{F}| = 1$. Consider a family of Büchi automata $\mathcal{A}_{k,l,m} = (((V, E), v_1, L, \mathcal{F})$ where $m > 1$ and

$$\begin{aligned}
V &= \{v_i \mid 1 \le i \le k+l\} \cup \{v'_i \mid k < i \le k+l\},\\
E &= \{(v_i, v_{i+1}) \mid 1 \le i \le k\} \cup \{(v_i, v_j), \mid k < i < j \le k+l\} \cup\\
&\qquad \{(v_i, v'_i), (v'_i, v_i) \mid k < i \le k+l\}, \text{ and}\\
\mathcal{F} &= \big\{\big(V, \{v_i \mid k < i \le k+l,\, i-k-1 \mod m \neq q\}\big) \mid 0 \le q < m\big\}.
\end{aligned}$$

The labeling function L is immaterial. We have $d(V, E) = k+1$, $h(V, E) = k + l$, and $|\mathcal{F}| = m$. See Fig. 1 for an example. On these graphs, the inner loop always performs $\Theta(k)$ steps (assuming $k > m$). Every pass executes the inner loop $|\mathcal{F}|$ times and removes only the rightmost SCC, for a total of $\Theta(l)$ passes. This gives a total of $\Theta(|\mathcal{F}|kl) = \Theta(|\mathcal{F}|dh)$ steps.

For the case of $|\mathcal{F}| = 1$ we use the graph above with two modifications: We omit the v'_i vertices and the acceptance condition is $\big\{\big(V, \{v_i \mid k < i \le k+l\}\big)\big\}$. Again, every pass removes only the rightmost SCC (which is now trivial), and we arrive at the same $\Theta(|\mathcal{F}|kl) = \Theta(|\mathcal{F}|dh)$ bound. $\square$

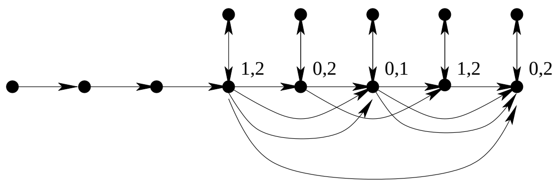

Fig. 1. The automaton $\mathcal{A}_{3,5,3}$. The states are labeled with the acceptance conditions to which they belong

Since $h \le N$, Theorem 1 implies that the Emerson-Lei algorithm is also $\Theta(|\mathcal{F}|dN)$. Note that the examples from Theorem 1 also show that the Emerson-Lei algorithm needs $\Theta(|\mathcal{F}|n^2)$ steps.

4 Algorithm

The algorithm of Xie and Beerel [XB99] picks a node v and then computes the set of nodes that have a path to v ($\mathsf{EF}\, v$) and the set of nodes that have a path from v ($\mathsf{EP}\, v$). The intersection of these two sets is the SCC containing v.

Both sets are SCC-closed and, after removal of the SCC, the algorithm proceeds recursively. We combine this idea with lockstep search into an algorithm that performs backward and forward search simultaneously, until one converges. This gives an $n \log n$ bound.

Fig. 2 shows the Lockstep algorithm. We shall first discuss the SCC-decomposition algorithm and then its application to Büchi and Streett emptiness. The difference between the various applications is isolated in the function Report, shown in Fig. 3.

The algorithm takes a Streett automaton as argument. If we set the acceptance condition to $\emptyset$ then all SCCs are fair, Report never recurs, and the algorithm performs SCC decomposition.

In Line 11, we pick a node v from V. We shall compute SCC(v). Sets F and B contain subsets of $\mathsf{EP}(v)$ and $\mathsf{EF}(v)$, respectively. In Lines 16–21, we successively approximate these sets, using the fixpoint characterizations given in Section 2, until either $F = \mathsf{EP}(v)$ or $B = \mathsf{EF}(v)$. Let us assume, for example, that $F = \mathsf{EP}(v)$, but $B \neq \mathsf{EF}(v)$. In Line 22, the intersection of B and F is a subset of the SCC containing v. The inclusion may be strict, since the largest distance from v to any node in the SCC may be smaller than the largest distance from any node in the SCC to v. (See Fig. 4.) We refine the approximation of B in Lines 28–33, in which Ffront remains empty. When Bfront $\cap F = \emptyset$, B contains all nodes in SCC(v), and the intersection of F and B is the SCC containing v. We report $C = F \cap B$, and then recur. For the recursion, we split the state space in three: $B \setminus C$, $V \setminus B$, and C, and recur on the first two sets.

It is easily seen that B is SCC-closed, and hence $B \setminus C$ and $V \setminus B$ are SCC-closed, too. Correctness follows by induction. If $V = \emptyset$ then it does not contain a fair SCC. If $V \neq \emptyset$, it contains a fair SCC if and only if that SCC is either C, or included in B or $V \setminus B$.

For Büchi emptiness, Report enumerates all fair SCCs, but it never recurs. We can obviously stop exploration when a fair SCC is identified. When checking Streett emptiness, if Report finds a nontrivial SCC that does not immediatly yield a fair cycle, it removes the 'bad nodes' and calls Lockstep on the rest of the SCC. (Cf. [EL87, Kur94].) Since Report recurs, it may report some subsets of SCCs as SCCs.

The restriction of the transition relation in Fig. 2 and Fig. 3 is purely for convenience. In practice we intersect the result of images and preimages with the argument V, so that the transition relation does not change during the computation.

5 Complexity Analysis

In this Section we shall analyze the complexity of the algorithm. We state our main result in Theorem 2, and we refine this bound for the case of SCC-decomposition and Büchi emptiness in Corollary 1.

Theorem 2. *Algorithm Lockstep runs in $\mathcal{O}(n(\log n + p))$ steps.*

```
algorithm Lockstep
in: Streett automaton 𝒜 = ((V, E), v_0, L, ℱ)

begin
   node v;
   set of nodes F, B;
   set of nodes Ffront, Bfront, C, Converged;

   if V = ∅ then return;

   v = pick(V);

   F := {v}; Ffront := {v};
   B := {v}; Bfront := {v};

   while Ffront ≠ ∅ and Bfront ≠ ∅ do
      Ffront := img(Ffront) \ F;
      Bfront := preimg(Bfront) \ B;
      F := F ∪ Ffront;
      B := B ∪ Bfront
   od;

   if Ffront = ∅ then
      Converged := F
   else
      Converged := B;

   while Ffront ∩ B ≠ ∅ or Bfront ∩ F ≠ ∅ do
      Ffront := img(Ffront) \ F;     // One of these two fronts is empty
      Bfront := preimg(Bfront) \ B;
      F := F ∪ Ffront;
      B := B ∪ Bfront
   od;

   C := F ∩ B;

   Report(𝒜, C);

   Lockstep((Converged \ C, E ∩ (Converged \ C)^2), v_0, L, ℱ);
   Lockstep((V \ Converged, E ∩ (V \ Converged)^2), v_0, L, ℱ)
end
```

Fig. 2. The Lockstep algorithm

```
algorithm Report
in: Streett automaton 𝒜 = ((V, E), v_0, L, ℱ)
     SCC C ⊆ V
begin
   set of nodes C';

   print SCC C identified;

   if C is trivial then return;

   if ∀i : C ∩ L_i ≠ ∅ implies C ∩ U_i ≠ ∅ then do
      print "SCC is fair; language is not empty";
      return
   od;

   C' = C;

   for each i such that C ∩ L_i ≠ ∅ and C ∩ U_i = ∅ do
      C' := C' \ L_i
   od

   if C' ≠ ∅ then
      Lockstep((C', E ∩ (C' × C')), v_0, L, ℱ)
end
```

Fig. 3. The Report function

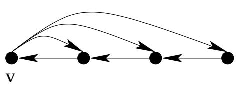

Fig. 4. two steps are needed to compute $\mathsf{EP}(v)$, but four are needed to compute $\mathsf{EF}(v)$. (One step in each direction to establish that a fixpoint has been reached)

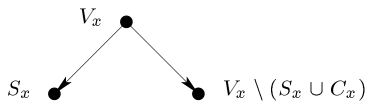

Fig. 5. A vertex x and its children in the recursion tree. Each vertex is labelled with the set V from its argument. We have $|S_x| \leq |V_x|/2$

Proof. We shall first analyze the complexity of the algorithm when it computes an SCC decomposition or performs a Büchi emptiness check. Then we shall extend the result to Streett emptiness, which is slightly harder because Report may call Lockstep.

We shall charge every step of the algorithm to a node, and for SCC decomposition we shall prove that we never charge more than $2 \lg n + 3$ steps to every node, bounding the number of steps by $2n \lg n + 3n$.

An invocation of Lockstep partitions V into Converged $\setminus C$, C, and $V \setminus$ Converged. One of the two sets Converged $\setminus C$ and $V \setminus$ Converged contains at most $n/2$ nodes. We refer to this set as S for small. If $|S \cup C| = k$, the number of steps in the first while loop is at most $2k$. If $S =$ Converged $\setminus C$ this follows easily from the fact that every step acquires at least one state. If $S = V \setminus$ Converged, then the non-converged set is a subset of $S \cup C$. Hence, it has fewer than $|S \cup C| = k$ states, and the first loop cannot have iterated more than k times. We charge 2 steps to every node in $S \cup C$ to account for the cost incurred in the first loop. The second while loop performs at most $|C|$ steps. We account for this by charging one step to every node in C.

To limit the total cost charged to any node, consider the recursion tree R. Every invocation of the algorithm corresponds to a vertex x of R. (See Fig. 5.) The argument V of that invocation is denoted by V_x, and the small set S and SCC C computed in that invocation are denoted by S_x and C_x. An arbitrary node $w \in V$ is passed down only one branch of the tree, until its SCC is identified. At the vertex of R at which SCC(w) is identified, w is charged 3 steps. We now consider the charge at vertices at which the SCC of w has not been identified yet. If w is charged at vertex x of R then w is an element of S_x. Suppose that x' is the closest descendant of x in which w is charged again. We have $w \in S_{x'}$. Since $|S_{x'}| \leq |V_{x'}|/2$, we have $|S_{x'}| \leq |S_x|/2$. Hence, w is charged 2 steps at most $\lg n$ times plus 3 steps when its SCC is identified. This limits the total charge per node to $2 \lg n + 3$, and the total complexity to $2n \lg n + 3n$.

To generalize this result to Streett automata, consider a branch in the recursion tree. We still charge to a node w when it is a member of S_x or C_x for some vertex x of R. Again, w is a member of S_x at no more than $\lg n$ different vertices x, and is charged at most $2 \lg n$ steps. Unlike the Büchi fairness case, w can be a member of C_x at more than one vertex x, because Report may call Lockstep. Every time Report does so, it removes all nodes from at least one set L_i. Since Report recurs only for non-Büchi conditions, it can be invoked in the branch a total of $p + 1$ times[1]. Hence, we can limit the total cost charged to w when it is a member of C_x for some x to $3(p + 1)$. This bounds the total charge to any node by $2 \lg n + 3(p + 1)$, and the total cost by $2n \lg n + 3(p + 1)n$. For Büchi automata, for which p is 0, this bound simplifies to $2n \lg n + 3n$, in agreement with the previous analysis. □

It is not hard to see that $n \log n$ is an asymptotically tight bound for Büchi emptiness. Let $G(n)$ be the linear graph of n nodes. (See Fig. 6.) If Lockstep always picks the middle node then it recurs twice on $G(n/2)$, and it takes $\Theta(n \log n)$

[1] This quantity can be limited to $\min(p + 1, n)$, but typically $p \ll n$.

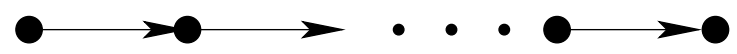

Fig. 6. A linear graph of n nodes

steps. On this example, the algorithm of Xie and Beerel may take up to $\Theta(n^2)$ steps, depending on how it picks nodes.

Lockstep does not always outperform Emerson-Lei. Consider a family of graphs consisting of k identical subgraphs with no edges connecting the subgraphs. The number of steps for the Emerson-Lei algorithm is independent of k, whereas in the case of Lockstep, it grows linearly with k. On the other hand, Lockstep can take arbitrarily fewer steps than Emerson-Lei on a graph like that in Fig. 1.

The next corollary focuses on the case $p = 0$, that is, SCC analysis and Büchi emptiness.

Corollary 1. *If $p = 0$, Lockstep uses $\mathcal{O}(n \log(dN/n))$ steps.*

Proof. We shall first account for the work spent identifying the small sets of size $> d$, and then we shall account for the small sets of size $\leq d$ (for "small" as defined in the proof of Theorem 2). The rest of the work is $\mathcal{O}(n)$ since, as in the proof of Theorem 2, it is charged to the SCCs C.

Observe that when $p = 0$, each argument to Lockstep is a path-closed subgraph (by Property 2 of Section 2.1). Hence d is an upper bound on the diameter of the argument in each invocation of Lockstep. It is easy to see that this implies that the number of steps in one invocation of Lockstep is $O(d)$.

We shall show that $O(n)$ steps are spent processing the small sets of size $> d$. The work on a small set S is $\mathcal{O}(d)$ steps. We can account for this work by charging $\mathcal{O}(d/|S|)$ units to each node of S. It suffices to show that this results in $\mathcal{O}(1)$ units to each fixed node v. For $i \geq 0$, v belongs to at most one small set S_i whose size is between $2^i d$ and $2^{i+1} d$. The charge to v for processing S_i is $\mathcal{O}(1/2^i)$. Hence the total charge to v is less than $\sum_{i=0}^{\infty} \mathcal{O}(1/2^i) = \mathcal{O}(1)$.

We shall now account for the cost incurred identifying small sets of size $\leq d$. Let n_i, $i = 1, \ldots, N$ denote the size of the ith SCC, ordered so that for all i we have $n_i \leq n_{i+1}$. Choose index s, $1 \leq s \leq N$, so that $n_i \leq d$ exactly when $i \leq s$. Since no SCC of size $> d$ is in a small set of size $\leq d$, we can limit our attention to the components indexed from 1 to s.

The proof of Theorem 2 implies that the number of steps spent processing the small sets of size $\leq d$ is

$$\mathcal{O}(\sum_{i=1}^{s} n_i \log(d/n_i)) \quad . \tag{1}$$

This holds because an SCC C is only involved in recursive calls on graphs with at least $|C|$ vertices. Hence, a node of S is charged at most $\log(d/|C|)$ times when it belongs to a small set with no more than d nodes.

The function $x \lg(d/x)$ is concave down. Hence Jensen's Inequality [HLP52] shows that the sum of (1) is maximized when all the n_i's are equal. We distinguish two cases. First, if $n \leq ds$ then $\sum_{i=1}^{s} n_i \leq n$ implies the sum is maximized when $n_i = n/s$ for all $i \leq s$. In this case, (1) simplifies to $\mathcal{O}(n \log(ds/n)) = \mathcal{O}(n \log(dN)/n)$. Second, if $n > ds$ then $\sum_{i=1}^{s} n_i \leq ds$ implies the maximum occurs when $n_i = d$ for all $i \leq s$. In this case (1) becomes $\mathcal{O}(n \log(d/d)) = \mathcal{O}(n)$. □

The bound of Corollary 1, $\mathcal{O}(n \log(dN/n))$, implies the three bounds $\mathcal{O}(n \log d)$, $\mathcal{O}(n \log N)$, and $\mathcal{O}(dN)$. (To see this use the inequalities $N \leq n$, $d \leq n$ and $\lg x \leq x$ respectively.) Furthermore, the bound of Corollary 1 is stronger than these other bounds (e.g., take $d = N = \sqrt{n} \lg n$).

In [XB99], the only bound claimed is $\mathcal{O}(dn)$, but the $\mathcal{O}(dN)$ bound is easily seen to hold for that algorithm, too.

It is not easy to generalize the result of Corollary 1 to Streett emptiness, because the diameter of the arguments cannot be expressed in terms of d when the algorithm removes nodes, and the SCCs that are found in different invocations of the algorithm are not disjoint.

Under the assumption that the number of Büchi acceptance conditions is bounded by a constant, the number of symbolic computations that the algorithm takes (including the set operations) can be bounded by the same asymptotic bound as the number of steps. Without this assumption, up to $|\mathcal{F}|n$ intersections may be used to check fairness, which is not within a constant factor of the number of steps.

6 Trimming

We will now describe a modification to the algorithm that allows us to further sharpen the bounds for Büchi emptiness.

In Line 10 of Lockstep, we can add the following code to *trim* the graph by removing the nodes that either have no path to a node in a nontrivial SCC or have no path from a node in a nontrivial SCC [RBS]:

```
while V changes do
   V := V ∩ img(V) ∩ preimg(V);
od.
```

We shall refer to the modified algorithm as *Lockstep with trimming.*

Note that the work added by this algorithm is limited by $\mathcal{O}(N) = \mathcal{O}(n)$ total, because every iteration of the while loop removes a trivial SCC from V. Hence, Theorem 2 and Corollary 1 remain valid with this modification.

The modification, however, allows us to prove two additional bounds for Büchi emptiness: $\mathcal{O}(dN' + N)$ and $\mathcal{O}(dN + h(N' + 1))$. Examples show that neither of these bounds dominates the other. The former bound improves the previous $\mathcal{O}(dN)$ bound.

Theorem 3. *When $p = 0$, Lockstep with trimming takes $\mathcal{O}(\min(dN' + N, dN' + h(N' + 1)))$ steps.*

Proof. Observe that the trimming preserves the invariant that V is path-closed. Hence d is an upper bound on the diameter in each invocation of Lockstep.

First we shall prove the $\mathcal{O}(dN' + h(N' + 1))$ bound. At every invocation of the algorithm, we either pick a node in a trivial SCC or in a nontrivial SCC. We can only pick a node in a nontrivial SCC N' times. Suppose we pick a node v in a trivial SCC, and suppose again that $F =$ Converged. Since we trimmed the graph before picking a node, v is on a 'bridge': it both has a path from and a path to a nontrivial SCC. The nontrivial SCC C that v can reach is by definition not minimal. When the algorithm recurs on F, set F is trimmed and C becomes minimal. Similarly, if $B =$ Converged, a nontrival SCC that was not maximal becomes maximal. An SCC can only become minimal or maximal once, and hence the total number of invocations in which we pick a node in a trivial SCC is limited to $N' + 1$. The cost of trimming the graphs is $\mathcal{O}(h)$ per invocation. The other costs per invocation are limited by $\mathcal{O}(d)$, as argued before, and hence the total number of steps is $\mathcal{O}(dN' + h(N' + 1))$.

The bound of $\mathcal{O}(dN' + N)$ steps holds because trimming can perform at most N steps total, and the complexity of the rest of the algorithm is bounded by dN', as argued. □

We give an example for which the $\mathcal{O}(dN' + h(N' + 1))$ bound is tight, but the $\mathcal{O}(dN' + N)$ bound is not. Consider the family of graphs G_{klm} that consist of m disjoint copies of the graph $G_{kl} = (V, E)$, defined as follows.

$$\begin{aligned}
V &= \{v_0, v_0', v_{k+1}, v_{k+1}'\} \cup \{v_{ij} \mid 0 < i \le k, 0 < j \le l\}, \text{ and} \\
E &= \{(v_0, v_0'), (v_0', v_0), (v_{k+1}, v_{k+1}'), (v_{k+1}', v_{k+1})\} \cup \{(v_0, v_{1j}) \mid 0 < j \le l\} \cup \\
&\quad\ \{(v_{kj}, v_{k+1}) \mid 0 < j \le l\} \cup \{(v_{ij}, v_{i'j} \mid 1 \le i < i' \le k, 0 < j \le l\}.
\end{aligned}$$

See Fig. 7 for an example. For graph G_{klm} we have $h = k + 1$, $d = k + 2$ $N = m(kl + 2)$, and $N' = 2m$. If the algorithm picks node v_0 in some copy of G_{kl} initially, Lockstep will first find the SCC consisting of v_0 and v_0' in a constant number of steps. Then, trimming takes $\Theta(k)$ steps, and identification of the SCC consisting of v_{k+1} and v_{k+1}' again takes a constant number of steps. This process is repeated m times, once for every copy of G_{kl}. This gives a total number of steps of $\Theta(km) = \Theta(dN' + h(N' + 1))$. The $\mathcal{O}(dN' + N)$ bound is not tight on this example, because N can be arbitrarily larger than k and m. Hence, $\mathcal{O}(dN' + N)$ does not dominate $\mathcal{O}(dN' + h(N' + 1))$. It is also possible to show that $\mathcal{O}(dN' + N)$ is tight and does not dominate $\mathcal{O}(dN' + h(N' + 1))$.

7 Finding Fair Paths with Lockstep

The Emerson-Lei algorithm computes a set of states that contains all the fair SCCs. If the set is not empty, it is desirable to extract a nonemptiness witness, that is a path starting at the initial state and terminating in a fair cycle. The

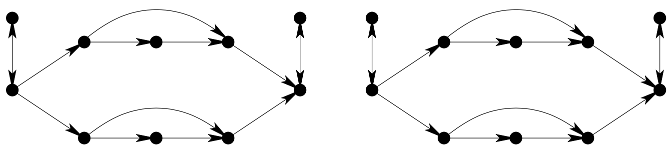

Fig. 7. Graph $G_{3,2,2}$

commonly used algorithm for this task is described in [CGMZ95], and can be shown to have complexity $\mathcal{O}(|\mathcal{F}|dh)$.

We present an algorithm that finds a maximal SCC in $\mathcal{O}(\min(dh, n))$ steps. It can be used to help find a fair cycle when using the Emerson-Lei algorithm, since Emerson-Lei constructs a subgraph where every maximal SCC is fair.

The algorithm is a variant of Lockstep called MaxSCC. It is called with arguments $((V, E), v)$, where (V, E) is a graph and $v \in V$. The algorithm begins with lines 4–10 and 12–35 of Lockstep, which identify sets F, B, C and Converged. (We omit line 11 which defines v, since v is already defined.) The rest of the algorithm is as follows:

```
if C = F then
   return C
else
   v := pick (img(C) \ C);
if F = Converged then
   Closed := F \ C
else
   Closed := V \ B \ C;
MaxSCC(Closed, E ∩ Closed², v)
```

The algorithm sets Closed to an SCC-closed set that contains a maximal SCC and v to a vertex of Closed, and then recurs on Closed. Since Closed is always a proper subset of (the current set) V, the algorithm eventually returns a maximal SCC. Hence the algorithm is correct.

Theorem 4. *Algorithm MaxSCC runs in* $\mathcal{O}(\min(dh, n))$ *steps.*

Proof. To show the bound $\mathcal{O}(dh)$ recall that lines 1–35 of Lockstep use $\mathcal{O}(d)$ steps. Furthermore the choice of v ensures that $\mathrm{SCC}(v)$ is a successor of the SCC found in the previous invocation of the algorithm (if any). Thus there are at most h invocations, and the algorithm uses $\mathcal{O}(dh)$ steps.

To show the $\mathcal{O}(n)$ bound, Lockstep guarantees that the number of steps in one invocation is $\mathcal{O}(|B \cup C|)$. In both cases of the definition of Closed, Closed is disjoint from $B \cup C$. We charge each vertex of $B \cup C$ $\mathcal{O}(1)$ steps. This accounts for all the steps that were executed, and gives total charge $\mathcal{O}(n)$. □

A simple variant of the algorithm consists of returning the first fair SCC found. This is normally desirable when constructing witnesses, because shorter paths are easier to examine.

Given an $\mathcal{O}(dh)$ algorithm to locate a fair SCC in the set returned by the Emerson-Lei algorithm, one can build a fair path in $\mathcal{O}(dh + |\mathcal{F}|d)$ steps. This improves on the $\mathcal{O}(|\mathcal{F}|dh)$ bound of the algorithm in [CGMZ95].

A minor modification of the algorithm described in this section can find a minimal SCC with the same complexity. This version can be used with algorithms like the ones of [HSBK93, KPR98], which return a set of states in which all minimal SCCs are guaranteed fair. In such a case, the algorithm of [CGMZ95] cannot be applied directly; instead, approaches similar to that of [XB99] are used to isolate a fair SCC. Using Lockstep instead of those approaches improves the complexity of this phase of the computation from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

8 Conclusions

We have presented Lockstep, a symbolic algorithm for SCC decomposition with applications in deciding Büchi and Streett emptiness. We analyzed its complexity, which is characterized in terms of the number of image and preimage computations.

For SCC-decomposition and Büchi emptiness, the algorithm has a complexity of $\Theta(n \log n)$ number of steps, where n is the number of nodes in the graph. This improves the known quadratic bounds in number of nodes for both problems [XB99, EL86], although Lockstep does not consistently outperform the Emerson-Lei algorithm.

We also showed a sharper bound of $\mathcal{O}(n \log(dN/n))$ steps, which implies bounds $\mathcal{O}(n \log d)$, $\mathcal{O}(n \log N)$, and $\mathcal{O}(dN)$; a simple optimization also achieves a bound of $\mathcal{O}(\min(dN' + N, dN' + h(N' + 1)))$ steps.

For Streett emptiness, we have bounded the complexity to $\mathcal{O}(n(\log n + p))$. This improves the quadratic bound that was formerly known [KPR98]. Finally, we have shown that Lockstep can be used to produce a nonemptiness witness in $\mathcal{O}(\min(dh, n) + |\mathcal{F}|d)$ steps.

The Lockstep algorithm uses two sets of variables in the characteristic functions (one set to encode the sources of the transitions and the other to encode the destinations) and does not require the modification of the transition relation. Both assumptions are important for runtime efficiency. Relaxing either one may lead to a reduction in the number of steps, but may render the algorithm less practical. For instance, using three sets of variables instead of two allows one to compute the transitive closure of the graph, from which the SCC of the graph can be easily derived [TBK95]. The number of major steps is $\mathcal{O}(d)$ (even without iterative squaring), but the resulting algorithm is impractical.

The computation of the transitive closure can be seen as a maximally parallelized version of an SCC enumeration procedure like the one described in this paper. Lesser degrees of parallelization can also be achieved by either adding fewer than one full set of variables, or by pruning the transition relation so as to conduct multiple searches in common. It remains to be ascertained whether such parallel searches lead to practical algorithms.

In their proposed scheme for complexity classes of symbolic algorithms, Hojati et al. [HTKB92] use the number of sets of variables as one parameter. They do not consider arbitrary numbers of variables, and they do not consider modifications to the transition relations. The SCC decomposition algorithms that conduct a limited number of parallel searches may lead to a refinement of classification of [HTKB92], or to a different approach to the classification of symbolic algorithms.

Acknowledgment

The authors thank Kavita Ravi for drawing their attention to lockstep search.

References

[B+96] R. K. Brayton et al. VIS. In *Formal Methods in Computer Aided Design*, pages 248–256. Springer-Verlag, Berlin, November 1996. LNCS 1166.

[BRS99] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In N. Halbwachs and D. Peled, editors, *Eleventh Conference on Computer Aided Verification (CAV'99)*, pages 222–235. Springer-Verlag, Berlin, 1999. LNCS 1633.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[Büc62] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the 1960 International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

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

[CGMZ95] E. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proceedings of the Design Automation Conference*, pages 427–432, San Francisco, CA, June 1995.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.

[EL86] 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.

[EL87] E. A. Emerson and C. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[ES81] S. Even and Y. Shiloach. An on-line edge-deletion problem. *Journal of the Association for Computing Machinery*, 28(1):1–4, January 1981.

[HHK96] R. H. Hardin, Z. Har'El, and R. P. Kurshan. COSPAN. In *Eighth Conference on Computer Aided Verification (CAV '96)*, pages 423–427. Springer-Verlag, 1996. LNCS 1102.

[HKSV97] 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.

[HLP52] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, 1952.

[HSBK93] R. Hojati, T. R. Shiple, R. Brayton, and R. Kurshan. A unified approach to language containment and fair CTL model checking. In *Proceedings of the Design Automation Conference*, pages 475–481, June 1993.

[HT96] M. R. Henzinger and J. A. Telle. Faster algorithms for the nonemptiness of Streett automata and for communication protocol pruning. In R. Karlsson and A. Lingas, editors, *Algorithm Theory: SWAT'96*, pages 16–27. Springer-Verlag, Berlin, 1996. LNCS 1097.

[HTKB92] 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.

[KPR98] 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.

[Kur94] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, Princeton, NJ, 1994.

[McM87] K. McMillan. Class project on BDD-based verification. Private Communication, E. M. Clarke, 1987.

[McM94] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 1994.

[McM99] K. L. McMillan. Verification of infinite state systems by compositional model checking. In *Correct Hardware Design and Verification Methods (CHARME'99)*, pages 219–233, Berlin, September 1999. Springer-Verlag. LNCS 1703.

[RBS] K. Ravi, R. Bloem, and F. Somenzi. A comparative study of symbolic algorithms for the computation of fair cycles. In these proceedings.

[RS97] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*. Springer, Berlin, 1997.

[Str82] R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54:121–141, 1982.

[Tar72] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1:146–160, 1972.

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

[Tho97] W. Thomas. Languages, automata, and logic. In Rozenberg and Salomaa [RS97], chapter 7, Vol. 3, pages 389–455.

[XB99] 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.