Büchi Complementation and Size-Change Termination*

Seth Fogarty and Moshe Y. Vardi**

Department of Computer Science, Rice University, Houston, TX
{sfogarty,vardi}@cs.rice.edu

Abstract. We compare tools for complementing nondeterministic Büchi automata with a recent termination-analysis algorithm. Complementation of Büchi automata is a key step in program verification. Early constructions using a Ramsey-based argument have been supplanted by rank-based constructions with exponentially better bounds. In 2001 Lee et al. presented the size-change termination (SCT) problem, along with both a reduction to Büchi automata and a Ramsey-based algorithm. This algorithm strongly resembles the initial complementation constructions for Büchi automata.
We prove that the SCT algorithm is a specialized realization of the Ramsey-based complementation construction. Surprisingly, empirical analysis suggests Ramsey-based approaches are superior over the domain of SCT problems. Upon further analysis we discover an interesting property of the problem space that both explains this result and provides a chance to improve rank-based tools. With these improvements, we show that theoretical gains in efficiency are mirrored in empirical performance.

1 Introduction

The automata-theoretic approach to formal program verification reduces questions about program adherence to a specification to questions about language containment. Representing liveness, fairness, or termination properties requires finite automata that operate on infinite words. One automaton, $\mathcal{A}$, encodes the behavior of the program, while another automaton, $\mathcal{B}$, encodes the formal specification. To ensure adherence, verify that the intersection of $\mathcal{A}$ with the complement of $\mathcal{B}$ is empty. Thus a vital problem is constructing the complementary automata $\overline{\mathcal{B}}$. Finite automata on infinite words are classified by their acceptance condition and transition structure. We consider here nondeterministic Büchi automata, in which a run is accepting when it visits at least one accepting state infinitely often.

The first complementation constructions for nondeterministic Büchi automata employed a Ramsey-based combinatorial argument to partition infinite words into a finite set of regular languages. Proposed by Büchi in 1962 [3], this construction was shown in 1987 by Sistla, Vardi, and Wolper to be implementable with a blow-up of $2^{O(n^2)}$ [14]. This brought the complementation problem into singly-exponential blow-up, but left a gap with the $2^{\Omega(n \log n)}$ lower bound proved by Michel [11].

* A full version of this paper, including proofs, is available at http://www.cs.rice.edu/~sfogarty/thesis.pdf

** Work supported in part by NSF grants CCR-0124077, CCR-0311326, CCF-0613889, ANI-0216467, and CCF-0728882, by BSF grant 9800096, and by a gift from Intel.

The gap was tightened in 1988, when Safra described a $2^{O(n \log n)}$ construction [13]. Work since then has focused on improving the practicality of $2^{O(n \log n)}$ constructions, either by providing simpler constructions, further tightening the bound, or improving the derived algorithms. In 2001, Kupferman and Vardi employed a rank-based analysis of Büchi automata to simplify complementation [9]. Recently Doyen and Raskin tightly integrated the rank-based construction with a subsumption relation to provide a complementation solver that scales to automata several orders of magnitude larger than previous tools [5].

Separately, in the context of of program termination analysis, Lee, Jones, and Ben-Amram presented the size-change termination (SCT) principle in 2001 [10]. This principle states that, for domains with well-founded values, if every infinite computation contains an infinitely decreasing value sequence, then no infinite computation is possible. Lee et al. describe a method of size-change termination analysis and reduce this problem to the containment of two Büchi automata. Stating the lack of efficient Büchi containment solvers, they also propose a Ramsey-based combinatorial solution that captures all possible call sequences in a finite set of graphs. The Lee, Jones, and Ben-Amram (LJB) algorithm was provided as a practical alternative to reducing the verification problem to Büchi containment, but bears a striking resemblance to the 1987 Ramsey-based complementation construction [14].

In this paper we show that the LJB algorithm for deciding SCT [10] is a specialized implementation of the 1987 Ramsey-based complementation construction [14]. We then empirically explore Lee et al.'s intuition that Ramsey-based algorithms are more practical than Büchi complementation tools on SCT problems. Initial experimentation does suggest that Ramsey-based tools are superior to rank-based tools on SCT problems. This is surprising, as the worst-case complexity of the LJB algorithm is significantly worse than that of rank-based tools. Investigating this discovery, we note that it is natural for SCT problems to be reverse-deterministic, and that for reverse-deterministic problems the worst-case bound for Ramsey-based algorithms matches that of the rank-based approach. This suggests improving the rank-based approach in the face of reverse determinism. We demonstrate that, indeed, reverse-deterministic automata have a maximum rank of 2, dramatically lowering the complexity of complementation to $2^{O(n)}$. Revisiting our experiments, we discover that with this improvement rank-based tools are superior on the domain of SCT problems.

2 Preliminaries

In this section we review the relevant details of the Büchi complementation and size-change termination, introducing along the way the notation used throughout this paper. An *nondeterministic Büchi automaton on infinite words* is a tuple $\mathcal{B} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$, where Σ is a finite nonempty alphabet, Q a finite nonempty set of states, $Q^{in} \subseteq Q$ a set of initial states, $F \subseteq Q$ a set of accepting states, and $\rho : Q \times \Sigma \to 2^Q$ a nondeterministic transition relation. We lift the ρ function to sets of states and words of arbitrary length in the usual fashion.

A *run* of a Büchi automaton $\mathcal{B}$ on a word $w \in \Sigma^\omega$ is a infinite sequence of states $q_0 q_1 ... \in Q^\omega$ such that $q_0 \in Q^{in}$ and, for every $i \geq 0$, we have $q_{i+1} \in \rho(q_i, w_i)$. A run is *accepting* iff $q_i \in F$ for infinitely many $i \in \mathbb{N}$. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{B}$ if

there is an accepting run of $\mathcal{B}$ on w. The words accepted by $\mathcal{B}$ form the language of $\mathcal{B}$, denoted by $L(\mathcal{B})$. A *path* in $\mathcal{B}$ from q to r is a finite subsection of a run beginning in q and ending in r. A path is *accepting* if some state in the path is in F.

A Büchi automaton $\mathcal{A}$ is contained in a Büchi automaton $\mathcal{B}$ iff $L(\mathcal{A}) \subseteq L(\mathcal{B})$, which can be checked by verifying that the intersection of $\mathcal{A}$ with the complement $\overline{B}$ of $\mathcal{B}$ is empty: $L(\mathcal{A}) \cap L(\overline{\mathcal{B}}) = \emptyset$. We know that the language of an automaton is non-empty iff there are states $q \in Q^{in},\ r \in F$ such that there is a path from q to r and an accepting path from r to itself. The initial path is called the prefix, and the combination of the prefix and cycle is called a *lasso* [16]. Further, the intersection of two automata can be constructed, having a number of states proportional to the product of the number states of the original automata [4]. Thus, the most computationally demanding step is constructing the complement of $\mathcal{B}$. In the formal verification field, existing work has focused on the simplest form of containment testing, universality testing, where $\mathcal{A}$ is the universal automaton [5, 15].

2.1 Ramsey-Based Universality

When Büchi introduced these automata in 1962, he described a complementation construction involving a Ramsey-based combinatorial argument. We describe an improved implementation presented in 1987. To construct the complement of $\mathcal{B}$, where $Q = \{q_0, ..., q_{n-1}\}$, we construct a set $\widetilde{Q}_{\mathcal{B}}$ whose elements capture the essential behavior of $\mathcal{B}$. Each element corresponds to an answer to the following question. Given a finite nonempty word w, for every two states $q, r \in Q$: is there a path in $\mathcal{B}$ from q to r over w, and is some such path accepting?

Define $Q' = Q \times \{0, 1\} \times Q$, and $\widetilde{Q}_{\mathcal{B}}$ to be the subset of $2^{Q'}$ whose elements do not contain both $\langle q, 0, r\rangle$ and $\langle q, 1, r\rangle$ for any q and r. Each element of $\widetilde{Q}_{\mathcal{B}}$ is a $\{0, 1\}$-arc-labeled graph on Q. An arc represents a path in $\mathcal{B}$, and the label is 1 if the path is accepting. Note that there are 3^{n^2} such graphs. With each graph $\widetilde{g} \in \widetilde{Q}_{\mathcal{B}}$ we associate a language $L(\widetilde{g})$, the set of words for which the answer to the posed question is the graph encoded by $\widetilde{g}$.

Definition 1. *Let $\widetilde{g} \in \widetilde{Q}_{\mathcal{B}}$ and $w \in \Sigma^+$. Then $w \in L(\widetilde{g})$ iff, for all pairs of states $q, r \in Q$:*
(1) $\langle q, a, r\rangle \in \widetilde{g},\ a \in \{0, 1\}$, iff there is a path in $\mathcal{B}$ from q to r over w.
(2) $\langle q, 1, r\rangle \in \widetilde{g}$ iff there is an accepting path in $\mathcal{B}$ from q to r over w.

The languages $L(\widetilde{g})$, for the graphs $\widetilde{g} \in \widetilde{Q}_{\mathcal{B}}$, form a partition of Σ^+. With this partition of Σ^+ we can devise a finite family of ω-languages that cover Σ^ω. For every $\widetilde{g},\ \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$, let Y_{gh} be the ω-language $L(\widetilde{g}) \cdot L(\widetilde{h})^\omega$. We say that a language Y_{gh} is *proper* if Y_{gh} is non-empty, $L(\widetilde{g}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{g})$, and $L(\widetilde{h}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{h})$. There are a finite, if exponential, number of such languages. A Ramsey-based argument shows that every infinite string belongs to a language of this form, and that $\overline{L(\mathcal{B})}$ can be expressed as the union of languages of this form.

Lemma 1. [3, 14]
(1) $\Sigma^\omega = \bigcup\{Y_{gh} \mid Y_{gh}$ is proper$\}$

(2) For $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$, either $Y_{gh} \cap L(\mathcal{B}) = \emptyset$ or $Y_{gh} \subseteq L(\mathcal{B})$.
(3) $\overline{L(\mathcal{B})} = \bigcup\{Y_{gh} \mid Y_{gh}$ is proper and $Y_{gh} \cap L(\mathcal{B}) = \emptyset\}$.

To obtain the complementary Büchi automaton $\overline{\mathcal{B}}$, Sistla et al. construct, for each $\widetilde{g} \in \widetilde{Q}_{\mathcal{B}}$, a deterministic automata on finite words, $\mathcal{B}_g$, that accepts exactly $L(\widetilde{g})$. Using the automata $\mathcal{B}_g$, one can then construct the complementary automaton $\overline{\mathcal{B}}$ [14]. We can then use a lasso-finding algorithm on $\overline{\mathcal{B}}$ to prove the emptiness of $\overline{\mathcal{B}}$, and thus the universality of $\mathcal{B}$. We can avoid an explicit lasso search, however, by employing the rich structure of the graphs in $\widetilde{Q}_{\mathcal{B}}$. For every two graphs $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$, determine if Y_{gh} is proper. If Y_{gh} is proper, test if it is contained in $L(\mathcal{B})$ by looking for a lasso with a prefix in $\widetilde{g}$ and a cycle in $\widetilde{h}$. $\mathcal{B}$ is universal if every proper Y_{gh} is so contained.

Lemma 2. *Given an Büchi automaton $\mathcal{B}$ and the set of graphs $\widetilde{Q}_{\mathcal{B}}$,*
(1) $\mathcal{B}$ is universal iff, for every proper Y_{gh}, $Y_{gh} \subseteq L(\mathcal{B})$.
(2) Let $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$ be two graphs where Y_{gh} is proper. $Y_{gh} \subseteq L(\mathcal{B})$ iff there exists $q \in Q^{in},\ r \in Q,\ a \in \{0,1\}$ where $\langle q, a, r\rangle \in \widetilde{g}$ and $\langle r, 1, r\rangle \in \widetilde{h}$.

Lemma 2 yields a PSPACE algorithm to determine universality [14]. Simply check each $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$. If Y_{gh} is both proper and not contained in $L(\mathcal{B})$, then the pair $(\widetilde{g}, \widetilde{h})$ provide a counterexample to the universality of $\mathcal{B}$. If no such pair exists, the automaton must be universal.

2.2 Rank-Based Complementation

If a Büchi automaton $\mathcal{B}$ does not accept a word w, then every run of $\mathcal{B}$ on w must eventually cease visiting accepting states. The rank-based construction uses a notion of ranks to track the progress of each possible run towards fair termination. A *level ranking* for an automaton $\mathcal{B}$ with n states is a function $f : Q \to \{0...2n, \bot\}$, such that if $q \in F$ then $f(q)$ is even or $\bot$. Let a be a letter in Σ and f, f' be two level rankings f. Say that f *covers* f' *under* a when for all q and every $q' \in \rho(q, a)$, if $f(q) \neq \bot$ then $f'(q') \leq f(q)$; i.e. no transition between f and f' on a increases in rank. Let F_r be the set of all level rankings.

If $\mathcal{B} = \langle \Sigma, Q, Q^{in}, \rho, F\rangle$ is a Büchi automaton, define $KV(\mathcal{B})$ to be the automaton $\langle \Sigma, F_r \times 2^Q, \langle f_{in}, \emptyset\rangle, \rho', F_r \times \{\emptyset\}\rangle$, where

- $f_{in}(q) = 2n$ for each $q \in Q^{in}$, $\bot$ otherwise.
- Define $\rho' : \langle F_r \times 2^Q\rangle \times \sigma \to 2^{\langle F_r \times 2^Q\rangle}$ to be
  - If $o \neq \emptyset$ then $\rho'(\langle f, o\rangle, \sigma) =$
    $\{\langle f', o' \setminus d\rangle \mid f$ covers f' under σ, $o' = \rho(o, \sigma)$, $d = \{q \mid f'(q)$ odd$\}\}$.
  - If $o = \emptyset$ then $\rho'(\langle f, o\rangle, \sigma) =$
    $\{\langle f', f' \setminus d\rangle \mid f$ covers f' under a, $d = \{q \mid f'(q)$ odd$\}\}$.

Lemma 3. [9] *For every Büchi automaton $\mathcal{B}$, $L(KV(\mathcal{B})) = \overline{L(\mathcal{B})}$.*

An algorithm seeking to refute the universality of $\mathcal{B}$ can look for a lasso in the state-space of $KV(\mathcal{B})$. The strongest algorithm performing this search takes advantage of the presence of a subsumption relation in the KV construction: one state $\langle f, o\rangle$ subsumes

another $\langle f', o' \rangle$ iff $f'(x) \leq f(x)$ for every $x \in Q$, $o' \subseteq o$, and $o = \emptyset$ iff $o' = \emptyset$. When computing the backward-traversal lasso-finding fixed point, it is sufficient to represent a set of states with the maximal elements under this relation. Further, the predecessor operation over a single state and letter results in at most two incomparable elements. This algorithm has scaled to automata an order of magnitude larger than other approaches [5].

2.3 Size-Change Termination

In [10] Lee et al. proposed the size-change termination (SCT) principle for programs: "If every infinite computation would give rise to an infinitely decreasing value sequence, then no infinite computation is possible." The original presentation concerned a first-order pure functional language, where every infinite computation arises from an infinite call sequence and values are always passed through a sequence of parameters.

Proving that a program is size-change terminating is done in two phases. The first extracts from a program a set of size-change graphs, $\mathcal{G}$, containing guarantees about the relative size of values at each function call site. The second phase, and the phase we focus on, analyzes these graphs to determine if every infinite call sequence has a value that descends infinitely along a well-ordered set. For a discussion of the abstraction of language semantics, refer to [10].

Definition 2. *A* size-change graph *(SCG) from function f_1 to function f_2, written $G : f_1 \to f_2$, is a bipartite $\{0, 1\}$-arc-labeled graph from the parameters of f_1 to the parameters of f_2, where $G \subseteq P(f_1) \times \{0, 1\} \times P(f_2)$ does not contain both $x \xrightarrow{1} y$ and $x \xrightarrow{0} y$.*

Size-change graphs capture information about a function call. An arc $x \xrightarrow{1} y$ indicates that the value of x in the function f_1 is strictly greater than the value passed as y to function f_2. An arc $x \xrightarrow{0} y$ indicates that x's value is greater than or equal to the value given to y. We assume that all call sites in a program are reachable from the entry points of the program[1].

A *size-change termination* (SCT) problem is a tuple $L = \langle H, P, C, \mathcal{G} \rangle$, where H is a set of functions, P a mapping from each function to its parameters, C a set of call sites between these functions, and $\mathcal{G}$ a set of SCGs for C. A call site is written $c : f_1 \to f_2$ for a call to function f_2 occurring in the body of f_1. The size-change graph for a call site $c : f_1 \to f_2$ is written as G_c. Given a SCT problem L, a *call sequence* in L is a infinite sequence $cs = c_0, c_1, \ldots \in C^\omega$, such that there exists a sequence of functions $f_0, f_1, \ldots$ where $c_0 : f_0 \to f_1$, $c_1 : f_1 \to f_2 \ldots$. A *thread* in a call sequence $c_0, c_1, \ldots$ is a connected sequence of arcs, $x \xrightarrow{a} y, y \xrightarrow{b} z, \ldots$, beginning in some call c_i such that $x \xrightarrow{a} y \in G_{c_i}, y \xrightarrow{b} z \in G_{c_{i+1}}, \ldots$. We say that L is *size-change terminating* if every call sequence contains a thread with infinitely many 1-labeled arcs. Note that a thread need not begin at the start of a call sequence. A sequence must terminate if *any* well-founded value decreases infinitely often. Therefore threads can begin at any function

[1] The implementation provided by Lee et al. [10] also make this assumption, and in the presence of unreachable functions size-change termination may be undetectable.

call, in any parameter. We call this the *late-start property* of SCT problems, and revisit it in Section 3.2.

Every call sequence can be represented as a word in C^ω, and a SCT problem reduced to the containment of two ω-languages. The first language $Flow(L) = \{cs \in C^\omega \mid cs \text{ is a call sequence}\}$, contains all call sequences. The second language, $Desc(L) = \{cs \in Flow(L) \mid \text{some thread in } cs \text{ has infinitely many 1-labeled arcs}\}$, contains only call sequences that guarantee termination. A SCT problem L is size-change terminating if and only if $Flow(L) \subseteq Desc(L)$.

Lee et al. [10] describe two Büchi automata, $\mathcal{A}_{Flow(L)}$ and $\mathcal{A}_{Desc(L)}$, that accept these languages. $\mathcal{A}_{Flow(L)}$ is simply the call graph of the program. $\mathcal{A}_{Desc(L)}$ waits in a copy of the call graph and nondeterministically chooses the beginning point of a descending thread. From there it ensures that a 1-labeled arc is taken infinitely often. To do so, it keeps two copies of each parameter, and transitions to the accepting copy only on a 1-labeled arc. Lee et al. prove that $L(\mathcal{A}_{Flow(L)}) = Flow(L)$, and $L(\mathcal{A}_{Desc(L)}) = Desc(L)$

Definition 3. [2]
$\mathcal{A}_{Flow(L)} = \langle C, H, H, \rho_F, H\rangle$, where
- $\rho_F(f_1, c) = \{f_2 \mid c : f_1 \to f_2\}$

$\mathcal{A}_{Desc(L)} = \langle C, Q_1 \cup H, H, \rho_D, F\rangle$, where
- $Q_1 = \{\langle x, r\rangle \mid f \in H,\ x \in P(f),\ r \in \{1, 0\}\}$,
- $\rho_D(f_1, c) = \{f_2 \mid c : f_1 \to f_2\} \cup \{\langle x, r\rangle \mid c : f_1 \to f_2,\ x \in P(f_2),\ r \in \{0, 1\}\}$
- $\rho_D(\langle x, r\rangle, c) = \{\langle x', r'\rangle \mid x \xrightarrow{r'} x' \in \mathcal{G}_c\}$,
- $F = \{\langle x, 1\rangle \mid f \in H,\ x \in P(f)\}$

Using the complementation constructions of either Section 2.1 or 2.2 and a lasso-finding algorithm, we can determine the containment of $\mathcal{A}_{Flow(L)}$ in $\mathcal{A}_{Desc(L)}$. Lee et al. propose an alternative graph-theoretic algorithm, employing SCGs to encode descent information about entire call sequences. A notion of composition is used, where a call sequence $c_0 ... c_{n-1}$ has a thread from x to y if and only if the composition of the SCGs for each call, $G_{c_0}; ...; G_{c_{n-1}}$, contains the arc $x \xrightarrow{a} y$. The closure S of $\mathcal{G}$ under the composition operation is then searched for a counterexample describing an infinite call sequence with no infinitely descending thread.

Definition 4. *Let $G : f_1 \to f_2$ and $G' : f_2 \to f_3$ be two SCGs. Their composition $G; G'$ is defined as $G'' : f_1 \to f_3$ where:*

$$\begin{aligned} G'' = \{&x \xrightarrow{1} z \mid x \xrightarrow{a} y \in G,\ y \xrightarrow{b} z \in G',\ y \in P(f_2),\ a = 1 \text{ or } b = 1\} \\ \cup\ \{&x \xrightarrow{0} z \mid x \xrightarrow{0} y \in G,\ y \xrightarrow{0} z \in G',\ y \in P(f_2), \text{ and} \\ &\forall y', r, r' \,.\, x \xrightarrow{r} y' \in G \wedge y' \xrightarrow{r'} z \in G' \text{ implies } r = r' = 0\} \end{aligned}$$

[2] The original LJB construction [10] restricted edges from functions to parameters to the 0-labeled parameters. This was changed to simplify Section 3.3. The modification does not change the accepted language.

Theorem 1. [10] *A SCT problem $L = \langle H, P, C, \mathcal{G}\rangle$ is* not *size-change terminating iff S, the closure of $\mathcal{G}$ under composition, contains a SCG graph $G : f \to f$ such that $G = G;G$ and G does* not *contain an arc of the form $x \overset{1}{\to} x$.*

Theorem 1, whose proof uses a Ramsey-based argument, yields an algorithm that determines the size-change termination of an SCT problem $L = \langle H, P, C, \mathcal{G}\rangle$ by ensuring the absence of a counterexample in the closure of $\mathcal{G}$ under composition. First, use an iterative algorithm to build the closure set S: initialize S as $\mathcal{G}$; and for every $G : f_1 \to f_2$ and $G' : f_2 \to f_3$ in S, include the composition $G;G'$ in S. Second, check every $G : f_1 \to f_1 \in S$ to ensure that if G is idempotent, i.e. $G = G;G$, then G contains an arc of the form $x \overset{1}{\to} x$.

3 Size-Change Termination and Ramsey-Based Containment

The Ramsey-based test of Section 2.1 and the LJB algorithm of Section 2.3 bear a more than passing similarity. In this section we bridge the gap between the Ramsey-based universality test and the LJB algorithm, by demonstrating that the LJB algorithm is a specialized realization of the Ramsey-based containment test. This first requires developing a Ramsey-based framework for Büchi -containment testing.

3.1 Ramsey-Based Containment with Supergraphs

To test the containment of a Büchi automaton $\mathcal{A}$ in a Büchi automaton $\mathcal{B}$, we could construct the complement of $\mathcal{B}$ using either the Ramsey-based or rank-based construction, compute the intersection automaton of $\mathcal{A}$ and $\overline{\mathcal{B}}$, and search this intersection automaton for a lasso. With universality, however, we avoided directly constructing $\overline{\mathcal{B}}$ by exploiting the structure of states in the Ramsey-based construction (see Lemma 2). We demonstrate a similar test for containment.

Consider two automata, $\mathcal{A} = \langle \Sigma, Q_{\mathcal{A}}, Q^{in}_{\mathcal{A}}, \rho_{\mathcal{A}}, F_{\mathcal{A}}\rangle$ and $\mathcal{B} = \langle \Sigma, Q_{\mathcal{B}}, Q^{in}_{\mathcal{B}}, \rho_{\mathcal{B}}, F_{\mathcal{B}}\rangle$. When testing the universality of $\mathcal{B}$, any word not in $L(\mathcal{B})$ is a sufficient counterexample. To test $L(\mathcal{A}) \subseteq L(\mathcal{B})$ we must restrict our search to the subset of Σ^ω accepted by $\mathcal{A}$. In Section 2.1, we defined a set $\widetilde{Q}_{\mathcal{B}}$, which provides a family of languages that covers Σ^ω (see Lemma 1). We now define a set, $\widehat{Q}_{\mathcal{A},\mathcal{B}}$, which provides a family of languages covering $L(\mathcal{A})$.

We first define $\bar{Q}_{\mathcal{A}} = Q_{\mathcal{A}} \times Q_{\mathcal{A}}$ to capture the connectivity in $Q_{\mathcal{A}}$. An element $\bar{g} = \langle q, r\rangle \in \bar{Q}_{\mathcal{A}}$ is a single arc asserting the existence of a path in $\mathcal{A}$ from q to r. With each arc we associate a language, $L(\bar{g})$. Given a word $w \in \Sigma^+$, say that $w \in L(\langle q, r\rangle)$ iff there is a path in $\mathcal{A}$ from q to r over w. Define $\widehat{Q}_{\mathcal{A},\mathcal{B}}$ as $\bar{Q}_{\mathcal{A}} \times \widetilde{Q}_{\mathcal{B}}$. The elements of $\widehat{Q}_{\mathcal{A},\mathcal{B}}$, called *supergraphs*, are pairs consisting of an arc from $\bar{Q}_{\mathcal{A}}$ and a graph from $\widetilde{Q}_{\mathcal{B}}$. Each element simultaneously captures all paths in $\mathcal{B}$ and a single path in $\mathcal{A}$. The language $L(\langle \bar{g}, \widetilde{g}\rangle)$ is then $L(\bar{g}) \cap L(\widetilde{g})$. For convenience, we implicitly take $\widehat{g} = \langle \bar{g}, \widetilde{g}\rangle$, and say $\langle q, a, r\rangle \in \widehat{g}$ when $\langle q, a, r\rangle \in \widetilde{g}$.

The languages $L(\widehat{g}),\ \widehat{g} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$, cover all finite subwords of $L(\mathcal{A})$. With them we define a finite family of ω-languages that cover $L(\mathcal{A})$. Given $\widehat{g},\ \widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$, let Z_{gh} be the ω-language $L(\widehat{g}) \cdot L(\widehat{h})^\omega$. Z_{gh} is called *proper* if: (1) Z_{gh} is non-empty; (2)

$\bar{g} = \langle q, r\rangle$ and $\bar{h} = \langle r, r\rangle$ where $q \in Q^{in}_{\mathcal{A}}$ and $r \in F_{\mathcal{A}}$; (3) $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{g})$ and $L(\widehat{h}) \cdot L(\widehat{h}) \subseteq L(\widehat{h})$. We note that Z_{gh} is non-empty if $L(\widehat{g})$ and $L(\widehat{h})$ are non-empty, and that, by the second condition, every proper Z_{gh} is contained in $L(\mathcal{A})$.

Lemma 4. *Let $\mathcal{A}$ and $\mathcal{B}$ be two Büchi automata, and $\widehat{Q}_{\mathcal{A},\mathcal{B}}$ be the corresponding set of supergraphs.*
(1) $L(\mathcal{A}) = \bigcup\{Z_{gh} \mid Z_{gh}$ is proper$\}$
(2) For all proper Z_{gh}, either $Z_{gh} \cap L(\mathcal{B}) = \emptyset$ or $Z_{gh} \subseteq L(\mathcal{B})$
(3) $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff every proper language $Z_{gh} \subseteq L(\mathcal{B})$.
(4) Let $\widehat{g}, \widehat{h}$ be two supergraphs such that Z_{gh} is proper. $Z_{gh} \subseteq L(\mathcal{B})$ iff there exists $q \in Q^{in}_{\mathcal{B}},\ r \in Q_{\mathcal{B}},\ a \in \{0,1\}$ such that $\langle q, a, r\rangle \in \widehat{g}$ and $\langle r, 1, r\rangle \in \widehat{h}$.

In an analogous fashion to Section 2.1, we can use supergraphs to test the containment of two automata, $\mathcal{A}$ and $\mathcal{B}$. Search all pairs of supergraphs, $\widehat{g}, \widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$ for a pair that is both proper and for which there does not exist a $q \in Q^{in}_{\mathcal{B}}$, $r \in Q_{\mathcal{B}}, a \in \{0,1\}$ such that $\langle q, a, r\rangle \in \widehat{g}$ and $\langle r, 1, r\rangle \in \widehat{h}$. Such a pair is a counterexample to containment. If no such pair exists, then $L(\mathcal{A}) \subseteq L(\mathcal{B})$. We call this search the *double-graph search*, to distinguish from later algorithms for which a counterexample is a single graph.

The double-graph search faces difficulty on two fronts. First, the number of potential supergraphs is very large. Secondly, checking language nonemptiness is an exponentially difficult problem. To address these problems we construct only supergraphs with non-empty languages. Borrowing the notion of composition from Section 2.3 allows us to use exponential space to compute exactly the needed supergraphs. We start with graphs corresponding to single letters and compose them until we reach closure. The resulting subset of $\widehat{Q}_{\mathcal{A},\mathcal{B}}$, written $\widehat{Q}^{f}_{\mathcal{A},\mathcal{B}}$, contains exactly the supergraphs with non-empty languages. In addition to removing the need to check for emptiness, composition allows us to test the sole remaining aspect of properness, language containment, in time polynomial in the size of the supergraphs.

3.2 Strongly Suffix Closed Languages

Theorem 1 suggests that, for some languages, a cycle implies the existence of a lasso. For Büchi automata of such languages, it is sufficient, when disproving containment, to search for a graph $\widehat{h} \in \widehat{Q}_{\mathcal{B}}$, where $\widehat{h}; \widehat{h} = \widehat{h}$, with no arc $\langle r, 1, r\rangle$. This single-graph search reduces the complexity of our algorithm significantly. What enables this in size-change termination is the late-start property: threads can begin at any point. We here define the class of automata amenable to this optimization, beginning with universality for simplicity.

In size-change termination, an accepting cycle can start at any point. Thus the arc $\langle r, 1, r\rangle \in \widetilde{h}$ does not need an explicit matching prefix $\langle q, a, r\rangle$ in some $\widetilde{g}$. In the context of universality, we can apply this method when it is safe to add or remove arbitrary prefixes of a word. To describe these languages we extend the standard notion of *suffix closure*. A language L is suffix closed when, for every $w \in L$, every suffix of w is in L.

Definition 5. *A language L is* strongly suffix closed *if it is suffix closed and for every $w \in L,\ w_1 \in \Sigma^+$, we have that $w_1 w \in L$.*

Lemma 5. *Let $\mathcal{B}$ be an Büchi automaton where every state in Q is reachable and $L(\mathcal{B})$ is strongly suffix closed. $\mathcal{B}$ is* not *universal iff the set of supergraphs with non-empty languages, $\widetilde{Q}^f_{\mathcal{B}}$, contains a graph $\widetilde{h} = \widetilde{h}; \widetilde{h}$ with no arc of the form $\langle r, 1, r\rangle$.*

To extend this notion to handle containment questions $L_1 \subseteq L_2$, we restrict our focus to words in L_1. Instead of requiring L_2 to be closed under arbitrary prefixes, L_2 need only be closed under prefixes that keep the word in L_1.

Definition 6. *A language L_2 is* strongly suffix closed with respect to L_1 *when L_2 is suffix closed and, for every $w \in L_1 \cap L_2,\ w_1 \in \Sigma^+$, if $w_1 w \in L_1$ then $w_1 w \in L_2$.*

Lemma 6. *Let $\mathcal{A}$ and $\mathcal{B}$ be two Büchi automata where $Q^{in}_{\mathcal{A}} = Q_{\mathcal{A}}$,[3] every state in $Q_{\mathcal{B}}$ is reachable, and $L(\mathcal{B})$ is strongly suffix closed with respect to $L(\mathcal{A})$. Then $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$ iff $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ contains a supergraph $\widehat{h} = \langle\langle s, s\rangle, \widetilde{h}\rangle$ where $s \in F_{\mathcal{A}}$, $\widehat{h}; \widehat{h} = \widehat{h}$ and there is no arc $\langle r, 1, r\rangle \in \widehat{h}$.*

Lemma 6 provides a simplified test for the containment of $\mathcal{A}$ in $\mathcal{B}$ when $L(\mathcal{B})$ is strongly suffix closed with respect to $L(\mathcal{A})$. Search all supergraphs in $\widehat{Q}_{\mathcal{A},\mathcal{B}}$ for an supergraph $\widehat{h}$ where $\widehat{h}; \widehat{h} = \widehat{h}$ that does not contain an arc of the form $\langle r, 1, r\rangle$. The presence of this counterexample refutes containment, and the absence of such a supergraph proves containment. We call this search the *single-graph search*.

3.3 From Ramsey-Based Containment to Size-Change Termination

We can now delve into the connection between the LJB algorithm for size-change termination and the Ramsey-based containment test. SCGs of the LJB algorithm are direct analogues of supergraphs in the Ramsey-based containment test of $\mathcal{A}_{Flow(L)}$ and $\mathcal{A}_{Desc(L)}$.

Noting that the LJB algorithm examines single SCGs G where $G = G; G$, we show that for an SCT problem $L = \langle H, P, C, \mathcal{G}\rangle$ the conditions of Lemma 6 are met. First, every state in $\mathcal{A}_{Flow(L)}$ is an initial state. Second, every function in L is reachable, and so every state in $\mathcal{A}_{Desc(L)}$ is reachable.[4] Finally, the late-start property is precisely $Desc(L)$ being strongly suffix closed with respect to $Flow(L)$. Therefore we can use the single-graph search.

Consider supergraphs in $\widehat{Q}_{\mathcal{A}_{Flow(L)},\mathcal{A}_{Desc(L)}}$. The state space of $\mathcal{A}_{Flow(L)}$ is the set of functions H, and the state space of $\mathcal{A}_{Desc(L)}$ is the union of H and Q_1, the set of all $\{0,1\}$-labeled parameters. A supergraph in $\widehat{Q}_{\mathcal{A}_{Flow(L)},\mathcal{A}_{Desc(L)}}$ thus comprises an arc $\langle q, r\rangle$ in H and a $\{0,1\}$-labeled graph $\widetilde{g}$ over $H \cup Q_1$. The arc asserts the existence of a call path from q to r, and the graph $\widetilde{g}$ captures the relevant information about corresponding paths in $\mathcal{A}_{Desc(L)}$.

These supergraphs are almost the same as SCGs, $G : q \to r$. Aside from notational differences, both contain an arc, which asserts the existence of a call path between two functions, and a $\{0,1\}$-labeled graph. There are vertices in both graphs that correspond

[3] With a small amount of work, the restriction that $Q^{in}_{\mathcal{A}} = Q_{\mathcal{A}}$ can be relaxed to the requirement that $L(\mathcal{A})$ be suffix closed.

[4] In the original reduction, 1-labeled parameters may not have been reachable.

to parameters of functions, and arcs between two such vertices describe a thread between the corresponding parameters. The analogy falls short, however, on three points:

(1) In SCGs, vertices are always parameters of functions. In supergraphs, vertices can be either parameters of functions or function names.

(2) In SCGs, vertices are unlabeled. In supergraphs, vertices are labeled either 0 or 1.

(3) In SCGs, only vertices corresponding to parameters of two specific functions are present. In supergraphs, vertices corresponding to every parameters of every functions exist.

We show, in turn, that each difference is an opportunity to specialize the Ramsey-based containment algorithm.

(1) No functions in H are accepting for $\mathcal{A}_{Desc(L)}$, and once we transition out of H into Q_1 we can never return to H. Therefore vertices corresponding to function names can never be part of a descending arc $\langle r, 1, r\rangle$. Since we only search $\widehat{J}$ for a cycle $\langle r, 1, r\rangle$, we can simplify supergraphs in $\widehat{Q}_{\mathcal{A}_{Flow(L)},\mathcal{A}_{Desc(L)}}$ by removing all vertices corresponding to functions.

(2) The labels on parameters are the result of encoding a Büchi edge acceptance condition in a Büchi state acceptance condition automaton, and can be dropped from supergraphs with no loss of information. Consider an arc $\langle\langle f, a\rangle, b, \langle g, c\rangle\rangle$. If b is 1, we know the corresponding thread contains a descending arc. The value of c tells us if the final arc in the thread is descending, but which arc is descending is irrelevant. Thus it is safe to simplify supergraphs in $\widehat{Q}_{\mathcal{A}_{Flow(L)},\mathcal{A}_{Desc(L)}}$ by removing labels on parameters.

(3) While all parameters are states in $\mathcal{A}_{Desc(L)}$, each supergraph describes threads in a call sequence between two functions. There are no threads in this call sequence between parameters of other functions, and so no supergraph with a non-empty language has arcs between the parameters of other functions. We can thus simplify supergraphs in $\widehat{Q}_{\mathcal{A}_{Flow(L)},\mathcal{A}_{Desc(L)}}$ by removing all vertices corresponding to parameters of other functions.

We can specialize the Ramsey-based containment algorithm for $L(\mathcal{A}_{Flow(L)}) \subseteq L(\mathcal{A}_{Desc(L)})$ in two ways. First, by Lemma 6 we know that $Flow(L) \subseteq Desc(L)$ if and only if $\widehat{Q}_{\mathcal{A}_{Flow(L)},\mathcal{A}_{Desc(L)}}$ contains an idempotent graph $\widehat{g} = \widehat{g}; \widehat{g}$ with no arc of the form $\langle r, 1, r\rangle$. Secondly, we can simplify supergraphs in $\widehat{Q}_{\mathcal{A}_{Flow(L)},\mathcal{A}_{Desc(L)}}$ by removing the labels on parameters and keeping only the vertices associated with appropriate parameters. The simplifications of supergraphs whose languages contain single characters are in one-to-one corresponding with $\mathcal{G}$, the initial set of SCGs. As every state in $Flow(L)$ is accepting, every idempotent supergraph can serve as a counterexample. Therefore $Desc(L) \subseteq Flow(L)$ if and only if the closure of the set of simplified supergraphs under composition contains an idempotent supergraph with no arc of the form $\langle r, 1, r\rangle$. This is precisely the algorithm provided by Theorem 1.

4 Empirical Analysis

All the Ramsey-based algorithms presented in Section 2.3 have worst-case running times that are exponentially slower than those of the rank-based algorithms. We now compare existing, Ramsey-based, SCT tools tools to a rank-based Büchi containment solver on the domain of SCT problems.

4.1 Towards an Empirical Comparison

To facilitate a fair comparison, we briefly describe two improvements to the algorithms presented above. First, in constructing the analogy between SCGs in the LJB algorithm and supergraphs in the Ramsey-based containment algorithm, we noticed that supergraphs contain vertices for every parameter, while SCGs contain only vertices corresponding to parameters of relevant functions. These vertices are states in $\mathcal{A}_{Desc(L)}$. While we can specialize the Ramsey-based test to avoid them, Büchi containment solvers might suffer. These states duplicate information. As we already know which functions each supergraph corresponds to, there is no need for each vertex to be unique to a specific function.

The extra states emerge because $Desc(L)$ only accepts strings that are contained in $Flow(L)$. But the behavior of $\mathcal{A}_{Desc(L)}$ on strings not in $Flow(L)$ is irrelevant to the question of $Flow(L) \subseteq Desc(L)$, and we can replace the names of parameters in $\mathcal{A}_{Desc(L)}$ with their location in the argument list. By using this observation, we can simplify the reduction from SCT problems to Büchi containment problems. Experimental results demonstrate that these changes do improve performance.

Second, in [2], Ben-Amram and Lee present a polynomial approximation of the LJB algorithm for SCT. To facilitate a fair comparison, they optimize the LJB algorithm for SCT by using subsumption to remove certain SCGs when computing the closure under composition. This suggests that the single-graph search of Lemma 6 can also employ subsumption. When computing the closure of a set of supergraphs under compositions, we can ignore elements when they are conservatively approximated, or subsumed, by other elements. Intuitively, a supergraph $\widehat{g}$ conservatively approximates another supergraph $\widehat{h}$ when it is strictly harder to find a 1-labeled sequence of arcs through $\widehat{g}$ than through $\widehat{h}$. When the right arc can be found in $\widehat{g}$, then it also occurs in $\widehat{h}$. If $\widehat{g}$ does not have a satisfying arc, then we already have a counterexample supergraph. Formally, given two graphs $\widehat{g}, \widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$ where $\bar{g} = \bar{h}$, say that $\widehat{g}$ *conservatively approximates* $\widehat{h}$, written $\widehat{g} \preceq \widehat{h}$, when for every arc $\langle q, a, r\rangle \in \widehat{g}$ there is an arc $\langle q, a', r\rangle \in \widehat{h}$, where if $a = 1$ then $a' = 1$. Note that conservative approximation is a transitive relation. In order to safely employ conservative approximation as a subsumption relation, we replace the search for a single arc in idempotent graphs with a search for a strongly connected component in all graphs. Extending this relationship to the double-graph search is an open problem.

4.2 Experimental Results

All experiments were performed on a Dell Optiplex GX620 with a single 1.7Ghz Intel Pentium 4 CPU and 512 MB. Each tool was given 3500 seconds, a little under one hour, to complete each task.

Tools: The formal-verification community has implemented rank-based tools in order to measure the scalability of various approaches. The programming-languages community has implemented several Ramsey-based SCT tools. We use the best-of-breed rank-based tool, **Mh**, developed by Doyen and Raskin [5], that leverages a subsumption relation on ranks. We expanded the Mh tool to handle Büchi containment problems

with arbitrary languages, thus implementing the full containment-checking algorithm presented in their paper.

We use two Ramsey-based tools. **SCTP** is a direct implementation of the LJB algorithm of Theorem 1, written in Haskell [7]. We have extended SCTP to reduce SCT problems to Büchi containment problems, using either Definition 3 or our improved reduction. **sct/scp** is an optimized C implementation of the SCT algorithm, which uses the subsumption relation of Section 4.1 [2].

Problem Space: Existing experiments on the practicality of SCT solvers focus on examples extracted from the literature [2]. We combine examples from a variety of sources [1, 2, 7, 8, 10, 12, 17]. The time spent reducing SCT problems to Büchi automata never took longer than 0.1 seconds and was dominated by I/O. Thus this time was not counted. We compared the performance of the rank-based Mh solver on the derived Büchi containment problems to the performance of the existing SCT tools on the original SCT problems. If an SCT problem was solved in all incarnations and by all tools in less than 1 second, the problem was discarded as uninteresting. Unfortunately, of the 242 SCT problems derived from the literature, only 5 prove to be interesting.

Experiment Results: Table 1 compares the performance of the rank-based Mh solver against the performance of the existing SCT tools, displaying which problems each tool could solve, and the time taken to solve them. Of the interesting problems, both SCTP and Mh could only complete 3. On the other hand, sct/scp completed all of them, and had difficulty with only one problem.

Problem	SCTP (s)	Mh (s)	sct/scp (s)
ex04 [2]	1.58	Time Out	1.39
ex05 [2]	Time Out	Time Out	227.7
ms [7]	Time Out	0.1	0.02
gexgcd [7]	0.55	14.98	0.023
graphcolour2 [8]	0.017	3.18	0.014

Table 1: SCT problem completion time by tool.

The small problem space makes it difficult to draw firm conclusions, but it is clear that Ramsey-based tools are comparable to rank-based tools on SCT problems: the only tool able to solve all problems was Ramsey based. This is surprising given the significant difference in worst-case complexity, and motivates further exploration.

5 Reverse-Determinism

In the previous section, the theoretical gap in performance between Ramsey and rank-based solutions was not reflected in empirical analysis. Upon further investigation, it is revealed that a property of the domain of SCT problems is responsible. Almost all problems, and every difficult problem, in this experiment have SCGs whose vertices have an in-degree of at most 1. This property was first observed by Ben-Amram and Lee in their analysis of SCT complexity [2]. After showing why this property explains the performance of Ramsey-based algorithms, we explore why this property emerges and argue that it is a reasonable property for SCT problems to possess. Finally, we improve the rank-based algorithm for problems with this property.

As stated above, all interesting SCGs in this experiment have vertices with at most one incoming edge. In analogy to the corresponding property for automaton, we call this property of SCGs *reverse-determinism*. Given a set of reverse-deterministic SCGs $\mathcal{G}$, we observe three consequences. First, a reverse-deterministic SCG can have no more than n arcs: one entering each vertex. Second, there are only $2^{O(n \log n)}$ possible such combinations of n arcs. Third, the composition of two reverse-deterministic SCGs is also reverse-deterministic. Therefore every element in the closure of $\mathcal{G}$ under composition is also reverse-deterministic. These observations imply that the closure of $\mathcal{G}$ under composition contains at most $2^{O(n \log n)}$ SCGs. This reduces the worst-case complexity of the LJB algorithm to $2^{O(n \log n)}$. In the presence of this property, the massive gap between Ramsey-based algorithms and rank-based algorithms vanishes, helping to explain the surprising strength of the LJB algorithm.

Lemma 7. *When operating on reverse-deterministic SCT problems, the LJB algorithm has a worst-case complexity of* $2^{O(n \log n)}$.

It is not a coincidence that all SCT problems considered possess this property. As noted in [2], straightforward analysis of functional programs generates only reverse-deterministic problems. In fact, every tool we examined is only capable of producing reverse-deterministic SCT problems. To illuminate the reason for this, imagine a SCG $G : f \rightarrow g$ where f has two parameters, x and y, and g the single parameter z. If G is not reverse deterministic, this implies both x and y have arcs, labeled with either 0 or 1, to z. This would mean that z's value is both *always* smaller than or equal to x and *always* smaller than or equal to y. In order for this to occur, we would need a min operation that returns the smaller of two elements. For the case of lists, for example, min would return the shorter of two lists. This is not a common operation, and none of the size-change analyzers were designed to discover such properties of functions.

We now consider the rank-based approach to see if it can benefit from reverse-determinism. We say that an automaton is *reverse-deterministic* when no state has two incoming arcs labeled with the same character. Formally, an automaton is reverse-deterministic when, for each state q and character a, there is at most one state p such that $q \in \rho(p, a)$. Given a reverse-deterministic SCT problem L, both $\mathcal{A}_{Flow(L)}$ and $\mathcal{A}_{Desc(L)}$ are reverse-deterministic. As a corollary to the above, the Ramsey-based complementation construction has a worst-case complexity of $2^{O(n \log n)}$ for reverse deterministic automata. Examining the rank-based approach, we note that with reverse-deterministic automata we do not have to worry about multiple paths to a state. Thus a maximum rank of 2, rather than $2n$, suffices to prove termination of every path, and the worst-case bound of the rank-based construction improves to $2^{O(n)}$.

Lemma 8. *Given a reverse-deterministic Büchi automaton $\mathcal{B}$ with n states, there exists an automaton $\mathcal{B}'$ with $2^{O(n)}$ states such that $L(\mathcal{B}') = \overline{L(\mathcal{B})}$.*

In light of this discovery, we revisit the experiments and again compare rank and Ramsey-based approaches on SCT problems. This time we tell Mh, the rank-based solver, that the problems have a maximum rank of 2. Table 2 compares the running time of Mh and sct/scp on the five most difficult problems. As before, time taken to reduce SCT problems to automata containment problems was not counted.

Problem	Mh (s)	sct/scp (s)
ex04	0.01	1.39
ex05	0.13	227.7
ms	0.1	0.02
gexgcd	0.39	0.023
graphcolour2	0.044	0.014

Table 2: SCT problem completion time times by tool, exploiting reverse-determinism.

While our problem space is small, the theoretical worst-case bounds of Ramsey and rank-based approach appears to be reflected in the table. The Ramsey-based sct/scp completes some problems more quickly, but in the worst cases, ex04 and ex05, performs significantly more slowly than Mh. It is worth noting, however, that the benefits of reverse-determinism on Ramsey-based approaches emerges automatically, while rank-based approaches must explicitly test for this property in order to exploit it.

6 Conclusion

In this paper we demonstrate that the Ramsey-based size-change termination algorithm proposed by Lee, Jones, and Ben-Amram [10] is a specialized realization of the 1987 Ramsey-based complementation construction [3, 14]. With this link established, we compare rank-based and Ramsey-based tools on the domain of SCT problems. Initial experimentation revealed a surprising competitiveness of the Ramsey-based tools, and led us to further investigation. By exploiting reverse-determinism, we were able to demonstrate the superiority of the rank-based approach.

Our experiments operated on a very sparse space of problem, and still yielded two interesting observations. First, subsumption appears to be critical to the performance of Büchi complementation tools using both rank and Ramsey-based algorithms. It has already been established that rank-based tools benefit strongly from the use of subsumption [5]. Our results demonstrate that Ramsey-based tools also benefit from subsumption, and in fact experiments with removing subsumption from sct/scp seem to limit its scalability. Second, by exploiting reverse-determinism, we can dramatically improve the performance of both rank and Ramsey-based approaches to containment checking.

Our test space was unfortunately small, with only five interesting problems emerging. In [5, 15], a space of random automata universality problems is used to provide a diverse problem domain. We plan to similarly generate a space of random SCT problems to provide a more informative problem space. Sampling this problem space is complicated by the low transition density of reverse-deterministic problems: in [5, 15] the most interesting problems had a transition density of 2. Intrigued by the competitive performance of Ramsey-based solutions, we also intend to compare Ramsey and rank-based approaches on the domain of random universality problems.

On the theoretical side, we are interested in extending the subsumption relation present in sct/scp. It is not immediately clear how to use subsumption for problems that are not strongly suffix-closed. While arbitrary problems can be phrased as a single-graph search, doing so imposes additional complexity. Extending the subsumption relation to the double-graph search of Lemma 4 would simplify this solution greatly.

The effects of reverse-determinism on the complementation of automata bear further study. Reverse-determinism is not an obscure property, it is known that automata derived from LTL formula are reverse-deterministic [6]. As noted above, both rank and Ramsey-based approaches improves exponentially when operating on reverse-deterministic automata. Further, Ben-Amram and Lee have defined SCP, a polynomial-time approximation algorithm for SCT. For a wide subset of SCT problems with restricted in degrees, including the set used in this paper, SCP is exact. In terms of automata, this property is similar, although perhaps not identical, to reverse-determinism. The presence of an exact polynomial algorithm for the SCT case suggests a interesting subset of Büchi containment problems may be solvable in polynomial time. The first step in this direction would be to determine what properties a containment problem must have to be solved in this fashion.

References

1. Daedalus. `http://www.di.ens.fr/~cousot/projects/DAEDALUS/`.
2. A.M. Ben-Amram and C. Lee. Program termination analysis in polynomial time. *ACM Trans. Program. Lang. Syst*, 29(1), 2007.
3. J.R. Büchi. On a decision method in restricted second order arithmetic. In *ICLMPS*, pages 1–12. Stanford University Press, 1962.
4. Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and Systems Science*, 8:117–141, 1974.
5. L. Doyen and J. Raskin. Improved algorithms for the automata-based approach to model-checking. In *TACAS*, pages 451–465, 2007.
6. A.E. Emerson and A.P. Sistla. Deciding full branching time logics. *Information and Control*, 61(3):175–201, 1984.
7. C.C. Frederiksen. A simple implementation of the size-change termination principle. *Tech. Rep. D-442*, DIKU, 2001.
8. A.J. Glenstrup. Terminator ii: Stopping partial evaluation of fully recursive programs. Master's thesis, DIKU, University of Copenhagen, June 1999.
9. O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak In *Transactions on Computational Logic*, pages 409–429, 2001
10. C. Lee, N.D. Jones, and A.M. Ben-Amram. The size-change principle for program termination. In *POPL*, pages 81–92, 2001.
11. M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.
12. D. Sereni and N.D. Jones. Termination analysis of higher-order functional programs. In *APLAS 2005*, 2005.
13. S. Safra On the Complexity of ω-Automata In *FOCS*, pages 319-327, 1988.
14. A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *Proc. 12th ICALP*, pages 217–237, 1985.
15. D. Tabakov and M.Y. Vardi. Experimental evaluation of classical automata constructions. In *Proc. 32nd LPAR*, volume 3835 of *LNCS*, pages 396–411, 2005.
16. M.Y. Vardi. Automata-theoretic model checking revisited. In *Proc. 8th VMCAI*, volume 4349 of *Lecture Notes in Computer Science*, pages 137–150. Springer, 2007.
17. D. Wahlstedt. Detecting termination using size-change in parameter values. Master's thesis, Göteborgs Universitet, 2000.