BÜCHI COMPLEMENTATION AND SIZE-CHANGE TERMINATION

SETH FOGARTY AND MOSHE Y. VARDI

Department of Computer Science, Rice University, Houston, TX
e-mail address: sfogarty@gmail.com

Department of Computer Science, Rice University, Houston, TX
e-mail address: 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. The Ramsey-based algorithm was presented as a more practical alternative to the automata-theoretic approach, but 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. To do so, we extend the Ramsey-based complementation construction to provide a containment-testing algorithm. Surprisingly, empirical analysis suggests that despite the massive gap in worst-case complexity, 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. 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. For these automata, the complementation problem is known to involve an exponential blowup [19]. Thus the most difficult step in checking containment is constructing the complementary automata $\overline{\mathcal{B}}$.

Key words and phrases: Büchi Complementation, Model Checking, Formal Verification, Automata, Büchi Automata.
Earlier version appeared in TACAS09.

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.

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

The first complementation constructions for nondeterministic Büchi automata employed a Ramsey-based combinatorial argument to partition the infinite set of infinite words into a finite set of regular languages. Proposed by Büchi in 1962 [4], this construction was shown in 1987 by Sistla, Vardi, and Wolper to be implementable with a blow-up of $2^{O(n^2)}$ [23]. 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 [19].

The gap was tightened in 1988, when Safra described a $2^{O(n \log n)}$ construction [20]. 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 [21], or improving the derived algorithms. In 2001, Kupferman and Vardi employed a rank-based analysis of Büchi automata to simplify complementation [18]. Recently, Doyen and Raskin have demonstrated the necessity of using a subsumption technique in the rank-based approach, providing a direct universality checker that scales to automata several orders of magnitude larger than previous tools [8].

Separately, in the context of of program termination analysis, Lee, Jones, and Ben-Amram presented the size-change termination (SCT) principle in 2001 [6]. 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 [23].

In this paper we show that the LJB algorithm for deciding SCT [6] is a specialized implementation of the 1987 Ramsey-based complementation construction [23]. Section 2 presents the background and notation for the paper. Section 3 expands the Ramsey-based complementation construction into a containment algorithm, and then presents the proof that the LJB algorithm is a specialized realization of this Ramsey-based containment algorithm. In Section 4, we 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 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 in Section 5, 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. Indeed, we find that reverse-deterministic SCT problems have a maximum rank of 2, collapsing the complexity of rank-based 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. While Section 6 concludes, we note here that the best-of-breed tools for both the Ramsey and rank based approaches employ subsumption to minimize the explored state space.

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 function. We lift the ρ function to sets of states and words of arbitrary length as follows. Given a set of states R, define $\rho(R, \sigma)$ to be $\bigcup_{q \in R} \rho(q, \sigma)$. Inductively, given a set of states R: let $\rho(R, \epsilon) = R$, and for every word $w = \sigma_0 ... \sigma_n$ let $\rho(R, w)$ be defined as $\rho(\rho(R, \sigma_0), \sigma_1 ... \sigma_n)$.

A *run* of a Büchi automaton $\mathcal{B}$ on a word $w \in \Sigma^\omega$ is a infinite sequence of states $r = q_0 q_1 ... \in Q^\omega$ 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 $i \in I\!N$. 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 $L(\mathcal{B})$. Correspondingly, a *path* in $\mathcal{B}$ from q to r on a word $w \in \Sigma^+$ is a finite sequence of states $r = q_0 ... q_n \in Q^+$ such that $q_0 = q$, $q_n = r$, and, for every $i \in \{0...n-1\}$, we have $q_{i+1} \in \rho(q_i, w_i)$. 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* [25]. 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 [5]. Thus, the most computationally demanding step is constructing the complement of $\mathcal{B}$. In the formal verification field, existing empirical work has focused on the simplest form of containment testing, *universality* testing, where $\mathcal{A}$ is the universal automaton [7, 24].

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 optimized implementation presented in 1987. To construct the complement of $\mathcal{B} = \langle \Sigma, Q, Q^{in}, \rho, F \rangle$ 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 $q, r \in Q$:
> (1) Is there a path in $\mathcal{B}$ from q to r over w?
> (2) If so, 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 every 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 2.1. 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.

Lemma 2.2. [4, 23]

(1) $L(\widetilde{g}),\ \widetilde{g} \in \widetilde{Q}_{\mathcal{B}}$, form a partition of Σ^+
(2) If $u \in L(\widetilde{g}),\ v \in L(\widetilde{h}),$ and $uv \in L(\widetilde{k}),$ then $L(\widetilde{g}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{k})$

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 $Y(\widetilde{g}, \widetilde{h})$ be the ω-language $L(\widetilde{g}) \cdot L(\widetilde{h})^\omega$. Say that a language $Y(\widetilde{g}, \widetilde{h})$ is *proper* if $Y(\widetilde{g}, \widetilde{h})$ 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 2.3. [4, 23] $\Sigma^\omega = \bigcup\{Y(\widetilde{g}, \widetilde{h}) \mid Y(\widetilde{g}, \widetilde{h}) \text{ is proper}\}$

Proof: The proof is based on Ramsey's Theorem. Consider an infinite word $w = \sigma_0\sigma_1...$ By Lemma 2.2, every prefix of the word w is in some graph $\widetilde{g}_i$. Let $k = 3^{n^2}$ be the number of graphs. Thus w defines a partition of $\mathbb{N}$ into k sets $D_1, ..., D_k$ such that $i \in D_l$ iff $\sigma_0...\sigma_{i-1} \in L(\widetilde{g}_l)$. Clearly there is some m such that D_m is infinite.

Similarly, by Lemma 2.2 we can use the the word w to define a partition of all unordered *pairs* of elements from D_m. This partition consists of k sets $C_1, ...C_k$, such that $\langle i, j\rangle \in C_l$ iff $\sigma_i...\sigma_{j-1} \in L(\widetilde{g}_l)$. Ramsey's Theorem tells us that, given such a partition, there exists an infinite subset $\{i_1, i_2, ...\}$ of D_m and a C_n such that for all pairs of distinct elements i_j, i_k, it holds that $\langle i_j, i_k\rangle \in C_n$.

This implies that the word w can be partitioned into

$$w_1 = \sigma_0...\sigma_{i_1-1}, \quad w_2 = \sigma_{i_1}...\sigma_{i_2-1}, \quad w_3 = \sigma_{i_2}...\sigma_{i_3-1}, \quad ...,$$

where $w_1 \in L(\widetilde{g}_m)$ and $w_i \in L(\widetilde{g}_n)$ for $i > 1$. As $\sigma_0...\sigma_{i_j-1} \in L(\widetilde{g}_m)$ for every i_j, we have that $w_1w_2 \in L(\widetilde{g}_m)$. As $\sigma_{i_j}...\sigma_{i_k-1} \in L(\widetilde{g}_n)$ for every pair i_j, i_k, we have that $w_2w_3 \in L(\widetilde{g}_n)$. By Lemma 2.2, it follows that $L(\widetilde{g}_m) \cdot L(\widetilde{g}_n) \subseteq L(\widetilde{g}_m)$, and that $L(\widetilde{g}_n) \cdot L(\widetilde{g}_n) \subseteq L(\widetilde{g}_n)$, and thus $Y(\widetilde{g_m}, \widetilde{g_n})$ is proper. □

Further, Sistla et al. prove that each proper language is entirely contained or entirely disjoint from $L(\mathcal{B})$. This provides a way to construct the complement of $L(\mathcal{B})$: take the union every proper language that is disjoint from $L(\mathcal{B})$.

Lemma 2.4. [4, 23]

(1) For $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$, either $Y(\widetilde{g}, \widetilde{h}) \cap L(\mathcal{B}) = \emptyset$ or $Y(\widetilde{g}, \widetilde{h}) \subseteq L(\mathcal{B})$.

(2) $\overline{L(\mathcal{B})} = \bigcup\{Y(\widetilde{g}, \widetilde{h}) \mid Y(\widetilde{g}, \widetilde{h}) \text{ is proper and } Y(\widetilde{g}, \widetilde{h}) \cap L(\mathcal{B}) = \emptyset\}$

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 $\overline{\mathcal{B}}$ [23]. 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 $\mathcal{B}$. However, we can avoid an explicit lasso search 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 $Y(\widetilde{g}, \widetilde{h})$ is proper. If $Y(\widetilde{g}, \widetilde{h})$ 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 $Y(\widetilde{g}, \widetilde{h})$ is so contained.

Lemma 2.5. [23] Given an Büchi automaton $\mathcal{B}$ and the set of graphs $\widetilde{Q}_{\mathcal{B}}$,

(1) $\mathcal{B}$ is universal iff for every proper $Y(\widetilde{g}, \widetilde{h})$, it holds that $Y(\widetilde{g}, \widetilde{h}) \subseteq L(\mathcal{B})$.

(2) Let $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$ be two graphs where $Y(\widetilde{g}, \widetilde{h})$ is proper. $Y(\widetilde{g}, \widetilde{h}) \subseteq L(\mathcal{B})$ 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 2.5 yields a PSPACE algorithm to determine universality [23]. Simply check each $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$. If $Y(\widetilde{g}, \widetilde{h})$ 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, first introduced in [16], uses a notion of ranks to track the progress of each possible run towards fair termination. Consider a Büchi automaton $\mathcal{B} = \langle \Sigma, Q, Q_\in, \rho, Q_f \rangle$ and an infinite word $w = \omega_0\omega_1...$. The runs of $\mathcal{B}$ on w can be arranged in an infinite DAG (directed acyclic graph), $G_w = \langle V, E \rangle$, where

- $V \subseteq Q \times \mathbb{N}$ is such that $\langle q, l \rangle \in V$ iff some run r of $\mathcal{B}$ on w has $r(l) = q$.
- $E \subseteq \bigcup_{l \geq 0}(Q \times \{l\}) \times (Q \times \{l+1\})$ is $E(\langle q, l \rangle, \langle q', l+1 \rangle)$ iff $\langle q, l \rangle \in V$ and $q' \in \rho(q, \omega_l)$.

G_w, called the *run DAG* of $\mathcal{B}$ on w, exactly embodies all possible runs of $\mathcal{B}$ on w. We define a run DAG G_w to be accepting when there exists a path in G_w with infinitely many states in F. This path corresponds to an accepting run of $\mathcal{B}$ on w. When G_w is not accepting, we say it is a rejecting run DAG. Say that a node $\langle q, i \rangle$ of a graph is *finite* if it has only finitely many descendants, that it is accepting if $q \in Q_f$, and that it is F*-free* if it is not accepting and does not have accepting descendants.

Given a run DAG G_w, we inductively define a sequence of subgraphs by eliminating nodes that cannot be part of accepting runs. A node that is finite can clearly not be part of an infinite run, much less an accepting infinite run. Similarly, a node that is F-free may be part of an infinite run, but this infinite run does not contain accepting states, much less an infinite number of them.

- $G_w(0) = G_w$
- $G_w(2i+1) = G_w(2i) \setminus \{\langle q, l \rangle \mid \langle q, l \rangle$ is finite in $G_w(2i)\}$
- $G_w(2i+2) = G_w(2i+1) \setminus \{\langle q, l \rangle \mid \langle q, l \rangle$ is F-free in $G_w(2i)\}$

If the final graph a node appears in in $G_w(i)$, we say that node is of rank i. If the rank of every node is at most i, we say G_w has a rank of i. $\mathcal{B}$ has a maximum rank of i when every rejecting run DAG of $\mathcal{B}$ has a maximum rank of i. Note that nodes with odd ranks are removed because they are F-free. Therefore no accepting state can have an odd rank. [16] prove that the maximum rank of a rejecting run DAG for every automaton is bounded by $2n-2$. This allows us to create an automaton that guesses the rejecting run DAG of $\mathcal{B}$ as it proceeds along the word.

A *level ranking* for an automaton $\mathcal{B}$ with n states is a function $f : Q \to \{0...2n-2, \bot\}$, 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.

Definition 2.6. 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

- $f_{in}(q) = 2n-2$ 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 2.7. [17] For every Büchi automaton $\mathcal{B}$, $L(KV(\mathcal{B})) = \overline{L(\mathcal{B})}$.

This automaton tracks the progress of $\mathcal{B}$ along a word $w = \sigma_0\sigma_1...$ by attempting to find an infinite series of level rankings $f_0 f_1...$. We start with the most general possible level ranking, and ensure that every rank f_i covers f_{i+1} under σ_i. Every run has a non-increasing rank, and so must eventually become trapped in some rank. To accept a word, the automaton requires that each run visit an odd rank infinitely often. Recall that accepting states cannot be assigned an odd rank. Thus,

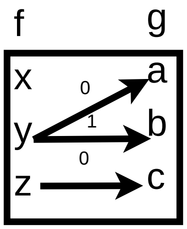

Figure 1: Size-Change Graphs: A size-change graph for a call from function f to function g.

for a word rejected by $\mathcal{B}$, every run can eventually become trapped in an odd rank. Conversely, if there is an accepting run that visits an accepting node infinitely often, that run cannot visit an odd rank infinitely often and the complementary automaton rejects it.

An algorithm seeking to refute the universality of $\mathcal{B}$ can look for a lasso in the state-space of the rank-based complement of $\mathcal{B}$. A classical approach is Emerson-Lei backward-traversal nested fixpoint $\nu Y.\mu X.(X \cup (Y \cap F))$ [10]. This nested fixpoint employs the observation that a state in a lasso can reach an arbitrary number of accepting states. The outer fixpoint iteratively computes sets $Y_0, Y_1, ...$ such that Y_i contains all states with a path visiting i accepting states. Universality is checked by testing if Y_∞, the set of all states with a path visiting arbitrarily many accepting states, intersects Q^{in}. The strongest algorithm implementing this approach, from Doyen and Raskin, takes advantage of the presence of a subsumption relation in the rank-based 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 sets in the Emerson-Lei approach, it is sufficient to store only the maximal elements under this relation. Further, the predecessor operation for 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 [7].

2.3. **Size-Change Termination**

In [6] 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 an excellent discussion of the abstraction of functional language semantics, refer to [15]. We consider here a set H of functions, and denote the parameters of a function f by $P(f)$.

Definition 2.8. 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 $x \xrightarrow{0} y$.

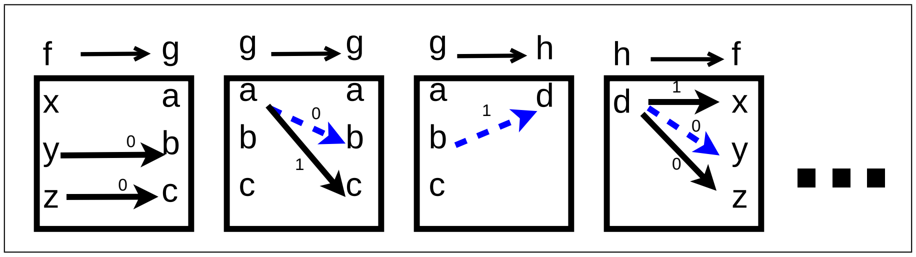

Figure 2: The dotted line forms a prefix of a late-start thread in a call sequence.

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 a well-founded value decreases infinitely often, regardless of when this decrease begins. Therefore threads can begin in arbitrary function calls, in arbitrary parameters. We call this the *late-start property* of SCT problems, and provide an example in Figure 2.3. We revisit this property in Section 3.3.

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. [6] 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 $L(\mathcal{A}_{Desc(L)}) = Desc(L)$.

[1] The implementation provided by Lee et al. [6] also make this assumption, and in the presence of unreachable functions size-change termination may be undetectable.

Definition 2.9.[2]
$\mathcal{A}_{Flow(L)} = \langle C, H, H, \rho_F, H\rangle$, where

- $\rho_F(f_1, c) = \{f_2 \mid c : f_1 \to f_2\}$

$\mathcal{A}_{Desc(L)} = \langle C, Q_p \cup H, Q_p \cup H, \rho_D, F\rangle$, where

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

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 of $\mathcal{G}$ under the composition operation, called S, is then searched for a counterexample describing an infinite call sequence with no infinitely descending thread.

Definition 2.10. 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:

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

Using composition, we can focus on a subset of graphs. Say that a graph $G : f \to f$ is *idempotent* when $G = G; G$. Each idempotent graph describes a cycle in the call graph, and each cycle in the call graph can be accounted for by at least one idempotent graph.

Algorithm LJB searches for a counterexample to size-change termination. First, it iteratively 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, the algorithm check every $G : f_1 \to f_1 \in S$ to ensure that if G is idempotent, then G has an associated thread with infinitely many 1-labeled arcs. This thread is represented by an arc of the form $x \xrightarrow{1} x$.

Theorem 2.11, whose proof uses a Ramsey-based argument, demonstrates the correctness of Algorithm LJB in determining the size-change termination of an SCT problem $L = \langle H, P, C, \mathcal{G}\rangle$.

Theorem 2.11. [6] 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 an idempotent SCG graph $G : f \to f$ that does *not* contain an arc of the form $x \xrightarrow{1} x$.

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 remarkable similarity. In this section we bridge the gap between the Ramsey-based universality test and the LJB

[2] The original LJB construction [6] restricted starting states in $\mathcal{A}_{Desc(L)}$to functions. This was changed to simplify Section 3.4. The modification does not change the accepted language.

Algorithm 1: `LJB` ($\langle H, P, C, \mathcal{G} \rangle$)

```
Data: A size-change termination problem $\langle H, P, C, \mathcal{G} \rangle$.
Result: Whether or not the problem is size-change terminating.
Initialize $\mathsf{S} \Leftarrow \mathcal{G}$
for all pairs $G : f \rightarrow g$, $G' : g \rightarrow h$ in $\mathsf{S}$ do
    $G'' : f \rightarrow h \Leftarrow G; G$
    Add $G''$ to $\mathsf{S}$
    if $f = h$ and $G''; G'' = G''$ then
        if there does not exist an arc of the form $x \xrightarrow{1} x$ in $G''$ then
            return Not Terminating
return Terminating
```

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 2.5). 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 2.3). 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, $L(\bar{g})$.

Definition 3.1. Given $w \in \Sigma^+$, 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 $\langle q, a, r \rangle \in \widetilde{g}$. Since the language of each graph consists of finite words, we employ the concatenation of languages to characterize infinite runs. To do so, we first prove Lemma 3.2, which simplifies the concatenation of entire languages by demonstrating an equivalence to the concatenation of arbitrary words from these languages.

Lemma 3.2. If $u \in L(\widehat{g}),\ v \in L(\widehat{h}),\ uv \in L(\widehat{k})$, and $L(\bar{g}) \cdot L(\bar{h}) \subseteq L(\bar{k})$, then $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{k})$

Proof: Assume we have such an u and v. We demonstrate every word $w \in L(\widehat{g}) \cdot L(\widehat{h})$ must be in $L(\widehat{k})$. If we expand the premise, we obtain $w \in (L(\bar{g}) \cap L(\widetilde{g})) \cdot (L(\bar{h}) \cap L(\widetilde{h}))$. This implies w must be in $L(\bar{g}) \cdot L(\bar{h})$ and in $L(\widetilde{g}) \cdot L(\widetilde{h})$. Next, we know that $u \in L(\widetilde{g}),\ v \in L(\widetilde{h})$, and $uv \in L(\widetilde{k})$. Thus by Lemma 2.2, $L(\widetilde{g}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{k})$, and $w \in L(\widetilde{k})$. Along with the premise $L(\bar{g}) \cdot L(\bar{h}) \subseteq L(\bar{k})$, we can now conclude $w \in L(\bar{k}) \cap L(\widetilde{k})$, which is $L(\widehat{k})$. □

The languages $L(\widehat{g}),\ \widehat{g} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$, cover all finite subwords of $L(\mathcal{A})$. A subword of $L(\mathcal{A})$ has at least one path between two states in Q_A, and thus is in the language of an arc in $\bar{Q}_{\mathcal{A}}$. Further, this word is described by some graph, and the pair of the arc and the graph makes a supergraph. Unlike the case of graphs and Σ^+, the languages of supergraphs do not form a partition of $L(\mathcal{A})$: a word might have multiple paths between states in $\mathcal{A}$, and so be described by more than one arc in $\bar{Q}_{\mathcal{A}}$. 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 $Z(\widehat{g}, \widehat{h})$ be the ω-language $L(\widehat{g}) \cdot L(\widehat{h})^{\omega}$. $Z(\widehat{g}, \widehat{h})$ is called *proper* if: (1) $Z(\widehat{g}, \widehat{h})$ 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 $L(\widehat{h}) \cdot L(\widehat{h}) \subseteq L(\widehat{h})$. Call a pair of supergraphs $\langle \widehat{g}, \widehat{h} \rangle$ proper if $Z(\widehat{g}, \widehat{h})$ is proper. We note that $Z(\widehat{g}, \widehat{h})$ is non-empty if $L(\widehat{g})$ and $L(\widehat{h})$ are non-empty, and that, by the second condition, every proper $Z(\widehat{g}, \widehat{h})$ is contained in $L(\mathcal{A})$.

Lemma 3.3. Let $\mathcal{A}$ and $\mathcal{B}$ be two Büchi automata. $L(\mathcal{A}) = \bigcup \{ Z(\widehat{g}, \widehat{h}) \mid Z(\widehat{g}, \widehat{h}) \text{ is proper} \}$.

Proof: We extend the Ramsey argument of Lemma 2.3 to supergraphs.

Consider an infinite word $w = \sigma_0 \sigma_1 ...$ with an accepting run $p = p_0 p_1 ...$ in $\mathcal{A}$. As p is accepting, we know that $p_0 \in Q^{in}_{\mathcal{A}}$ and $p_i \in F_{\mathcal{A}}$ for infinitely many i. Since $F_{\mathcal{A}}$ is finite, at least one accepting state q must appear infinitely often. Let $D \subseteq I\!N$ be the set of indexes i such that $p_i = q$.

We pause to observe that, by the definition of the languages of arcs, for every $i \in D$ the word $\sigma_0 ... \sigma_{i-1}$ is in $L(\langle p_0, q \rangle)$, and for every $i, j \in D,\ i < j$, the word $\sigma_i ... \sigma_{j-1} \in L(\langle q, q \rangle)$. Every language $Z(\widehat{h}, \widehat{k})$ where $\bar{h} = \langle p_0, q \rangle$ and $\bar{h} = \langle q, q \rangle$ thus satisfies the second requirement of properness.

In addition to restricting our attention the subset of nodes where $p_i = q$, we further partition D into $k = 3^{n^2}$ sets $D_1, ..., D_k$ based on the prefix of w until that point, where there is a D_l associated with each possible graph $\widetilde{g}_l$. By Lemma 2.2, every finite word is in the language of some graph $\widetilde{g}$. Say that $i \in D_l$ iff $\sigma_0 ... \sigma_{i-1} \in L(\widetilde{g}_l)$. As k is finite, for some m D_m must be infinite. Let $\widetilde{h} = \widetilde{g}_m$.

Similarly, by Lemma 2.2 we can use the the word w to define a partition of all unordered *pairs* of elements from D_m. This partition consists of k sets $C_1, ... C_k$, such that $\{i, j\} \in C_l$ iff $\sigma_i ... \sigma_{j-1} \in L(\widetilde{g}_l)$. Ramsey's Theorem tells us that, given such a partition, there exists an infinite subset $\{i_1, i_2, ...\}$ of D_m and a C_n such that $\{i_j, i_k\} \in C_n$ for all pairs of distinct elements i_j, i_k.

This is precisely to say there is a graph $\widetilde{k}$ so that, for every $i_j, i_k \in C_n$, it holds that $\sigma_{i_j} ... \sigma_{i_k - 1} \in L(\widetilde{k})$. C_n thus partitions the word w into

$$w_1 = \sigma_0 ... \sigma_{i_1 - 1}, \quad w_2 = \sigma_{i_1} ... \sigma_{i_2 - 1}, \quad w_3 = \sigma_{i_2} ... \sigma_{i_3 - 1}, \quad ...,$$

such that $w_1 \in L(\widetilde{h})$ and $w_i \in L(\widetilde{k})$ for $i > 1$. Let $\widehat{h} = \langle \langle p_0, q \rangle, \widetilde{h} \rangle$ and let $\widehat{k} = \langle \langle q, q \rangle, \widetilde{k} \rangle$. By the above partition of w, we know that $w \in Z(\widehat{h}, \widehat{k})$. We now show that $Z(\widehat{h}, \widehat{k})$ is proper.

First, as $w \in Z(\widehat{h}, \widehat{k})$, we know $Z(\widehat{h}, \widehat{k})$ is non-empty. Second, as noted above, the second requirement is satisfied by the arcs $\langle p_0, q \rangle$ and $\langle q, q \rangle$. Finally, we demonstrate the third condition holds. First, as $\sigma_0 ... \sigma_{i-1} \in L(\widetilde{h})$ for every $i \in C_n$, we have that $w_1 w_2 \in L(\widetilde{h})$. Both w_1 and $w_1 w_2$ are in $L(\langle p_0, q \rangle)$ and so $w_1,\ w_1 w_2 \in L(\widehat{h})$. By the definition of the language of arcs, $L(\langle p_0, q \rangle) \cdot L(\langle q, q \rangle) \subseteq L(\langle p_0, q \rangle)$. Thus by Lemma 3.2, we can conclude that $L(\widehat{h}) \cdot L(\widehat{k}) \subseteq L(\widehat{h})$. Second, $\sigma_i ... \sigma_{j-1} \in L(\widetilde{k})$ for every pair $i, j \in C_n$, we have that $w_2 w_3 \in L(\widetilde{k})$. As $w_2,\ w_2 w_3$ are both in

$\langle q,q\rangle$, it holds that $w_2,\ w_2w_3 \in L(\widehat{k})$. By the definition of the language of arcs, $L(\langle q,q\rangle)\cdot L(\langle q,q\rangle) \subseteq L(\langle q,q\rangle)$. By Lemma 3.2 we can now conclude $L(\widehat{k}) \cdot L(\widehat{k}) \subseteq L(\widehat{k})$. Therefore $Z(\widehat{h},\widehat{k})$ is a proper language containing w. □

Lemma 3.4. Let $\mathcal{A}$ and $\mathcal{B}$ be two Büchi automata, and $\widehat{Q}_{\mathcal{A},\mathcal{B}}$ the corresponding set of supergraphs.

(1) For all proper $Z(\widehat{g},\widehat{h})$, either $Z(\widehat{g},\widehat{h}) \cap L(\mathcal{B}) = \emptyset$ or $Z(\widehat{g},\widehat{h}) \subseteq L(\mathcal{B})$.
(2) $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff every proper language $Z(\widehat{g},\widehat{h}) \subseteq L(\mathcal{B})$.
(3) Let $\widehat{g},\widehat{h}$ be two supergraphs such that $Z(\widehat{g},\widehat{h})$ is proper. $Z(\widehat{g},\widehat{h}) \subseteq L(\mathcal{B})$ 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 $\langle r,1,r\rangle \in \widehat{h}$.

Proof: Given two supergraphs $\widehat{g} = \langle \bar{g},\widetilde{g}\rangle$ and $\widehat{h} = \langle \bar{h},\widetilde{h}\rangle$, recall that $Y(\widetilde{g},\widetilde{h})$ is the ω-language $L(\widetilde{g}) \cdot L(\widetilde{h})^{\omega}$. Further note that $L(\widehat{g}) \subseteq L(\widetilde{g})$ and $L(\widehat{h}) \subseteq L(\widetilde{h})$, and therefore $Z(\widehat{g},\widehat{h}) \subseteq Y(\widetilde{g},\widetilde{h})$.

(1): Consider two supergraphs $\widehat{g},\widehat{h}$. By Lemma 2.4 either $Y(\widetilde{g},\widetilde{h}) \cap L(\mathcal{B}) = \emptyset$ or $Y(\widetilde{g},\widetilde{h}) \subseteq L(\mathcal{B})$. Since $Z(\widehat{g},\widehat{h}) \subseteq Y(\widetilde{g},\widetilde{h})$, it holds that $Z(\widehat{g},\widehat{h}) \cap L(\mathcal{B}) = \emptyset$ or $Z(\widehat{g},\widehat{h}) \subseteq L(\mathcal{B})$.

(2): Immediate from Lemma 3.3 and clause (1).

(3): By Lemma 2.4 either $Y(\widetilde{g},\widetilde{h}) \subseteq L(\mathcal{B})$ or $Y(\widetilde{g},\widetilde{h}) \cap L(\mathcal{B}) = \emptyset$. By Lemma 2.5 $Y(\widetilde{g},\widetilde{h}) \subseteq L(\mathcal{B})$ iff a q,r and a exist such that $\langle q,a,r\rangle \in \widetilde{g}$ and $\langle r,1,r\rangle \in \widetilde{h}$. Since $Z(\widehat{g},\widehat{h}) \subseteq Y(\widetilde{g},\widetilde{h})$, $Z(\widehat{g},\widehat{h}) \subseteq L(\mathcal{B})$ iff such a q,r and a exist. □

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 $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

3.2. Composition of Supergraphs

The double-graph search faces difficulty on two fronts. First, the number of supergraphs is very large. Second, checking language nonemptiness and containment are exponentially difficult problems. 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. Along the way we develop a polynomial-time test for the containment of supergraph languages. Our plan is to 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. We begin by defining the composition of simple graphs.

Definition 3.5. Given two graphs $\widetilde{g}$ and $\widetilde{h}$ define their composition, written as $\widetilde{g};\widetilde{h}$, as the graph

$$\begin{aligned}
&\{\langle q,1,r\rangle \mid q,r,s \in Q_{\mathcal{B}},\ \langle q,b,s\rangle \in \widetilde{g},\ \langle s,c,r\rangle \in \widetilde{h},\ b=1 \text{ or } c=1\}\\
\cup\ &\{\langle q,0,r\rangle \mid q,r,s \in Q_{\mathcal{B}},\ \langle q,0,s\rangle \in \widetilde{g},\ \langle s,0,r\rangle \in \widetilde{h},\ \text{and}\\
&\forall t \in Q_{\mathcal{B}},\ b,c \in \{0,1\}\ .\ \langle q,a,t\rangle \in \widetilde{g} \wedge \langle t,b,r\rangle \in \widetilde{h} \text{ implies } a=b=0\}
\end{aligned}$$

We can then define the composition of two supergraphs $\widehat{g} = \langle\langle q,r\rangle,\widetilde{g}\rangle$, $\widehat{h} = \langle\langle r,s\rangle,\widetilde{h}\rangle$, written $\widehat{g};\widehat{h}$, as the supergraph $\langle\langle q,s\rangle,\widetilde{g};\widetilde{h}\rangle$. To generate exactly the set of supergraphs with non-empty languages, we start with supergraphs describing single letters For a containment problem $L(\mathcal{A}) \subseteq$

$L(\mathcal{B})$, define the subset of $\widehat{Q}_{\mathcal{A},\mathcal{B}}$ corresponding to single letters to be $\widehat{Q}^1_{\mathcal{A},\mathcal{B}} = \{\widehat{g} \mid \widehat{g} \in \widehat{Q}_{\mathcal{A},\mathcal{B}},\ a \in \Sigma,\ a \in L(\widehat{g})\}$. For completeness, we present a constructive definition of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$.

Definition 3.6.

$$\begin{aligned} \widehat{Q}^1_{\mathcal{A},\mathcal{B}} = \{\langle\langle q,r\rangle,\widetilde{g}\rangle \quad \mid \quad & q \in Q_{\mathcal{A}}, r \in \rho_{\mathcal{A}}(q,a),\ a \in \Sigma, \\ & \widetilde{g} = \{\langle q',0,r'\rangle \mid q' \in Q_{\mathcal{B}} \setminus F_{\mathcal{B}},\ r' \in (\rho_{\mathcal{B}}(q',a) \setminus F_{\mathcal{B}})\} \cup \\ & \{\langle q',1,r'\rangle \mid q' \in Q_{\mathcal{B}},\ r' \in \rho_{\mathcal{B}}(q',a), q' \text{ or } r' \in F_{\mathcal{B}})\}\} \end{aligned}$$

Algorithm DoubleGraphSearch, which we prove correct below, employs composition to check the containment of two automata. It first generates the set of initial supergraphs, and then computes the closure of this set under composition. Along the way it tests properness by using composition. Every time we it encounters a proper pair of supergraphs, it either verifies that a satisfying pair of arcs exist, or halts with a counterexample to containment. We call this search the *double-graph search*.

Algorithm 2: DoubleGraphSearch($\mathcal{A}$,$\mathcal{B}$)

```
Data: Two Büchi automata, $\mathcal{A}$ and $\mathcal{B}$.
Result: Whether $L(\mathcal{A})$ is contained in $L(\mathcal{B})$.
Initialize $\widehat{Q}^f \Leftarrow \widehat{Q}^1_{\mathcal{A},\mathcal{B}}$
for all pairs $\widehat{g}, \widehat{h} \in \widehat{Q}^f$ where $\bar{g} = \langle q,r\rangle$ and $\bar{h} = \langle r,s\rangle$ do
    Add $\widehat{g};\widehat{h}$ to $\widehat{Q}^f$
    if $q \in Q^{in}_{\mathcal{A}},\ r \in F_{\mathcal{A}},\ s = r,\ \widetilde{g};\widetilde{h} = \widetilde{g}$ and $\widetilde{h};\widetilde{h} = \widetilde{h}$ then
        if there do not exist $\langle q,a,r\rangle \in \widetilde{g}$ and $\langle r,1,r\rangle \in \widetilde{h}$ where $q \in Q^{in}_{\mathcal{B}}$ then
            return Not Contained
return Contained
```

To begin proving our algorithm correct, we link composition and the concatenation of languages, first for simple graphs and then more strongly for supergraphs.

Lemma 3.7. For every two graphs $\widetilde{g}$ and $\widetilde{h}$, it holds that $L(\widetilde{g}) \cdot L(\widetilde{h}) \subseteq L(\widetilde{g};\widetilde{h})$.

Proof: Consider two words $w_1 \in L(\widetilde{g}),\ w_2 \in L(\widetilde{h})$. By Definition 2.1, to prove $w_1w_2 \in L(\widetilde{g};\widetilde{h})$ we must show that for every $q, r \in Q$: both (1) $\langle q,a,r\rangle \in \widetilde{g};\widetilde{h}$ iff there is a path from q to r over w_1w_2, and (2) that $a = 1$ iff there is an accepting path.

If an arc $\langle q,a,r\rangle \in \widetilde{g};\widetilde{h}$ exists, then there is an $s \in Q$ such that $\langle q,b,s\rangle \in \widetilde{g}$ and $\langle s,c,r\rangle \in \widetilde{h}$. By Definition 2.1, this implies the existence of a path x_1s from q to s over w_1, and a path sx_2 from s to r over w_2. Thus x_1sx_2 is a path from q to r over w_1w_2.

If a is 1, then either b or c must be 1. By Definition 2.1, b (resp., c) is 1 iff there is an accepting path x'_1s (resp., sx'_2) over w_1 (resp.,w_2) from q to s (resp., s to r). In this case x'_1sx_2 (resp., $x_1sx'_2$) is an accepting path in $\mathcal{B}$ from q to r over w_1w_2.

Symmetrically, if there is a path x from q to r over w_1w_2, then after reading w_1 we are in some state s and have split x into x_1sx_2, so that x_1s is a path from q to s and sx_2 a path from s to r. Thus by Definition 2.1 $\langle q,b,s\rangle \in \widetilde{g}$, $\langle s,c,r\rangle \in \widetilde{h}$, and $\langle q,a,s\rangle \in \widetilde{g};\widetilde{h}$.

Further, if there is an accepting path from q to r over w_1w_2, then after reading w_1 we are in some state s and have split the path into x_1sx_2, so that x_1s is a path from q to s, and sx_2 a path from s to r. Either x_1s or sx_2 must be accepting, and thus by Definition 2.1 $\langle q,b,s\rangle \in \widetilde{g}$, $\langle s,c,r\rangle \in \widetilde{h}$, and either b or c must be 1. Therefore a must be 1. □

Lemma 3.8. Let $\widehat{g}, \widehat{h}, \widehat{k}$ be supergraphs in $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ such that $\bar{g} = \langle q, r\rangle$, $\bar{h} = \langle r, s\rangle$, and $\bar{k} = \langle q, s\rangle$. Then $\widehat{g}; \widehat{h} = \widehat{k}$ iff $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{k})$.

Proof: Assume $\widehat{g}; \widehat{h} = \widehat{k}$ as a premise. This implies $\widehat{k} = \langle\langle q, s\rangle, \widetilde{g}; \widetilde{h}\rangle)$. Take two words $u \in L(\widehat{g})$, $v \in L(\widehat{h})$. By construction, $u \in L(\langle q, r\rangle)$ and $v \in L(\langle r, s\rangle)$. The definition of the languages of arcs therefore implies the existence of a path from q to r over u and a path from r to s over v. Thus $uv \in L(\langle q, s\rangle)$. Similarly, $u \in L(\widetilde{g})$, $v \in L(\widetilde{h})$, and Lemma 3.7 implies that $uv \in L(\widetilde{k})$. Thus uv is in $L(\langle\langle q, r\rangle, \widetilde{k}\rangle)$. and by Lemma 3.2 $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{k})$.

In the other direction, if $L(\widehat{g}) \cdot L(\widehat{h}) \subseteq L(\widehat{k})$, we show that $\widehat{g}; \widehat{h} = \widehat{k}$. By definition, $\widehat{g}; \widehat{h}$ is $\langle\langle q, s\rangle, \widetilde{g}; \widetilde{h}\rangle$. As $\widehat{g}, \widehat{h} \in \widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ and is therefore non-empty, there is a word $w \in L(\widehat{g}) \cdot L(\widehat{h})$. This expands to $w \in (L(\bar{g}) \cap L(\widetilde{g})) \cdot (L(\bar{h}) \cap L(\widetilde{h}))$, which implies $w \in L(\widetilde{g}) \cdot L(\widetilde{h})$. By Lemma 3.7, w is then in $L(\widetilde{g}; \widetilde{h})$. Since, by Lemma 2.2, w is the language of exactly one graph, we have that $\widetilde{g}; \widetilde{h} = \widetilde{k}$, which proves $\widehat{g}; \widehat{h} = \widehat{k}$. □

Lemma 3.8 provides the polynomial time test for properness employed in Algorithm `DoubleGraphSearch`. Namely, given two supergraphs $\widehat{g} = \langle\langle q, r\rangle, \widetilde{g}\rangle$ and $\widehat{h} = \langle\langle r, r\rangle, \widetilde{h}\rangle$ from $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$, the pair $\langle \widehat{g}, \widehat{h}\rangle$ is proper exactly when $q \in Q^{in}_{\mathcal{A}}$, $r \in F_{\mathcal{A}}$, $\widetilde{g}; \widetilde{h} = \widetilde{g}$ and $\widetilde{h}; \widetilde{h} = \widetilde{h}$. We can also provide the final piece of our puzzle: proving that the closure of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ under composition contains every non-empty supergraph.

Lemma 3.9. For two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, every $\widehat{h} \in \widehat{Q}_{\mathcal{A},\mathcal{B}}$, where $L(\widehat{h}) \neq \emptyset$, is in the closure of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ under composition.

Proof: Let $\widehat{h} = \langle\langle q, r\rangle, \widetilde{h}\rangle$ where $L(\widehat{h}) \neq \emptyset$. Then there is at least one word$w = \sigma_0 ... \sigma_{n-1} \in L(\widehat{h})$, which is to say $w \in L(\langle q, r\rangle) \cap L(\widetilde{h})$. By the definition of the languages of arcs, there is a path $p = p_0 ... p_n$ in $\mathcal{A}$ over w such that $p_0 = q$ and $p_n = r$.

Define $\widetilde{g}_{\sigma_i}$ to be the graph in $\widetilde{Q}^1_{\mathcal{B}}$ containing σ_i. Let $\widehat{g}_{\sigma_i}$ be $\langle\langle p_i, p_{i+1}\rangle, \widetilde{g}_{\sigma_i}\rangle$, and let $\widehat{g}_w$ be $\widehat{g}_{\sigma_0}; \widehat{g}_{\sigma_1}; ...; \widehat{g}_{\sigma_{n-1}}$. Note that each $\widehat{g}_{\sigma_i} \in \widehat{Q}^1_{\mathcal{A},\mathcal{B}}$. By Lemma 3.8 $w \in \widetilde{g}_w$. By Lemma 2.2, w is in only one graph and $\widetilde{g}_w = \widetilde{h}$. By construction, $\bar{g}_w = \langle q, r\rangle$. Therefore $\langle \bar{g}_w, \widetilde{g}_w\rangle = \langle\langle q, r\rangle, \widetilde{h}\rangle = \widehat{h}$, and $\widehat{h}$ is in the closure of $\widehat{Q}^1_{\mathcal{A},\mathcal{B}}$ under composition. □

Lemma 3.10. Let $\mathcal{A}$ and $\mathcal{B}$ be two Büchi automata. $L(\mathcal{A})$ is *not* contained in $L(\mathcal{B})$ iff $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ contains a pair of supergraphs $\widehat{g}$, $\widehat{h}$ such that $\langle \widehat{g}, \widehat{h}\rangle$ is proper and there do *not* exist arcs $\langle q, a, r\rangle \in \widehat{g}$ and $\langle r, 1, r\rangle \in \widehat{h}$, $q \in Q^{in}_{\mathcal{B}}$.

Proof: As all proper graphs are non-empty, this follows from parts (2) and (3) of Lemma 3.4 and Lemma 3.9. □

We can now show the correctness of Algorithm `DoubleGraphSearch`, using Lemma 3.8 to justify testing properness with composition, and Lemma 3.10 to justify the correctness and completeness of our search for a counterexample.

Theorem 3.11. For every two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, it holds that $L(\mathcal{A}) \subseteq L(\mathcal{B})$ iff `DoubleGraphSearch(`$\mathcal{A}$,$\mathcal{B}$`)` returns Contained.

Proof: By Lemma 3.8, testing for composition is equivalent to testing for language containment, and the outer conditional in Algorithm `DoubleGraphSearch` holds only for proper pairs of supergraphs. By Lemma 3.10, the inner conditional checks if a proper pair of supergraphs is a counterexample, and if no such proper pair in $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ is a a counterexample then containment most hold. □

3.3. Strongly Suffix Closed Languages

Algorithm `DoubleGraphSearch` has much the same structure as Algorithm `LJB`. The most noticeable difference is that Algorithm `DoubleGraphSearch` checks pairs of supergraphs, where Algorithm `LJB` checks only single size-change graphs. Indeed, Theorem 2.11 suggests that, for some languages, a cycle implies the existence of a lasso. When proving containment of Büchi automata with such languages, it is sufficient 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 arbitrary points. We here define the class of automata amenable to this optimization, first presenting the case for universality testing, without proof, for clarity.

In size-change termination, the late-start property asserts that an accepting cycle can start at an arbitrary point. Intuitively, this suggests that an arc $\langle r, 1, r\rangle \in \widetilde{h}$ might not need a matching prefix $\langle q, a, r\rangle$ in some $\widetilde{g}$: the cycle can just start at r. 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 3.12. 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 3.13. 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 $\widetilde{h}$ such that $\widetilde{h};\widetilde{h}$ and $\widetilde{h}$ does *not* contain an arc of the form $\langle r, 1, r\rangle$.

As an intuition for the correctness of Lemma 3.13, note that the existence of an 1-labeled cyclic arc in $\widetilde{h}$ implies a loop, that Q being reachable implies a prefix can be prepended to this loop to make a lasso, and that strong suffix closure allows us to swap this prefix for the prefix of every other word that share this cycle.

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 3.14. 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 $w_1 w \in L_2$.

When checking the containment of $\mathcal{A}$ in $\mathcal{B}$ for the case when $L(\mathcal{B})$ is strongly suffix closed with respect to $L(\mathcal{A})$, we can employ a the simplified algorithm below. As in Algorithm `DoubleGraphSearch`, we search all supergraphs in $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$. Rather than searching for a proper pair of supergraphs, however, Algorithm `SingleGraphSearch` searches for a single supergraph $\widehat{h}$ where $\widehat{h};\widehat{h} = \widehat{h}$ that does not contain an arc of the form $\langle r, 1, r\rangle$. We call this search the *single-graph search*.

We now prove Algorithm `SingleGraphSearch` correct. Theorem 3.15 demonstrates that, under the requirements specified, the presence of a single-graph counterexample refutes containment, and the absence of such a supergraph proves containment.

```
Algorithm 3: SingleGraphSearch($\mathcal{A}$,$\mathcal{B}$)
  Data: Two Büchi automata, $\mathcal{A}$ and $\mathcal{B}$.
  Require $Q^{in}_{\mathcal{A}} = Q_{\mathcal{A}}$, $Q_{\mathcal{B}}$ is reachable, and $L(\mathcal{B})$ is strongly suffix closed w.r.t. $L(\mathcal{A})$
  Result: Whether $L(\mathcal{A})$ is contained in $L(\mathcal{B})$.
  Initialize $\widehat{Q}^f \Leftarrow \widehat{Q}^1_{\mathcal{A},\mathcal{B}}$
  for allpairs $\widehat{g}, \widehat{h} \in \widehat{Q}^f$ where $\bar{g} = \langle q, r\rangle$ and $\bar{h} = \langle r, s\rangle$ do
      $\widehat{k} \Leftarrow \widehat{g}; \widehat{h}$
      Add $\widehat{k}$ to $\widehat{Q}^f$
      if $q \in F_{\mathcal{A}}$, $q = s$, and $\widehat{k}; \widehat{k} = \widehat{k}$ then
          if there does not exist an arc $\langle r, 1, r\rangle \in \widehat{k}$ then
              return Not Contained
  return Contained
```

Theorem 3.15. Let $\mathcal{A}$ and $\mathcal{B}$ be two Büchi automata where $Q^{in}_{\mathcal{A}} = Q_{\mathcal{A}}$, every state in $Q_{\mathcal{B}}$ is reachable, and $L(\mathcal{B})$ is strongly suffix closed with respect to $L(\mathcal{A})$. Then $L(\mathcal{A})$ is *not* contained in $L(\mathcal{B})$ iff $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ contains a supergraph $\widehat{h} = \langle\langle s, s\rangle, \widetilde{h}\rangle$ such that $s \in F_{\mathcal{A}}$, $\widehat{h}; \widehat{h} = \widehat{h}$ and $\widehat{h}$ does *not* contain an arc $\langle r, 1, r\rangle$.

Proof: In one direction, assume $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ 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 $\langle r, 1, r\rangle \in \widehat{h}$. We show that $Z(\widehat{h}, \widehat{h})$ is a proper language not contained in $L(\mathcal{B})$. As $\widehat{h} \in \widehat{Q}^f_{\mathcal{A},\mathcal{B}}$, we know $L(\widehat{h})$ is not empty, implying $Z(\widehat{h}, \widehat{h})$ is non-empty. As $Q_{\mathcal{A}} = Q^{in}_{\mathcal{A}}$, it holds that $s \in Q^{in}_{\mathcal{A}}$. By Lemma 3.8, the premise $\widehat{h}; \widehat{h} = \widehat{h}$ implies $L(\widehat{h}) \cdot L(\widehat{h}) \subseteq L(\widehat{h})$, and $Z(\widehat{h}, \widehat{h})$ is proper. Finally, as there is no $\langle r, 1, r\rangle \in \widehat{h}$, by Theorem 3.11, $Z(\widehat{h}, \widehat{h}) \not\subseteq L(B)$, and $Z(\widehat{h}, \widehat{h})$ is a counterexample to $L(\mathcal{A}) \subseteq L(\mathcal{B})$.

In the opposite direction, assume the premise that $\widehat{Q}^f_{\mathcal{A},\mathcal{B}}$ does not contain 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 $\langle r, 1, r\rangle \in \widehat{h}$. We prove that every word $w \in L(\mathcal{A})$ is also in $L(\mathcal{B})$. Take a word $w \in L(\mathcal{A})$. By Lemma 3.3, w is in some proper language $Z(\widehat{g}, \widehat{h})$ and can be broken into $w_1 w_2$ where $w_1 \in L(\widehat{g}),\ w_2 \in L(\widehat{h})^\omega$.

Because $Z(\widehat{g}, \widehat{h})$ is proper, Lemma 3.8 implies $\widehat{h} = \langle\langle s, s\rangle, \widetilde{h}\rangle$ where $s \in F_{\mathcal{A}}$ and $\widehat{h}; \widehat{h} = \widehat{h}$. This, along with our premise, implies $\widehat{h}$ contains an arc $\langle r, 1, r\rangle$. Since all states in $Q_{\mathcal{B}}$ are reachable, there is $q \in Q^{in}_{\mathcal{B}}$ and $u \in \Sigma^+$ with a path in $\mathcal{B}$ from q to r over u. By Lemma 2.5, this implies uw_2 is accepted by $\mathcal{B}$. For $L(\mathcal{B})$ to strongly suffix closed with respect to $L(\mathcal{A})$, it must be suffix closed. Therefore $w_2 \in L(\mathcal{B})$. Now we move to $L(\mathcal{A})$, and note that the premise $Q^{in}_{\mathcal{A}} = Q_{\mathcal{A}}$ implies $L(\mathcal{A})$ is suffix closed. Thus the fact that $w_1 w_2 \in L(\mathcal{A})$ implies $w_2 \in L(\mathcal{A})$. Since $L(\mathcal{B})$ is strongly suffix closed with respect to $L(\mathcal{A})$, and $w_2 \in L(\mathcal{B})$, it must be that $w_1 w_2 \in L(\mathcal{B})$. □

3.4. **From Ramsey-Based Containment to Size-Change Termination**

We now delve into the connection between the LJB algorithm for size-change termination and the single-graph search algorithm for Büchi containment. We will show that Algorithm LJB is a specialized realization of Algorithm SingleGraphSearch. Given an SCT problem L, size-change graphs in LJB(L) are direct analogues of supergraphs in

`SingleGraphSearch(`$\mathcal{A}_{Flow(L)}$, $\mathcal{A}_{Desc(L)}$`)`. For convenience, take $L = \langle H, P, C, \mathcal{G}\rangle$, $\mathcal{A}_{Flow(L)} = \langle C, Q_{Fl}, Q^{in}_{Fl}, \rho_{Fl}, F_{Fl}\rangle$, and $\mathcal{A}_{Desc(L)} = \langle C, Q_{Ds}, Q^{in}_{Ds}, \rho_{Ds}, F_{Ds}\rangle$.

We first show that $\mathcal{A}_{Flow(L)}$, $\mathcal{A}_{Desc(L)}$ meet the preconditions of Algorithm `SingleGraphSearch`: that $Q^{in}_{Fl} = Q_{Fl}$; that every state in Q_{Ds} is reachable; and that $Desc(L)$ is strongly suffix closed with respect to $Flow(L)$. For the first and second requirement, it suffices to observe that every state in both $\mathcal{A}_{Flow(L)}$ and $\mathcal{A}_{Desc(L)}$ is initial.[3]

For the third, strong suffix closure is a direct consequence of the definition of a thread: since a thread can start at arbitrary points, it does not matter what call path we use to reach that point. Adding a prefix to a call path cannot cause that call path to become non-terminating. Thus the late-start property is precisely $Desc(L)$ being strongly suffix closed with respect to $Flow(L)$, and we can employ the single-graph search.

Consider supergraphs in $\widehat{Q}_{\mathcal{A}_{Flow(L)},\mathcal{A}_{Desc(L)}}$, from here simply denoted by $\widehat{Q}_L$. 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_p, the set of all $\{0,1\}$-labeled parameters. A supergraph in $\widehat{Q}_L$ thus comprises an arc $\langle q, r\rangle$ in H and a $\{0,1\}$-labeled graph $\widetilde{g}$ over $H \cup Q_p$. 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 asserting the existence of a call path between two functions, and a $\{0,1\}$-labeled graph. There are nodes in both graphs that correspond to parameters of functions, and arcs between two such nodes describe a thread between the corresponding parameters. The analogy falls short, however, on three points:

(1) In SCGs, nodes are always parameters of functions. In supergraphs, nodes can be either parameters of functions or function names.
(2) In SCGs, nodes are unlabeled. In supergraphs, nodes are labeled either 0 or 1.
(3) In an SCG, only the nodes corresponding to the parameters of two specific functions are present. In a supergraph, nodes corresponding to every parameter of every function exist.

Each difference is an opportunity to specialize the Ramsey-based containment algorithm, Algorithm `SingleGraphSearch`, by simplifying supergraphs. When these specializations are taken together, we have Algorithm `LJB`.

(1) No functions in H are accepting for $\mathcal{A}_{Desc(L)}$, and once we transition out of H into Q_p we can never return to H. Therefore nodes r corresponding to function names can never be part of a descending arc $\langle r, 1, r\rangle$. Since we only search for arcs of the form $\langle r, 1, r\rangle$, we can simplify supergraphs in $\widehat{Q}_L$ by removing all nodes 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 $\widehat{Q}_L$ by removing labels on parameters.
(3) While all parameters have corresponding states in $\mathcal{A}_{Desc(L)}$, each supergraph describes threads in a call sequence between two particular 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 $\widehat{Q}_L$ by removing all nodes corresponding to parameters of other functions.

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

To formalize this notion of simplification, we first define, $\widehat{G}_L$, the set of simplified supergraphs and show that $\widehat{G}_L$ is in one-to-one correspondence with S, the closure of $\mathcal{G}$ under composition.

Definition 3.16. $\widehat{G}_L = \{\langle\langle f_1, f_2\rangle, \widetilde{k}\rangle \mid f_1, f_2 \in H,\ \widetilde{k} \subseteq 2^{P(f_1)\times\{0,1\}\times P(f_2)}\}$

Say that $\langle r, \widetilde{g}\rangle \in \widehat{Q}_L$ *simplifies to* $\langle r, \widetilde{k}\rangle \in \widehat{G}_L$ when $\langle q, b, r\rangle \in \widetilde{k}$ iff there exists $a, c \in \{0,1\}$ such that $\langle\langle q, a\rangle, b, \langle r, c\rangle\rangle \in \widetilde{g}$. Let $\widehat{G}^1_L$ be $\{\widehat{k} \mid \widehat{k} \in \widehat{G}_L,\ \widehat{g} \in \widehat{Q}^1_L,\ \widehat{g}$ simplifies to $\widehat{k}\}$, and $\widehat{G}^f_L$ be the closure of $\widehat{G}^1_L$ under composition.

We can map SCGs directly to elements of $\widehat{G}_L$. Say $G : f_1 \to f_2 \equiv \langle\langle f_1, f_2\rangle, \widetilde{g}\rangle$ when $q \xrightarrow{a} r \in G$ iff $\langle q, a, r\rangle \in \widetilde{g}$. Note that the composition operations for supergraphs of this form is identical to the composition of SCGs: if $G_1 \equiv \widehat{g}$ and $G_2 \equiv \widehat{h}$, then $G_1; G_2 \equiv \widehat{g}; \widehat{h}$. Therefore every element of $\widehat{Q}^f_L$ simplifies to some element of $\widehat{G}^f_L$.

We now show that supergraphs whose languages contain single characters are in one-to-one correspondence with $\mathcal{G}$, and that every idempotent element of $\widehat{G}^f_L$ contains an arc of the form$\langle r, 1, r\rangle$ exactly when the closure of $\mathcal{G}$ under composition does not contain a counterexample graph.

Lemma 3.17. Let $L = \langle H, P, C, \mathcal{G}\rangle$ be an SCT problem.

(1) The $\equiv$ relation is a one-to-one correspondence between $\widehat{G}^1_L$ and $\mathcal{G}$

(2) L is *not* size-change terminating iff $\widehat{G}^f_L$ contains a supergraph $\widehat{k}$ such that $\widehat{k}; \widehat{k} = \widehat{k}$ and there does *not* exist an arc of the form $\langle r, 1, r\rangle$ in $\widehat{k}$.

Proof:

(1): Given a size-change graph $G \in \mathcal{G}$, we construct a unique supergraph $\widehat{k} \in \widehat{G}^1$ such that $\widehat{k} \equiv G$. Every size-change graph $G : f_1 \to f_2 \in \mathcal{G}$ is the SCG for a call site c from f_1 to f_2. This is a call sequence of length one. Thus there is a $\widehat{g} \in \widehat{Q}^1_L$ so that $c \in L(\widehat{g})$ and $\bar{g} = \langle f_1, f_2\rangle$. We show that the simplification of $\widehat{g}$ is equivalent to G. By the reduction of Definition 2.9 and the definition of graphs in Definition 2.1, the arc $\langle\langle q, b\rangle, a, \langle r, c\rangle\rangle \in \widehat{g}$, for some $b, c \in \{0,1\}$, exactly when $q \xrightarrow{a} r \in G$. The supergraph $\widehat{g}$ simplifies to some $\widehat{k} \in \widehat{G}_L$. By the definition of simplification, $\langle\langle q, b\rangle, a, \langle r, c\rangle\rangle \in \widehat{g}$ exactly when $\langle q, a, r\rangle \in \widehat{k}$. Thus $\widehat{k} \equiv G : f_1 \to f_2$.

In the other direction, $\widehat{k} \in \widehat{G}^1$ iff there exists a $\widehat{g} = \langle\langle f_1, f_2\rangle, \widetilde{g}\rangle \in \widehat{Q}^1_L$ that simplifies to $\widehat{k}$. By Definition 3.6, which defines $\widehat{Q}^1_L$ and Definition 2.9, $\widehat{g}$ exists because there is a call site c. This call site corresponds to a SCG $G : f_1 \to f_2$. Analogously to the above, by Definition 2.9 the arcs in $\widetilde{g}$ between parameters correspond to arcs in G: $\langle\langle q, b\rangle, a, \langle r, c\rangle\rangle \in \widetilde{g}$, for some $b, c \in \{0,1\}$, exactly when $q \xrightarrow{a} r \in G$. These are the only arcs that remain after simplification, during which the labels are removed. Thus $\widehat{k} \equiv G : f_1 \to f_2$.

(2):

By (1), $\mathcal{G}$ is in one-to-one correspondence with $\widehat{G}^1_L$ under the $\equiv$ relation. Since composition of supergraphs and SCGs is identical, S, the closure of $\mathcal{G}$ under composition, is in one-to-one correspondence with the $\widehat{G}^f_L$. Claim (3) then follows from Theorem 2.11 and claim (2). □

In conclusion, 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 Theorem 3.15 we know that $Flow(L) \subseteq Desc(L)$ if and only if $\widehat{Q}_L$ contains an idempotent graph $\widehat{g} = \widehat{g}; \widehat{g}$ with no arc of the form $\langle r, 1, r\rangle$. Thus we can employ the single-graph search instead of the double-graph search. Secondly, we can simplify supergraphs in $\widehat{Q}_L$ by removing the labels on nodes and keeping only nodes associated with appropriate parameters for the source and target function. 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, which is in one-to-one correspondence with $\mathcal{G}$, under composition does not contain an idempotent supergraph with no arc of the form $\langle r, 1, r\rangle$. This is precisely Algorithm LJB.

4. Empirical Analysis

All the Ramsey-based algorithms presented in Section 2.3 have worst-case running times that are exponentially larger 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. To facilitate a fair comparison, we briefly describe two improvements to the algorithms presented above.

4.1. Towards an Empirical Comparison

First, in constructing the analogy between SCGs in the LJB algorithm and supergraphs in the Ramsey-based containment algorithm, we noticed that supergraphs contain nodes for every parameter, while SCGs contain only nodes corresponding to parameters of relevant functions. These nodes 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 node to be unique to a specific parameter.

These extra states emerge because $Desc(L)$ only accepts strings that are contained in $Flow(L)$, and in doing so demands that parameters only be reached by appropriate call paths. 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. Further, we can rely on $Flow(L)$ to verify the sequence of function calls before our accepting thread and make do with a single waiting state.

Definition 4.1. Given an SCT problem $L = \langle H, P, C, \mathcal{G}\rangle$ and a projection Ar of all parameters onto their positions $1..n$ in the argument list, define:

$$\begin{array}{rcl} \mathcal{A}'_{Desc(L)} & = & \langle C, S \cup \{q_0\}, S \cup \{q_0\}, \rho_D, F\rangle \\ & \text{where} & S = \{1..n\} \times \{1, 0\} \\ & & \rho_D(q_0, c) = \{q_0\} \cup \{\langle Ar(x), 0\rangle \mid c : f_1 \to f_2,\ x \in P(f_2)\} \\ & & \rho_D(\langle h, a\rangle, c) = \{\langle Ar(y), a'\rangle \mid x \xrightarrow{a'} y \in \mathcal{G}_c,\ h = Ar(x)\} \\ & & F = \{1..n\} \times \{1\} \end{array}$$

Lemma 4.2. $L(\mathcal{A}_{Flow(L)}) \subseteq L(\mathcal{A}_{Desc(L)})$ iff $L(\mathcal{A}_{Flow(L)}) \subseteq L(\mathcal{A}'_{Desc(L)})$

Proof: The languages of $\mathcal{A}_{Desc(L)}$ and $\mathcal{A}'_{Desc(L)}$ are not the same. What we demonstrate is that for every word in $Flow(L)$, we can convert an accepting run in one of $\mathcal{A}_{Desc(L)}$ or $\mathcal{A}'_{Desc(L)}$ into an accepting run in the other. Recall that the states of $\mathcal{A}_{Flow(L)}$ are functions $f \in H$. States of $\mathcal{A}_{Desc(L)}$ are either elements of H or elements of $Q_p = \bigcup_{f \in H} P(f) \times \{1, 0\}$, the set of labeled parameters. For convenience, given a pair $\langle x, a\rangle \in Q_p$, define $Ar(\langle x, a\rangle)$ to be $\langle Ar(x), a\rangle$.

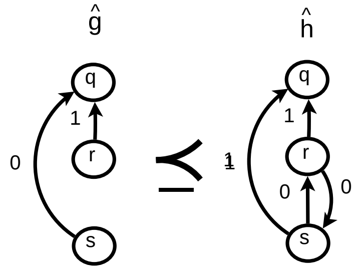

Figure 3: Subsumption: two graphs $\widetilde{g}$ and $\widetilde{h}$, where $\widetilde{g} \preceq \widetilde{h}$.

Consider an accepting run $r = r_0 r_1 ...$ of $\mathcal{A}_{Desc(L)}$ over a word w. Let $s = s_0 s_1 ...$ be the sequence of states in $\mathcal{A}'_{Desc(L)}$ such that when $r_i \in H$, $s_i = q_0$, and when $r_i \in Q_p$, $s_i = Ar(r_i)$. By the definition of $\mathcal{A}'_{Desc(L)}$, q_0 always transitions to q_0 and a transition between r_i and r_{i+1} implies a transition between $Ar(r_i)$ and $Ar(r_{i+1})$. Therefor s is a run of $\mathcal{A}'_{Desc(L)}$ over w. Further, if $\langle x, a\rangle$ is an accepting state in $\mathcal{A}_{Desc(L)}$, $a = 1$ and $\langle Ar(x), a\rangle$ is an accepting state $\mathcal{A}'_{Desc(L)}$. Thus, s is an accepting run of $\mathcal{A}'_{Desc(L)}$ over w.

Conversely, consider a word w with an accepting run $r = r_0 r_1 ...$ of $\mathcal{A}'_{Desc(L)}$ and an accepting run $s = s_0 s_1 ...$ of $\mathcal{A}_{Flow(L)}$. We define an accepting run $t = t_0 t_1 ..$ of $\mathcal{A}_{Desc(L)}$ on w. Each t_i depends on the corresponding r_i and s_i. If $r_i = q_0$ and $s_i = f$, then $t_i = f$. If $r_i = \langle k, a\rangle$ and $s_i = f$, then $t_i = \langle x, a\rangle$ where x is the kth parameter in f's argument list.

For a call $c : f_1 \to f_2$, take two labeled parameters, q a labeled parameter of f_1 and r a labeled parameter of f_2. If $\langle Ar(q), c, Ar(r)\rangle$ is a transition in $\mathcal{A}'_{Desc(L)}$, then $\langle q, c, r\rangle$ is a transition in $\mathcal{A}_{Desc(L)}$. Therefore t is a run of $\mathcal{A}_{Desc(L)}$ on w. Further, note that $\langle x, a\rangle \in F_{\mathcal{A}_{Desc(L)}}$ and $\langle Ar(x), a\rangle \in F_{\mathcal{A}_{Desc(L)}}$ iff $a = 1$. Therefore t is an accepting run. □

Second, in [3], 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 Algorithm `SingleGraphSearch` can also employ subsumption. When computing the closure of a set of graphs under compositions, we can ignore elements when they are approximated by other elements. Intuitively, a graph $\widetilde{g}$ approximates another graph $\widetilde{h}$ when it is strictly harder to find a 1-labeled sequence of arcs through $\widetilde{g}$ than through $\widetilde{h}$. If we can replace $\widetilde{h}$ with $\widetilde{g}$ without losing arcs, we do not have to consider $\widetilde{h}$. When the right arc can be found in $\widetilde{g}$, then it also occurs in $\widetilde{h}$. On the other hand, when $\widetilde{g}$ does not have a satisfying arc, then we already have a counterexample.

Formally, given two graphs $\widetilde{g}, \widetilde{h} \in \widetilde{Q}_{\mathcal{B}}$ say that $\widetilde{g}$ *approximates* $\widetilde{h}$, written $\widetilde{g} \preceq \widetilde{h}$, when for every arc $\langle q, a, r\rangle \in \widetilde{g}$ there is an arc $\langle q, a', r\rangle \in \widetilde{h}$, $a \leq a'$. An example is provided in Figure 4.1. Note that approximation is a transitive relation. In order to safely employ approximation as a subsumption relation, Ben-Amram and Lee replace the search for a single arc in idempotent graphs with a search for a strongly connected component in all graphs. This was proven to be safe in [11].

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 [7], 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 2.11, written in Haskell [13]. We have extended SCTP to reduce SCT problems to Büchi containment problems, using either Definition 2.9 or 4.1. **sct/scp** is an optimized C implementation of the SCT algorithm, which uses the subsumption relation of Section 4.1 [3].

Problem Space: Existing experiments on the practicality of SCT solvers focus on examples extracted from the literature [3]. We combine examples from a variety of sources [1, 3, 13, 14, 6, 22, 26]. 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 [3]	1.58	Time Out	1.39
ex05 [3]	Time Out	Time Out	227.7
ms [13]	Time Out	0.1	0.02
gexgcd [13]	0.55	14.98	0.023
graphcolour2 [14]	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 nodes have an in-degree of at most 1. This property was first observed by Ben-Amram and Lee in their analysis of SCT complexity [3]. After showing how 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 nodes with at most one incoming edge. In analogy to the corresponding property for automaton, we call this property of SCGs *reverse-determinism*. Say that a SCG is reverse-deterministic if every parameter of f_2 has at most one incoming edge. 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 node. 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 5.1. When operating on reverse-deterministic SCT problems, the `LJB` algorithm has a worst-case complexity of $2^{O(n \log n)}$.

Proof: A reverse-deterministic SCT problem contains only reverse-deterministic size-change graphs. Observe that the composition of two reverse-deterministic SCGs is itself reverse-deterministic. As there are only $2^{O(n \log n)}$ possible reverse-deterministic SCGs, the closure computed in the LJB algorithm cannot become larger than $2^{O(n \log n)}$. The LJB algorithm checks each graph in the closure exactly once, and so has a time complexity of $2^{O(n \log n)}$. □

It is not a coincidence that all SCT problems considered possess this property. As noted in [3], 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 $G : f \to g$ 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.

The program in Algorithm 4 can produce non-reverse-deterministic size-change graphs, and serves to demonstrate the difficult analysis required to do so[4]. Consider the SCG for the call on line 2. It is clear there should be a 0-labeled arc from X to X. To reach this point, however, we must satisfy the inequality on line 1. Therefore we can also assert that $Y > X$, and include a 1-labeled arc from Y to X. This is a kind of analysis is difficult to make, and none of the size-change analyzers we examined were capable of detecting this relation.

[4] This example emerges from the Terminweb experiments by Mike Codish, and was translated into a functional language by Amir Ben-Amram and Chin Soon Lee. The authors are grateful to Amir Ben-Amram for bringing this illustrative example to our attention.

Algorithm 4: `gcd(`X`,`Y`)`

```
if $Y > X$ then
    gcd($X$, $Y - X$)
else if $X > Y$ then
    gcd($X - Y$,$Y$)
else return $X$
```

5.1. Reverse Determinism and Rank-Based Containment

Since the Ramsey-based approach benefited so strongly from reverse-determinism, we examine the rank-based approach to see if it can similarly benefit. As a first step, we demonstrate that reverse-deterministic automata have a maximum rank of 2, dramatically lowering the complexity of complementation to $2^{O(n)}$. We note, however, that given a reverse-deterministic SCT problem L, $\mathcal{A}_{Desc(L)}$ is *not* reverse-deterministic. Thus a separate proof is provided to demonstrate that the rank of the resulting automata is still bounded by 2.

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 $q \in \rho(p, a)$. As a corollary to Lemma 5.1, 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 $2n - 2$, suffices to prove termination of every path, and the worst-case bound of the rank-based construction improves to $2^{O(n)}$.

Theorem 5.2. Given a reverse-deterministic Büchi automaton $\mathcal{B}$ with n states, there exists an automaton $\mathcal{B}'$ with $2^{O(n)}$ states such that $L(\mathcal{B}') = \overline{L(\mathcal{B})}$.

Proof: In a run DAG G_w of a reverse-deterministic automaton, all nodes have only one predecessor. This implies the run DAG is a tree, and that the number of infinite paths grows monotonically and at some point stabilizes. Call this point k. We claim that this is sufficient to show that, if G_w is rejecting, then every infinite path has a point past which all accepting states reachable from that path are finite in G_w. To see this, observe that each infinite path eventually stops visiting accepting states. Let j be the last such point over all infinite paths, or k, whichever is greater. Past j, consider a branch off this path containing an accepting state. This branch cannot be a new infinite path, as the number of infinite paths is stable. This branch cannot lead to an existing infinite path, because that would violate reverse determinism. Therefor this path must be finite, and the accepting state is finite.

Recall that $G_w(0)$ is G_w, $G_w(1)$ is $G_w(0)$ with all finite nodes removed, $G_w(2)$ is $G_w(1)$ with all F-free nodes removed, and $G_w(3)$ is $G_w(2)$ with all finite nodes removed. Because there are no infinite accepting nodes past j, $G_w(1)$ has no accepting nodes at all past j. Thus every node past j is F-free in $G_w(1)$, and $G_w(2)$ has no nodes past j. Thus $G_w(3)$ is empty, and the DAG has a rank of at most 2. Thus the maximum rank of rejecting run DAG is 2, and the state space of the automaton in Definition 2.6 can be restricted to level rankings with no ranking larger than 2. □

Unfortunately, neither the reduction of Definition 2.9 nor the reduction of Definition 4.1 preserve reverse determinism, which is to say that given a reverse-deterministic SCT problem, they do not produce a reverse-deterministic Büchi containment problem. However, we can show that,

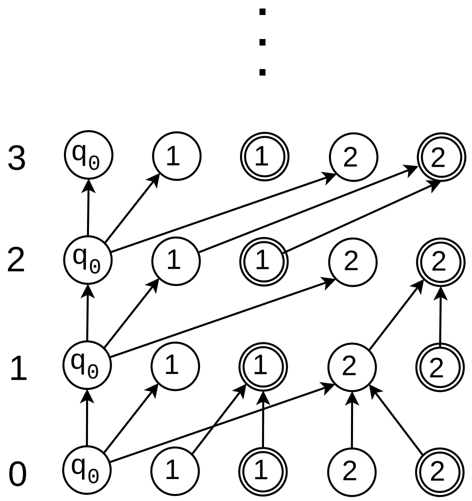

Figure 4: An overapproximation of the run DAG for $\mathcal{A}'_{Desc(L)}$ with maximum arity 2.

given a reverse-deterministic SCT problem, the automata produced by Definition 4.1 does have a maximum rank of 2. A similar claim could be made about Definition 2.9 with minor adjustments.

There are two kinds of states in $\mathcal{A}'_{Desc(L)}$. There is a waiting state, q_0, which always transitions to itself, and there are two states for every variable position $h \in 1..n$, $\langle h, 0\rangle$ and $\langle h, 1\rangle$. Every state is an initial state. Consider G_w, the run DAG of $\mathcal{A}'_{Desc(L)}$ on a word $c_0 c_1$.... Each character c_i represents a function call from some function f_i to another function f_{i+1}. At level i of the run DAG, the waiting state has outgoing edges to itself and the positions of 0-labeled parameters of f_{i+1}. Each variable state only has outgoing edges to a 0 or 1-labeled position. To get an idea of what the run DAG looks like, Figure 5.1 displays a supergraph of the run DAG that includes all states at all levels, even if they are not reachable. By definition, however, the true DAG has only nodes that can be reached from a node on the first level.

We now prove that, for every reverse-deterministic SCT problem L, $\mathcal{A}'_{Desc(L)}$ has a maximum rank of 2. Let w be an infinite word $c_1 c_2$... not in $L(\mathcal{A}'_{Desc(L)})$, and G_w the run DAG of $\mathcal{A}'_{Desc(L)}$ on w.

To demonstrate that G_w has a maximum rank of 2, we prove that there is a point $j \in \mathbb{N}$ past which all accepting states are finite in G_w. As above, this implies that $G_w(1)$ has no accepting states past j, $G_w(2)$ has no states at all past j, and $G_w(3)$ is empty.

Let G'_w be the subgraph of the run DAG that omits the waiting state q_0 at every level of the run DAG. Further, define G''_w to be the subgraph of G'_w containing only nodes with an incoming edge in G'_w. This removes nodes whose only incoming edge was from q_0, and thus removes accepting nodes which are not part of threads. While this also excludes nodes that begin threads, this cannot change the accepting or rejecting nature of the thread.

Lemma 5.3. For a level i, $f \in F$, and $x \in P(f)$, at most one of $(\langle Ar(x), 0\rangle, i)$ and $(\langle Ar(x), 1\rangle, i)$ has incoming edges in G'_w.

Proof: For $i = 0$, this holds trivially. For $i > 0$, take a pair of nodes $(\langle h, 0\rangle, i)$ and $(\langle h, 1\rangle, i)$. The edges from level $i - 1$ correspond to transitions in $\mathcal{A}'_{Desc(L)}$ on some call c. As c is a call to a single function, we know there is a unique variable x such that $h = Ar(x)$. Because L is reverse deterministic, we know that there is at most one edge in $\mathcal{G}_c$ leading to x. If there is no edge, then there are no edges entering $(\langle h, 0\rangle, i)$ or $(\langle h, 1\rangle, i)$.

Otherwise there is exactly one edge in $\mathcal{G}_c$, $y \overset{r}{\to} x$, $r \in \{0, 1\}$. In this case, the only nodes in level G'_w with an edge to either $(\langle h, 0\rangle, i)$ or $(\langle h, 1\rangle, i)$ are $(\langle Ar(y), 0\rangle, i-1)$ and $(\langle Ar(y), 1\rangle, i-1)$. By the transition function ρ_D, both of these states transition only to $(\langle h, r\rangle, i)$, and only $(\langle h, r\rangle, i)$ has incoming edges in G'_w. □

Lemma 5.4.

(1) G''_w is a forest.
(2) Every infinite path in G'_w appears in G''_w.
(3) Every accepting node in G_w is also in G''_w.

Proof:
(1): Lemma 5.3 implies, for every h and i, that only one of $(\langle h, 1\rangle, i)$ and $(\langle h, 0\rangle, i)$ is in G''_w. Combined with the fact that L is reverse deterministic, every node in G''_w can have at most one incoming edge, and thus it is a forest.
(2): For every infinite path in G'_w, all nodes past the first have an incoming edge from G'_w. Every node with an incoming edge from G'_w is in G''_w. Thus for every infinite path in G'_w, a corresponding path, perhaps without the first node, occurs in G''_w.
(3): Only nodes of the form $(\langle h, 1\rangle, i)$ are accepting. Let $(\langle h, 1\rangle, i)$ be a accepting node. By the definition of a run DAG, $(\langle h, 1\rangle, i)$ must be reachable if it is in G_w. Thus there is an edge from another node in G_w to $(\langle h, 1\rangle, i)$. By the transition function , the waiting state $(q_0, i-1)$ only has an edge to $(\langle h, 0\rangle, i)$. Therefore the only nodes that have an edge to $(\langle h, 1\rangle, i)$ are nodes of the form $(\langle h', r\rangle, i-1)$. All nodes of this form are in G'_w, and therefore $(\langle h, 1\rangle, i)$ has an incoming edge from G'_w, and is in G''_w. □

Lemma 5.5. There exists $j \in \mathbb{N}$ where for all $i > j$, G''_w has no infinite accepting nodes at level i.

Proof: As G''_w is a forest, the number of infinite nodes in level $i+1$ cannot be smaller than the number of infinite nodes in level i. Thus at some level k the number of infinite nodes reaches a maximum. Past level k, each infinite node has a unique infinite path through G''_w. As G_w is rejecting, every infinite path eventually stops visiting accepting nodes at some level. Let j be the last such point over all infinite paths, or k, whichever is greater. Past j, consider an accepting node v that branch off an infinite path. This branch cannot be part of the existing infinite path, as this path has ceased visiting accepting nodes. Likewise, this branch cannot be part of a new infinite path, as the number of infinite paths can not increase. Therefore v must be finite. □

Lemma 5.6. $G_w(3)$ is empty.

Proof: Let j be the level past which there are no infinite accepting nodes in G''_w, as per Lemma 5.5. This precisely means that, past j, every accepting node in G''_w has a finite path. As all accepting nodes in G_w are in G''_w, past j every reachable accepting node in G_w has a finite path. After level j, $G_w(1)$ contains only non-accepting nodes. This implies that $G_w(2)$ contains no nodes past j, and therefore that $G_w(3)$ is empty. □

Theorem 5.7. Given a reverse-deterministic SCT problem L with maximum arity n, there is an automaton $\mathcal{B}'$ with at most $2^{O(n)}$ states such that $L(\mathcal{B}') = \overline{L(\mathcal{A}'_{Desc(L)})}$.

Proof: By Lemma 5.6, we know that every rejecting run DAG of $\mathcal{A}'_{Desc(L)}$ has a maximum rank of 2. Therefor it suffices to restrict the rank in Definition 2.6 to 2, replacing all occurrences of $2n-2$ in the definition with 2. The resulting automata is simply three linked copies of the subset construction, and of size $2^{O(n)}$. □

5.2. **Experiments revisited**

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 of ex04 and ex05, sct/scp 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 [6] is a specialized realization of the 1987 Ramsey-based complementation construction [4, 23]. 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 [7]. 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 [7, 24], 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 [7, 24] 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. Recent work has extended the subsumption relation to the double-graph search of Algorithm `DoubleGraphSearch`, and improved the relation through the use of simulation [2, 12].

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 [9]. 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. Available on: `http://www.di.ens.fr/~cousot/projects/DAEDALUS/index.shtml`.

[2] Parosh Aziz Abdulla, Yu-Fang Chen, Lukás Holík, Richard Mayr, and Tomás Vojnar. When simulation meets antichains. In *TACAS*, pages 158–174, 2010.

[3] A.M. Ben-Amram and C.S. Lee. Program termination analysis in polynomial time. *TOPLAS*, 29(1), 2007.

[4] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.

[5] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and Systems Science*, 8:117–141, 1974.

[6] N.D. Jones C.S. Lee and A.M. Ben-Amram. The size-change principle for program termination. In *POPL*, pages 81–92, 2001.

[7] L. Doyen and J.-F. Raskin. Improved algorithms for the automata-based approach to model-checking. In *TACAS*, pages 451–465, 2007.

[8] Laurent Doyen and Jean-François Raskin. Antichains for the automata-based approach to model-checking. 5(1), 2009.

[9] A.E. Emerson and A.P. Sistla. Deciding full branching time logics. *Information and Control*, 61(3):175–201, 1984.

[10] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 267–278, 1986.

[11] S. Fogarty. Büchi containment and size-change termination. Master's thesis, Rice University, 2008.

[12] Seth Fogarty and Moshe Y. Vardi. Efficient büchi universality checking. In *TACAS*, pages 205–220, 2010.

[13] C.C. Frederiksen. A simple implementation of the size-change termination principle. 2001.

[14] Arne J. Glenstrup. Terminator ii: Stopping partial evaluation of fully recursive programs. Master's thesis, DIKU, University of Copenhagen, June 1999.

[15] N.D Jones and Nina Bohr. Termination analysis of the untyped λ-calculus. In V. van Oostrom, editor, *Rewriting Techniques and Applications. Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 1–23. Springer-Verlag, 2004.

[16] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symp. on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.

[17] O. Kupferman and M.Y. Vardi. Synthesizing distributed systems. In *Proc. 16th IEEE Symp. on Logic in Computer Science*, pages 389–398, 2001.

[18] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(2):408–429, 2001.

[19] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

[20] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.

[21] S. Schewe. Büchi complementation made tight. In *STACS*, pages 661–672, 2009.

[22] D. Sereni and N.D. Jones. Termination analysis of higher-order functional programs. In *APLAS 2005*, Lecture Notes in Computer Science, 2005.

[23] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *Proc. 12th Int. Colloq. on Automata, Languages, and Programming*, volume 194, pages 465–474. Springer, 1985.

[24] D. Tabakov and M.Y. Vardi. Experimental evaluation of classical automata constructions. In *Proc. 32nd Int. Colloq. on Automata, Languages, and Programming*, volume 3835 of *Lecture Notes in Computer Science*, pages 396–411. Springer, 2005.

[25] M.Y. Vardi. Automata-theoretic model checking revisited. In *Proc. 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 137–150. Springer, 2007.

[26] David Wahlstedt. Detecting termination using size-change in parameter values. Master's thesis, Göteborgs Universitet, 2000.