Advanced Ramsey-Based Büchi Automata Inclusion Testing*

Parosh Aziz Abdulla[1], Yu-Fang Chen[2], Lorenzo Clemente[3], Lukáš Holík[1,4],
Chih-Duo Hong[2], Richard Mayr[3], and Tomáš Vojnar[4]

[1] Uppsala University
[2] Academia Sinica
[3] University of Edinburgh
[4] Brno University of Technology

Abstract. Checking language inclusion between two nondeterministic Büchi automata $\mathcal{A}$ and $\mathcal{B}$ is computationally hard (PSPACE-complete). However, several approaches which are efficient in many practical cases have been proposed. We build on one of these, which is known as the *Ramsey-based approach*. It has recently been shown that the basic Ramsey-based approach can be drastically optimized by using powerful subsumption techniques, which allow one to prune the search-space when looking for counterexamples to inclusion. While previous works only used subsumption based on set inclusion or forward simulation on $\mathcal{A}$ and $\mathcal{B}$, we propose the following new techniques: (1) A larger subsumption relation based on a combination of backward and forward simulations on $\mathcal{A}$ and $\mathcal{B}$. (2) A method to additionally use forward simulation *between* $\mathcal{A}$ and $\mathcal{B}$. (3) Abstraction techniques that can speed up the computation and lead to early detection of counterexamples. The new algorithm was implemented and tested on automata derived from real-world model checking benchmarks, and on the Tabakov-Vardi random model, thus showing the usefulness of the proposed techniques.

1 Introduction

Checking inclusion between finite-state models is a central problem in automata theory. First, it is an intriguing theoretical problem. Second, it has many practical applications. For example, in the automata-based approach to model-checking [19], both the system and the specification are represented as finite-state automata, and the model-checking problem reduces to testing whether any behavior of the system is allowed by the specification, i.e., to a language inclusion problem.

We consider language inclusion for Büchi automata (BA), i.e., automata over infinite words. While checking language inclusion between nondeterministic BA is computationally hard (PSPACE-complete [13]), much effort has been devoted to devising approaches that can solve as many practical cases as possible. A naïve approach to

* This work was supported by Royal Society grant JP080268, the UPMARC project, the CONNECT project, National Science Council of Taiwan project no. 99-2218-E-001-002-MY3, the Czech Science Foundation (projects P103/10/0306 and 102/09/H042), the Czech Ministry of Education (projects COST OC10009 and MSM 0021630528), and the BUT FIT project FIT-S-11-1.

J.-P. Katoen and B. König (Eds.): CONCUR 2011, LNCS 6901, pp. 187–202, 2011.

language inclusion between BA $\mathcal{A}$ and $\mathcal{B}$ would first complement the latter into a BA $\mathcal{B}^c$, and then check emptiness of $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}^c)$. The problem is that $\mathcal{B}^c$ is in general exponentially larger than $\mathcal{B}$. Yet, one can determine whether $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}^c) \neq \emptyset$ by only looking at some "small" portion of $\mathcal{B}^c$. The *Ramsey-based approach* [16,9,10] gives a recipe for doing this. It is a descendant of Büchi's original BA complementation procedure, which uses the infinite Ramsey theorem in its correctness proof.

The essence of the Ramsey-based approach for checking language inclusion between $\mathcal{A}$ and $\mathcal{B}$ lies in the notion of *supergraph*, which is a data-structure representing a class of finite words sharing similar behavior in the two automata. Ramsey-based algorithms contain (i) an initialization phase where a set of supergraph seeds are identified, (ii) a search loop in which supergraphs are iteratively generated by composition with seeds, and (iii) a test operation where pairs of supergraphs are inspected for the existence of a counterexample. Intuitively, this counterexample has the form of an infinite ultimately periodic word $w_1(w_2)^\omega \in \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}^c)$, where one supergraph witnesses the prefix and the other the loop. While supergraphs themselves are small, and the test in (iii) can be done efficiently, the limiting factor in the basic algorithm lies in the exponential number of supergraphs that need to be generated. Therefore, a crucial challenge in the design of Ramsey-based algorithms is to limit the supergraphs explosion problem. This can be achieved by carefully designing certain *subsumption relations* [10,1], which allow one to safely discard subsumed supergraphs, thus reducing the search space. Moreover, methods based on minimizing supergraphs [1] by pruning their structure can further reduce the search space, and improve the complexity of (iii) above.

This paper contributes to the Ramsey-based approach to language inclusion in several ways. (1) We define a new subsumption relation based on both *forward* and *backward simulation* within the two automata. Our notion generalizes the subset-based subsumption of [10] and the forward simulation-based subsumption of [1]. (2) On a similar vein, we improve minimization of supergraphs by employing forward and backward simulation for minimizing supergraphs. (3) We introduce a method of exploiting forward simulation *between* the two automata, while previously only simulations internal to each automaton have been considered. (4) Finally, we provide a method to speed up the tests performed on supergraphs by grouping similar supergraphs together in a combined representation and extracting more abstract test-relevant information from it.

The correctness of the combined use of forward and backward simulation turns out to be far from trivial, requiring suitable generalizations of the basic notions of composition and test. Technically, we consider generalized composition and test operations where *jumps* are allowed—a jump occurring between states related by backward simulation. The proofs justifying the use of jumping composition and test, which can be found in [2], are much more involved than in previous works.

We have implemented our techniques and tested them on BA derived from a set of real-world model checking benchmarks [15] and from the Tabakov-Vardi random model [18]. The new technique is able to finish many of the difficult problem instances in minutes while the algorithm of [1] cannot finish them even in one day. All our benchmarks, the source code, and the executable of our implementation are available at `http://www.languageinclusion.org/CONCUR2011`. Due to limited space, some details of the experiments are deferred to [2].

Related Work. An alternative approach to language inclusion for BA is given by *rank-based* methods [14], which provide a different complementation procedure based on a rank-based analysis of rejecting runs. This approach is orthogonal to Ramsey-based algorithms. In fact, while rank-based approaches have a better worst-case complexity, Ramsey-based approaches can still perform better on many examples [10]. A subsumption-based algorithm for the rank-based approach has been given in [5]. Subsumption techniques have recently been considered also for automata over *finite words* [20,3].

2 Preliminaries

A *Büchi Automaton (BA)* $\mathcal{A}$ is a tuple $(\Sigma, Q, I, F, \delta)$ where Σ is a finite alphabet, Q is a finite set of states, $I \subseteq Q$ is a non-empty set of *initial* states, $F \subseteq Q$ is a set of *accepting* states, and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. A *run* of $\mathcal{A}$ on a word $w = \sigma_1\sigma_2 \ldots \in \Sigma^\omega$ *starting* in a state $q_0 \in Q$ is an infinite sequence $q_0q_1\ldots$ s.t. $(q_{j-1}, \sigma_j, q_j) \in \delta$ for all $j > 0$. The run is *accepting* iff $q_i \in F$ for infinitely many i. The *language of* $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{w \mid \mathcal{A}$ has an accepting run on w starting from some $q_0 \in I\}$.

A *path* in $\mathcal{A}$ on a finite word $w = \sigma_1 \ldots \sigma_n \in \Sigma^+$ is a finite sequence $q_0q_1 \ldots q_n$ s.t. $\forall 0 < j \leq n: \ (q_{j-1}, \sigma_j, q_j) \in \delta$. The path is *accepting* iff $\exists 0 \leq i \leq n: \ q_i \in F$. For any $p, q \in Q$, let $p \overset{w}{\rightsquigarrow}_F q$ iff there is an accepting path on w from p to q, and $p \overset{w}{\rightsquigarrow} q$ iff there is a (not necessarily accepting) path on w from p to q.

A *forward simulation* [4] on $\mathcal{A}$ is a relation $R \subseteq Q \times Q$ such that pRr only if $p \in F \implies r \in F$, and for every transition $(p, \sigma, p') \in \delta$, there exists a transition $(r, \sigma, r') \in \delta$ s.t. $p'Rr'$. A *backward simulation* on $\mathcal{A}$ ([17], where it is called *reverse simulation*) is a relation $R \subseteq Q \times Q$ s.t. $p'Rr'$ only if $p' \in F \implies r' \in F$, $p' \in I \implies r' \in I$, and for every $(p, \sigma, p') \in \delta$, there exists $(r, \sigma, r') \in \delta$ s.t. pRr. Note that this notion of backward simulation is stronger than the usual finite-word automata version, as we require not only compatibility w.r.t. initial states, but also w.r.t. final states. It can be shown that there exists a unique maximal forward simulation denoted by $\preceq_{\mathsf{f}}^{\mathcal{A}}$ and also a unique maximal backward simulation denoted by $\preceq_{\mathsf{b}}^{\mathcal{A}}$, which are both polynomial-time computable preorders [11]. We drop the superscripts when no confusion can arise.

In the rest of the paper, we fix two BA $\mathcal{A} = (\Sigma, Q_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}}, \delta_{\mathcal{A}})$ and $\mathcal{B} = (\Sigma, Q_{\mathcal{B}}, I_{\mathcal{B}}, F_{\mathcal{B}}, \delta_{\mathcal{B}})$. The *language inclusion problem* consists in deciding whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. It is well known that deciding language inclusion is PSPACE-complete [13], and that forward simulations [4] can be used as an underapproximation thereof. Here, we focus on deciding language inclusion precisely, by giving a complete algorithm.

3 Ramsey-Based Language Inclusion Testing

Abstractly, the Ramsey-based approach for checking $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ consists in building a *finite* set $\mathcal{X} \subseteq 2^{\mathcal{L}(\mathcal{A})}$ of fragments of $\mathcal{L}(\mathcal{A})$ satisfying the following two properties:

α (covering) $\bigcup \mathcal{X} = \mathcal{L}(\mathcal{A})$.
β (dichotomy) For all $X \in \mathcal{X}$, either $X \subseteq \mathcal{L}(\mathcal{B})$ or $X \cap \mathcal{L}(\mathcal{B}) = \emptyset$.

The covering property ensures that the considered fragments cover $\mathcal{L}(\mathcal{A})$, and the dichotomy property states that the fragments are either entirely in $\mathcal{L}(\mathcal{B})$ or disjoint from

$\mathcal{L}(\mathcal{B})$. Moreover, the fragments are chosen such that they can be effectively generated and such that their inclusion in $\mathcal{L}(B)$ is easy to test. During the generation of the fragments, it then suffices to test each of them for inclusion in $\mathcal{L}(\mathcal{B})$. If this is the case, the inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ holds. Otherwise, there is a fragment $X \subseteq \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$ s.t. every ω-word $w \in X$ is a counterexample to the inclusion of $\mathcal{L}(\mathcal{A})$ in $\mathcal{L}(\mathcal{B})$.

We now instantiate the above described abstract algorithm by giving primitives for representing fragments of $\mathcal{L}(\mathcal{A})$ satisfying the conditions of covering and dichotomy. Much like in [9], we introduce the notion of *arcs* for satisfying Condition α, the notion of *graphs* for Condition β, and then we put them together in the notion of *supergraphs* as to satisfy $\alpha+\beta$. Then, we explain that supergraphs can be effectively generated and that the fragment languages they represent can be easily tested for inclusion in $\mathcal{L}(B)$.

Condition α*: Edges and Properness.* An *edge* $\langle p,a,q\rangle$ is an element of $E_{\mathcal{A}} = Q_{\mathcal{A}} \times \{0,1\} \times Q_{\mathcal{A}}$. Its language $\mathcal{L}\langle p,a,q\rangle \subseteq \Sigma^+$ contains a word $w \in \Sigma^+$ iff either (1) $a=1$ and $p \overset{w}{\rightsquigarrow}_F q$, or (2) $a=0$, $p \overset{w}{\rightsquigarrow} q$, but not $p \overset{w}{\rightsquigarrow}_F q$. A pair of edges $(\langle q_1,a,q_2\rangle,\langle q_3,b,q_4\rangle)$ is *proper* iff $q_1 \in I_{\mathcal{A}}$, $q_2 = q_3 = q_4$, and $b=1$. A pair of edges (x,y) can be used to encode the ω-language $Y_{xy} = \mathcal{L}(x)\cdot(\mathcal{L}(y))^{\omega}$. Clearly, if the pair of edges is proper, $Y_{xy} \subseteq \mathcal{L}(\mathcal{A})$. Intuitively, the language of a proper pair of edges contains words accepted by lasso-shaped accepting runs starting from q_1 and looping through q_2. Furthermore, it is clearly the case that one can completely cover $\mathcal{L}(\mathcal{A})$ by languages Y_{xy}. Thus, the set $\mathcal{X}_{\mathsf{edges}} = \{Y_{xy} \mid (x,y) \text{ is proper }\}$ satisfies Condition α.

Condition β*: Graphs.* A *graph* g is a subset of edges from $E_{\mathcal{B}} = Q_{\mathcal{B}} \times \{0,1\} \times Q_{\mathcal{B}}$ containing at most one edge for every pair of states. Its language is defined as the set of words over Σ^+ that are consistent with all the edges of the graph. Namely, $w \in \mathcal{L}(g)$ iff, for any pair of states $p,q \in Q_{\mathcal{B}}$, either (1) $p \overset{w}{\rightsquigarrow}_F q$ and $\langle p,1,q\rangle \in g$, (2) $p \overset{w}{\rightsquigarrow} q$, $\neg(p \overset{w}{\rightsquigarrow}_F q)$, and $\langle p,0,q\rangle \in g$, or (3) $\neg(p \overset{w}{\rightsquigarrow} q)$ and there is no edge in g of the form $\langle p,a,q\rangle$. Intuitively, the language of a graph consists of words that all connect any chosen pair of states in the same way (i.e., possibly through an accepting state, through non-accepting states only, or not at all). Let G be the set of all graphs. Not all graphs, however, contain meaningful information, e.g., a graph may contain an edge between states not reachable from each other. Such contradictory information makes the language of a graph empty. Define $G^f = \{g \in G \mid \mathcal{L}(g) \neq \emptyset\}$ as the set of graphs with non-empty languages.

It can be shown that the languages of graphs partition Σ^+. Like with edges, a pair of graphs (g,h) can be used to encode the ω-language $Y_{gh} = \mathcal{L}(g)\cdot(\mathcal{L}(h))^{\omega}$. Intuitively, the pair of graphs g, h encodes *all* runs in $\mathcal{B}$ over the ω-words in Y_{gh}. These runs can be obtained by selecting an edge from g and possibly multiple edges from h that can be connected by their entry/exit states to form a lasso. Since the words in the language of graphs have the same power for connecting states, accepting runs exist for all elements of Y_{gh} or for none of them. The following lemma [16,9,10] shows that the set $\mathcal{X}_{\mathsf{graphs}} = \{Y_{gh} \mid g,h \in G^f\}$ satisfies Condition β.

Lemma 1. *For graphs g,h, either $Y_{gh} \subseteq \mathcal{L}(\mathcal{B})$ or $Y_{gh} \cap \mathcal{L}(\mathcal{B}) = \emptyset$.*

Condition $\alpha+\beta$*: Supergraphs.* We combine edges and graphs to build more complex objects satisfying, at the same time, Conditions α and β. A *supergraph* is a pair

$\mathbf{g} = \langle x, g\rangle \in E_{\mathcal{A}} \times G$.[1] A supergraph is only meaningful if the information in the edge-part is consistent with that in the graph-part. To this end, let $\mathcal{L}(\mathbf{g}) = \mathcal{L}(x) \cap \mathcal{L}(g)$ and let $S^f = \{\mathbf{g} \mid \mathcal{L}(\mathbf{g}) \neq \emptyset\}$ be the set of supergraphs with non-empty language. For two supergraphs $\mathbf{g} = \langle x, g\rangle$ and $\mathbf{h} = \langle y, h\rangle$, the pair $(\mathbf{g}, \mathbf{h})$ is *proper* if the edge-pair (x, y) is proper. Let $Y_{\mathbf{gh}} = \mathcal{L}(\mathbf{g}) \cdot (\mathcal{L}(\mathbf{h}))^{\omega}$. Notice that $Y_{\mathbf{gh}} \subseteq Y_{xy} \cap Y_{gh}$. Therefore, since Y_{gh} satisfies Condition β, so does $Y_{\mathbf{gh}} \subseteq Y_{gh}$. For Condition α, we show that Y_{xy} can be covered by a family of languages of the form $Y_{\langle x,g\rangle\langle y,h\rangle}$. This is sound since $Y_{\langle x,g\rangle\langle y,h\rangle} \subseteq Y_{xy}$ for *any* g, h. Completeness follows from the lemma below, stating that every word $w \in Y_{xy}$ lies in a set of the form $Y_{\langle x,g\rangle\langle y,h\rangle}$. It is proved by a Ramsey-based argument.

Lemma 2. *For proper edges (x, y) and $w \in Y_{xy}$, there exist graphs g, h s.t. $w \in Y_{\langle x,g\rangle\langle y,h\rangle}$.*

Thus, Y_{xy} can be covered by $\mathcal{X}_{xy} = \{Y_{\mathbf{gh}} \mid g, h \in G^f, \mathbf{g} = \langle x, g\rangle, \mathbf{h} = \langle y, h\rangle\}$. Since $\mathcal{X}_{\mathsf{edges}}$ covers $\mathcal{L}(\mathcal{A})$, and each $Y_{xy} \in \mathcal{X}_{\mathsf{edges}}$ can be covered by $\mathcal{X}_{xy}$, it follows that $\mathcal{X} = \{Y_{\mathbf{gh}} \mid \mathbf{g}, \mathbf{h} \in S^f, (\mathbf{g}, \mathbf{h}) \text{ is proper}\}$ covers $\mathcal{L}(\mathcal{A})$. Thus, $\mathcal{X}$ fulfills $\alpha + \beta$.

Generating and Testing Supergraphs. While supergraphs in S^f are a convenient syntactic object for manipulating languages in $\mathcal{X}$, testing that a given supergraph has non-empty language is expensive (PSPACE-complete). In [12], this problem is elegantly solved by introducing a natural notion of *composition* of supergraphs, which preserves non-emptiness: The idea is to start with a (small) set of supergraphs which have non-empty language by construction, and then to obtain S^f by composing supergraphs until no more supergraphs can be generated.

For a BA $\mathcal{C}$ and a symbol $\sigma \in \Sigma$, let $E^{\sigma}_{\mathcal{C}} = \{\langle p, a, q\rangle \mid (p, \sigma, q) \in \delta_{\mathcal{C}}, (a = 1 \iff p \in F \vee q \in F)\}$ be the set of edges induced by σ. The initial seed for the procedure is given by *one-letter supergraphs* in $S^1 = \bigcup_{\sigma\in\Sigma}\{(x, E^{\sigma}_{\mathcal{B}}) \mid x \in E^{\sigma}_{\mathcal{A}}\}$. Notice that $S^1 \subseteq S^f$ by construction. Next, two edges $x = \langle p, a, q\rangle$ and $y = \langle q', b, r\rangle$ are *composable* iff $q = q'$. For composable edges x and y, let $x;y = \langle p, \max(a, b), r\rangle$. Further, the *composition* $g;h$ of graphs g and h is defined as follows: $\langle p, c, r\rangle \in g;h$ iff there is a state q s.t. $\langle p, a, q\rangle \in g$ and $\langle q, b, r\rangle \in h$, and $c = \max_{q\in Q}\{\max(a, b) \mid \langle p, a, q\rangle \in g, \langle q, b, r\rangle \in h\}$. Then, supergraphs $\mathbf{g} = \langle x, g\rangle$ and $\mathbf{h} = \langle y, h\rangle$ are *composable* iff $\langle x, y\rangle$ are composable, and their *composition* is the supergraph $\mathbf{g};\mathbf{h} = \langle x;y, g;h\rangle$. Notice that S^f is closed under composition, i.e., $\mathbf{g}, \mathbf{h} \in S^f \implies \mathbf{g};\mathbf{h} \in S^f$. Composition is also *complete* for generating S^f:

Lemma 3. *[1] A supergraph $\mathbf{g}$ is in S^f iff $\exists \mathbf{g}_1, \ldots, \mathbf{g}_n \in S^1$ such that $\mathbf{g} = \mathbf{g}_1; \ldots; \mathbf{g}_n$.*

Now that we have a method for generating all relevant supergraphs, we need a way of checking inclusion of (supergraphs representing) fragments of $\mathcal{L}(\mathcal{A})$ in $\mathcal{L}(\mathcal{B})$. Let $(\mathbf{g}, \mathbf{h})$ be a (proper) pair of supergraphs. By the dichotomy property, $Y_{\mathbf{gh}} \subseteq \mathcal{L}(\mathcal{B})$ iff $Y_{\mathbf{gh}} \cap \mathcal{L}(\mathcal{B}) \neq \emptyset$. We test the latter condition by the so-called *double graph test*: For a pair of supergraphs $(\mathbf{g}, \mathbf{h})$, $DGT(\mathbf{g}, \mathbf{h})$ iff, whenever $(\mathbf{g}, \mathbf{h})$ is proper, then $LFT(g, h)$. Here, LFT is the so-called *lasso-finding test*: Intuitively, LFT checks for a lasso with a handle in g and an accepting loop in h. Formally, $LFT(g, h)$ iff there is an edge $\langle p, a_0, q_0\rangle \in g$ and an infinite sequence of edges $\langle q_0, a_1, q_1\rangle$, $\langle q_1, a_2, q_2\rangle, \ldots \in h$ s.t. $p \in I$ and $a_j = 1$ for infinitely many j's.

[1] The definition of supergraph given here is slightly different from [9,1], where the edge-part is just a pair of states (p, q). Having labels allows us to give a notion of properness which does not require to have $q \in F$.

Lemma 4. *[1] $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ iff for all $\mathbf{g}, \mathbf{h} \in S^f$, $DGT(\mathbf{g}, \mathbf{h})$.*

Basic Algorithm [9]. The basic algorithm for checking inclusion enumerates all supergraphs from S^f by extending supergraphs on the right by one-letter supergraphs from S^1; that is, a supergraph $\mathbf{g}$ generates new supergraphs by selecting some $\mathbf{h} \in S^1$ and building $\mathbf{g};\mathbf{h}$. Then, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ holds iff all the generated pairs pass the DGT.

Intuitively, the algorithm processes all lasso-shaped runs that can be used to accept some words in $\mathcal{A}$. These runs are represented by the edge-parts of proper pairs of generated supergraphs. For each such run of $\mathcal{A}$, the algorithm uses LFT to test whether there is a corresponding accepting run of $\mathcal{B}$ among *all* the possible runs of $\mathcal{B}$ on the words represented by the given pair of supergraphs. These latter runs are encoded by the graph-parts of the respective supergraphs.

4 Optimized Language Inclusion Testing

The basic algorithm of Section 3 is wasteful for two reasons. First, not all edges in the graph component of a supergraph are needed to witness a counterexample to inclusion: Hence, we can reduce a graph by keeping only a certain subset of its edges (Optimization 1). Second, not all supergraphs need to be generated and tested: We show a method which safely allows the algorithm to discard certain supergraphs (Optimization 2). Both optimizations rely on various notions of *subsumption*, which we introduce next.

Given two edges $x = \langle p, a, q \rangle$ and $y = \langle r, b, s \rangle$, we say that y *subsumes* x, written $x \sqsubseteq y$, if $p = r$, $a \leq b$, and $q = s$; that x *forward-subsumes* y, written $x \sqsubseteq_{\mathsf{f}} y$, if $p = r$, $a \leq b$, and $q \preceq_{\mathsf{f}} s$; that x *backward-subsumes* y, written $x \sqsubseteq_{\mathsf{b}} y$, if $p \preceq_{\mathsf{b}} r$, $a \leq b$, and $q = s$; and that x *forward-backward-subsumes* y, written $x \sqsubseteq_{\mathsf{fb}} y$, if $p \preceq_{\mathsf{b}} r$, $a \leq b$, and $q \preceq_{\mathsf{f}} s$. We lift all the notions of subsumption to graphs: For any $z \in \{\mathsf{f}, \mathsf{b}, \mathsf{fb}, \llcorner\!\lrcorner\}$ and for graphs g and h, let $g \sqsubseteq_z h$ iff, for every edge $x \in g$, there exists an edge $y \in h$ s.t. $x \sqsubseteq_z y$. Since the simulations $\preceq_{\mathsf{f}}$ and $\preceq_{\mathsf{b}}$ are preorders, all subsumptions are preorders. We define backward and forward-backward subsumption equivalence as $\simeq_{\mathsf{b}} = \sqsubseteq_{\mathsf{b}} \cap \sqsubseteq_{\mathsf{b}}^{-1}$ and $\simeq_{\mathsf{fb}} = \sqsubseteq_{\mathsf{fb}} \cap \sqsubseteq_{\mathsf{fb}}^{-1}$, respectively.

4.1 Optimization 1: Minimization of Supergraphs

The first optimization concerns the structure of individual supergraphs. Let $\mathbf{g} = \langle x, g \rangle \in S$ be a supergraph, with g its graph-component. We minimize g by deleting edges therein which are subsumed by $\sqsubseteq_{\mathsf{fb}}$-larger ones. That is, whenever we have $x \sqsubseteq_{\mathsf{fb}} y$ for two edges $x, y \in g$, we remove x and keep y. Intuitively, subsumption-larger arcs contribute more to the capability of representing lassoes since their right and left endpoints are $\preceq_{\mathsf{f}}$/$\preceq_{\mathsf{b}}$-larger, respectively, and have therefore a richer choice of possible futures and pasts. Subsumption smaller arcs are thus redundant, and removing them does not change the capability of g to represent lassoes in $\mathcal{B}$. Formally, we define a minimization operation *Min* mapping a supergraph $\mathbf{g} = \langle x, g \rangle$ to its minimized version $Min(\mathbf{g}) = \langle x, Min(g) \rangle$ where $Min(g)$ is the minimization applied to the graph-component.[2]

[2] In [1], we used $\sqsubseteq_{\mathsf{f}}$ for minimization. The theory allowing the use of $\sqsubseteq_{\mathsf{fb}}$ is significantly more involved, but as as shown in Section 8, the use of $\sqsubseteq_{\mathsf{fb}}$ turns out to be much more advantageous.

Definition 1. *For two graphs g and h, let $g \leqslant h$ iff (1) $g \sqsubseteq h$ and (2) $h \sqsubseteq_{\mathsf{fb}} g$. For supergraphs $\mathbf{g} = \langle x, g\rangle$ and $\mathbf{h} = \langle y, h\rangle$, let $\mathbf{g} \leqslant \mathbf{h}$ iff $x = y$ and $g \leqslant h$. A* minimization *of graphs is any function Min such that, for any graph h, $Min(h) \leqslant h$.*

Point 1 in the definition of $\leqslant$ allows some edges to be erased or their label decreased. Point 2 states that only subsumed arcs can be removed or have their label decreased. Note also that, clearly, $Min(\mathbf{h}) \leqslant \mathbf{h}$ holds for any supergraph $\mathbf{h}$. Finally, note that Min is not uniquely determined: First, there are many candidates satisfying $Min(h) \leqslant h$. Yet, an implementation will usually remove a maximal number of edges to keep the size of graphs to a minimum. Second, even if we required $Min(h)$ to be a $\leqslant$-smallest element (i.e., no further edge can be removed), the minimization process might encounter $\sqsubseteq_{\mathsf{fb}}$-equivalent edges, and in this case, we do not specify which ones get removed. Therefore, we prove correctness for any minimization satisfying $Min(h) \leqslant h$.

Intuitively, a minimized supergraph $\mathbf{g}$ can be seen as a small *representative* of all supergraphs $\mathbf{h} \in G^f$ with $\mathbf{g} \leqslant \mathbf{h}$, and of all the fragments of $\mathcal{L}(A)$ encoded by them. Using representatives allows us to deal with a smaller number of smaller supergraphs. We now explain how (sufficiently many) representatives encoding fragments of $\mathcal{L}(\mathcal{A})$ can be *generated* and *tested* for inclusion in $\mathcal{L}(\mathcal{B})$.

Generating Representatives of Supergraphs. We need to create a representative of each supergraph in S^f by composing representatives only. Let $\mathbf{g} = \langle x, g\rangle$ and $\mathbf{h} = \langle y, h\rangle$ be two composable supergraphs, representing $\mathbf{g}' = \langle x, g'\rangle$ and $\mathbf{h}' = \langle y, h'\rangle$, respectively. If graph composition were $\leqslant$-monotone, i.e., $\mathbf{g};\mathbf{h} \leqslant \mathbf{g}';\mathbf{h}'$, then we would be done. However, graph composition is not monotone: The reason is that some composable edges $e \in g'$ and $f \in h'$ may be erased by minimization, and be represented by some $\hat{e} \in g$ and $\hat{f} \in h$ instead, with $e \sqsubseteq_{\mathsf{fb}} \hat{e}$ and $f \sqsubseteq_{\mathsf{fb}} \hat{f}$. But now, $\hat{e}$ and $\hat{f}$ are not necessarily composable anymore. Thus, $\mathbf{g};\mathbf{h} \not\leqslant \mathbf{g}';\mathbf{h}'$. We solve this problem in two steps: We allow composition to jump to $\preceq_{\mathsf{b}}$-larger states (Def. 2), and relax the notion of representative (Def. 3).

Definition 2. *Given graphs $g, h \in G$, their* jumping composition *$g \,⨟_{\mathsf{b}}\, h$ contains an edge $\langle p, c, r\rangle \in g \,⨟_{\mathsf{b}}\, h$ iff there are edges $\langle p, a, q\rangle \in g$, $\langle q', b, r\rangle \in h$ s.t. $q \preceq_{\mathsf{b}} q'$, and $c = \max_{q,q'}\{\max(a,b) \mid \langle p, a, q\rangle \in g, \langle q', b, r\rangle \in h, q \preceq_{\mathsf{b}} q'\}$. For two composable supergraphs $\mathbf{g} = \langle x, g\rangle$ and $\mathbf{h} = \langle y, h\rangle$, let $\mathbf{g} \,⨟_{\mathsf{b}}\, \mathbf{h} = \langle x;y, g \,⨟_{\mathsf{b}}\, h\rangle$.*

Jumping composition alone does not yet give the required monotonicity property. The problem is that $\mathbf{g} \,⨟_{\mathsf{b}}\, \mathbf{h}$ is not necessarily a minimized version of $\mathbf{g}';\mathbf{h}'$, but it is only a minimized version of something $\simeq_{\mathsf{b}}$-equivalent to $\mathbf{g}';\mathbf{h}'$. This leads us to the following more liberal notion of representatives, which is based on $\leqslant$ modulo the equivalence $\simeq_{\mathsf{b}}$, and for which Lemma 5 proves the required monotonicity property.

Definition 3. *A graph $g \in G$ is a* representative *of a graph $h \in G^f$, denoted $g \trianglelefteq h$, iff there exists $\bar{h} \in G$ such that $g \leqslant \bar{h} \simeq_{\mathsf{b}} h$. For supergraphs $\mathbf{g} = \langle x, g\rangle, \mathbf{h} = \langle y, h\rangle \in S$, we say that $\mathbf{g}$ is a* representative *of $\mathbf{h}$, written $\mathbf{g} \trianglelefteq \mathbf{h}$, iff $x = y$ and $g \trianglelefteq h$. Let $S^R = \{\mathbf{g} \mid \exists \mathbf{h} \in S^f.\ \mathbf{g} \trianglelefteq \mathbf{h}\}$ be the set of representatives of supergraphs.*

Lemma 5. *For supergraphs $\mathbf{g}, \mathbf{h} \in S^R$ and $\mathbf{g}', \mathbf{h}' \in S^f$, if $\mathbf{g} \trianglelefteq \mathbf{g}'$, $\mathbf{h} \trianglelefteq \mathbf{h}'$ and $\mathbf{g}', \mathbf{h}'$ are composable, then $\mathbf{g}, \mathbf{h}$ are composable and $\mathbf{g} \,⨟_{\mathsf{b}}\, \mathbf{h} \trianglelefteq \mathbf{g}';\mathbf{h}'$ and $\mathbf{g} \,⨟_{\mathsf{b}}\, \mathbf{h} \in S^R$.*

Lemma 6. *Let* $\mathbf{f} \in S$, $\mathbf{g} \in S^R$, *and* $\mathbf{h} \in S^f$. *If* $\mathbf{f} \leqslant \mathbf{g}$ *and* $\mathbf{g} \trianglelefteq \mathbf{h}$, *then* $\mathbf{f} \trianglelefteq \mathbf{h}$ *(and thus* $\mathbf{f} \in S^R$*). In particular, the statement holds when* $\mathbf{f} = Min(\mathbf{g})$.

Lemmas 5, 6, and 3 imply that creating supergraphs by $\fatsemi_{\mathsf{b}}$-composing representatives, followed by further minimization, suffices to create a representative of each supergraph in S^f. This solves the problem of generating representatives of supergraphs.

Weak Properness and Relaxed DGT. We now present a relaxed DGT proposed in [1], which we further improve below. The idea is to weaken the properness condition in order to allow more pairs of supergraphs to be eligible for LFT on their graph part. This may lead to a quicker detection of a counterexample. Weak properness is sound since it still produces fragments $Y_{\mathbf{gh}} \subseteq \mathcal{L}(\mathcal{A})$ as required by Condition α. Completeness is guaranteed since properness implies weak properness.

Definition 4. *(adapted from [1]) A pair of edges* $(\langle p,a,q\rangle, \langle r,b,s\rangle)$ *is* weakly proper *iff* $p \in I_{\mathcal{A}}$, $r \preceq_{\mathsf{f}} q$, $r \preceq_{\mathsf{f}} s$, *and* $b = 1$,[3] *and a pair of supergraphs* $(\mathbf{g} = \langle x,g\rangle, \mathbf{h} = \langle y,h\rangle)$ *is weakly proper when* (x,y) *is weakly proper. Supergraphs* $\mathbf{g},\mathbf{h}$ *pass the* relaxed double graph test, *denoted* $RDGT(\mathbf{g},\mathbf{h})$, *iff whenever* $(\mathbf{g},\mathbf{h})$ *is weakly proper, then* $LFT(g,h)$.

Lemma 7. *[1]* $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ *iff for all* $\mathbf{g},\mathbf{h} \in S^f$, $RDGT(\mathbf{g},\mathbf{h})$.

Testing Representatives of Supergraphs. We need a method for testing inclusion in $\mathcal{L}(\mathcal{B})$ of the fragments of $\mathcal{L}(\mathcal{A})$ encoded by representatives of supergraphs that is equivalent to testing inclusion of fragments of $\mathcal{L}(\mathcal{A})$ encoded by the represented supergraphs. As with composition, minimization is not compatible with such testing since edges needed to find loops may be erased during the minimization process. Technically, this results in the LFT (and therefore RDGT) not being $\trianglelefteq$-monotone. Therefore, we generalize the LFT by allowing jumps to $\preceq_{\mathsf{b}}$-larger states, in a similar way as with $\fatsemi_{\mathsf{b}}$. Lemma 8 establishes the required monotonicity property.

Definition 5. *A pair of graphs* (g,h) *passes the* jumping lasso-finding test, *denoted* $LFT_{\mathsf{b}}(g,h)$, *iff there is an edge* $\langle p,a_0,q_0\rangle$ *in* g *and an infinite sequence of edges* $\langle q_0',a_1,q_1\rangle$, $\langle q_1',a_2,q_2\rangle,\dots$ *in* h *s.t.* $p \in I$, $q_i \preceq_{\mathsf{b}} q_i'$ *for all* $i \geq 0$, *and* $a_j = 1$ *for infinitely many* j*'s. A pair of supergraphs* $(\mathbf{g},\mathbf{h})$ *passes the* jumping relaxed double graph test, *denoted* $RDGT_{\mathsf{b}}(\mathbf{g},\mathbf{h})$, *iff whenever* $(\mathbf{g},\mathbf{h})$ *is weakly proper, then* $LFT_{\mathsf{b}}(g,h)$.

Lemma 8. *For any* $\mathbf{g},\mathbf{h} \in S^R$ *and* $\mathbf{g}',\mathbf{h}' \in S^f$ *such that* $\mathbf{g} \trianglelefteq \mathbf{g}'$ *and* $\mathbf{h} \trianglelefteq \mathbf{h}'$, *it holds that* $RDGT_{\mathsf{b}}(\mathbf{g},\mathbf{h}) \iff RDGT(\mathbf{g}',\mathbf{h}')$.

Algorithm with Minimization. By Lemma 8, $\mathrm{RDGT}_{\mathsf{b}}$ on representatives is equivalent to RDGT on the represented supergraphs. Together with Lemma 7, this means that it is enough to generate a representative of each supergraph from S^f, and test all pairs of the generated supergraphs with $\mathrm{RDGT}_{\mathsf{b}}$. Thus, we have obtained a modification of the basic algorithm which starts from minimized 1-letter supergraphs in $Min(S^1) = \{Min(\mathbf{g}) \mid \mathbf{g} \in S^1\}$, and constructs new supergraphs by $\fatsemi_{\mathsf{b}}$-composing already generated supergraphs with $Min(S^1)$ on the right. New supergraphs are further minimized with Min. Inclusion holds iff all pairs of generated supergraphs pass $\mathrm{RDGT}_{\mathsf{b}}$.

[3] We note that instead of testing $r \preceq_{\mathsf{f}} q$, testing inclusion of the languages of the states is sufficient. Furthermore, instead of testing $r \preceq_{\mathsf{f}} s$, one can test for *delayed simulation*, but not for language inclusion. See [2] for details.

4.2 Optimization 2: Discarding Subsumed Supergraphs

The second optimization gives a rule for discarding supergraphs subsumed by some other supergraph. This is safe in the sense that if a subsumed supergraph can yield a counterexample to language inclusion, then also the subsuming one can yield a counterexample. We present an improved version of the subsumption from [1]. The new version uses both $\preceq_{\mathsf{f}}$ and $\preceq_{\mathsf{b}}$ on the $\mathcal{B}$ part of supergraphs instead of $\preceq_{\mathsf{f}}$ only. This allows us to discard significantly more supergraphs than in [1], as illustrated in Section 8.

Definition 6. *We say that a supergraph* $\mathbf{g} = \langle x, g \rangle$ subsumes *a supergraph* $\mathbf{g}' = \langle y, g' \rangle$, *written* $\mathbf{g} \sqsubseteq_{\mathsf{fb}} \mathbf{g}'$, *iff* $y \sqsubseteq_{\mathsf{f}} x$ *and* $g \sqsubseteq_{\mathsf{fb}} g'$.

Intuitively, if $y \sqsubseteq_{\mathsf{f}} x$, then x has more power for representing lassoes in $\mathcal{A}$ than y since, by the properties of forward simulation, it has a richer choice of possible forward continuations in $\mathcal{A}$. On the other hand, $g \sqsubseteq_{\mathsf{fb}} g'$ means that g' has more chance of representing lassoes in $\mathcal{B}$ than g: In fact, g' contains edges that have a richer choice of backward continuations (due to the $\preceq_{\mathsf{b}}$ on the left endpoints of the edges) as well as a richer choice of forward continuations (due to the $\preceq_{\mathsf{f}}$ on the right endpoints). Thus, it is more likely for $\mathbf{g}$ than for $\mathbf{g}'$ to lead to a counterexample to language inclusion. This intuition is confirmed by the lemma below, stating the $\sqsubseteq_{\mathsf{fb}}$-monotonicity of RDGT$_{\mathsf{b}}$.

Lemma 9. *For supergraphs* $\mathbf{g}, \mathbf{h} \in S^R$ *and* $\mathbf{g}', \mathbf{h}' \in S$, *if* $\mathbf{g} \sqsubseteq_{\mathsf{fb}} \mathbf{g}'$ *and* $\mathbf{h} \sqsubseteq_{\mathsf{fb}} \mathbf{h}'$, *then* $RDGT_{\mathsf{b}}(\mathbf{g}, \mathbf{h}) \Rightarrow RDGT_{\mathsf{b}}(\mathbf{g}', \mathbf{h}')$.

Therefore, no counterexample is lost by testing only $\sqsubseteq_{\mathsf{fb}}$-smaller supergraphs. To show that we can completely discard $\sqsubseteq_{\mathsf{fb}}$-larger supergraphs, we need to show that subsumption is compatible with composition, i.e., that descendants of larger supergraphs are (eventually) subsumed by descendants of smaller ones. Ideally, we would achieve this by showing the following more general fact: For two composable representatives $\mathbf{g}', \mathbf{h}' \in S^R$ that are subsumed by supergraphs $\mathbf{g}$ and $\mathbf{h}$, respectively, the composite supergraph $\mathbf{g} \mathbin{;}_{\mathsf{b}} \mathbf{h}$ subsumes $\mathbf{g}' \mathbin{;}_{\mathsf{b}} \mathbf{h}'$. The problem is that subsumption does not preserve composability: Even if $\mathbf{g}', \mathbf{h}'$ are composable, this needs not to hold for $\mathbf{g}, \mathbf{h}$.

We overcome this difficulty by taking into account the specific way supergraphs are generated by the algorithm. Since we only generate new supergraphs by composing old ones on the right with 1-letter minimized supergraphs, we do not need to show that arbitrary composition is $\sqsubseteq_{\mathsf{fb}}$-monotone. Instead, we show that, for representatives $\mathbf{g}, \mathbf{g}' \in S^R$ and a 1-letter minimized supergraph $\mathbf{h}' \in Min(S^1)$, if $\mathbf{g}$ subsumes $\mathbf{g}'$, then there will always be a supergraph $\mathbf{h}$ available which is composable with $\mathbf{g}$ such that $\mathbf{g} \mathbin{;}_{\mathsf{b}} \mathbf{h}$ subsumes $\mathbf{g}' \mathbin{;}_{\mathsf{b}} \mathbf{h}'$. Thus, we can safely discard $\mathbf{g}'$ from the rest of the computation.

Lemma 10. *For any* $\mathbf{g}, \mathbf{g}' \in S^R$ *with* $\mathbf{g} \sqsubseteq_{\mathsf{fb}} \mathbf{g}'$ *and* $\mathbf{h}' \in Min(S^1)$ *such that* $\mathbf{g}'$ *and* $\mathbf{h}'$ *are composable, there exists* $\hat{\mathbf{h}} \in Min(S^1)$ *such that for all* $\mathbf{h} \in S^R$ *with* $\mathbf{h} \sqsubseteq_{\mathsf{fb}} \hat{\mathbf{h}}$, $\mathbf{g}$ *is composable with* $\mathbf{h}$ *and* $\mathbf{g} \mathbin{;}_{\mathsf{b}} \mathbf{h} \sqsubseteq_{\mathsf{fb}} \mathbf{g}' \mathbin{;}_{\mathsf{b}} \mathbf{h}'$.

Algorithm with Minimization and Subsumption. We have obtained a modification of the algorithm with minimization. It starts with a subset $Init \subseteq Min(S^1)$ of $\sqsubseteq_{\mathsf{fb}}$-smallest minimized one-letter supergraphs. New supergraphs are generated by $;_{\mathsf{b}}$-composition on the right with supergraphs in *Init*, followed by minimization with *Min*. Generated supergraphs that are $\sqsubseteq_{\mathsf{fb}}$-larger than other generated supergraphs are discarded. The inclusion holds iff all pairs of generated supergraphs that are not discarded pass RDGT$_{\mathsf{b}}$. (An illustration of a run of the algorithm can be found in [2].)

5 Using Forward Simulation Between $\mathcal{A}$ and $\mathcal{B}$

Previously, we showed that some supergraphs can safely be discarded because some $\sqsubseteq_{\mathsf{fb}}$-smaller ones are retained, which preserves the chance to find a counterexample to language inclusion. Our subsumption relation $\sqsubseteq_{\mathsf{fb}}$ is based on forward/backward simulation on $\mathcal{A}$ and $\mathcal{B}$. In order to use forward simulation *between* $\mathcal{A}$ and $\mathcal{B}$, we describe a different reason to discard supergraphs. Generally, supergraphs can be discarded because they can neither find a counterexample to inclusion (i.e., always pass the RDGT) nor generate any supergraph that can find a counterexample. However, the RDGT is asymmetric w.r.t. the left and right supergraph. Thus, a supergraph that is useless (i.e., not counterexample-finding) in the left role is not necessarily useless in the right role (and vice-versa). The following condition C is sufficient for a supergraph to be *useless on the left*. Moreover, C is efficiently computable and compatible with subsumption. Therefore, its use preserves the soundness and completeness of our algorithm.

Definition 7. *For* $\mathbf{g}=\langle\langle p,a,q\rangle,g\rangle \in S$, $\mathsf{C}(\mathbf{g})$ *iff* $p \notin I_{\mathcal{A}} \vee (\exists\langle r,b,s\rangle \in g.\ r \in I_{\mathcal{B}} \wedge q \preceq_{\mathsf{f}}^{\mathcal{AB}} s)$.

The first part $p \notin I_{\mathcal{A}}$ of the condition is obvious because paths witnessing counterexamples to inclusion must start in an initial state. The second part $(\exists\langle r,b,s\rangle \in g.\ r \in I_{\mathcal{B}}, q \preceq_{\mathsf{f}}^{\mathcal{AB}} s)$ uses forward-simulation $\preceq_{\mathsf{f}}^{\mathcal{AB}}$ *between* $\mathcal{A}$ and $\mathcal{B}$ to witness that neither this supergraph nor any other supergraph generated from it will find a counterexample *when used on the left side of the RDGT*. It might still be needed for tests on the right side of the RDGT though. Instead of $\preceq_{\mathsf{f}}^{\mathcal{AB}}$, every relation implying language inclusion would suffice, but (as mentioned earlier) simulation preorder is efficiently computable while inclusion is PSPACE-complete. The following lemma shows the correctness of C.

Lemma 11. $\forall \mathbf{g},\mathbf{h} \in S^R.\ \mathsf{C}(\mathbf{g}) \Rightarrow RDGT_{\mathsf{b}}(\mathbf{g},\mathbf{h})$.

C is $\sqsubseteq_{\mathsf{fb}}$-upward-closed and closed w.r.t. right extensions. Hence, it is compatible with subsumption-based pruning of the search space and with the employed incremental construction of supergraphs (namely, satisfaction of the condition is inherited to supergraphs newly generated by right extension with one-letter supergraphs).

Lemma 12. *Let* $\mathbf{g},\mathbf{h} \in S$ *s.t.* $\mathbf{g} \sqsubseteq_{\mathsf{fb}} \mathbf{h}$. *Then* $\mathsf{C}(\mathbf{g}) \Rightarrow \mathsf{C}(\mathbf{h})$.

Lemma 13. *Let* $\mathbf{g} \in S^R$, $\mathbf{h} \in Min(S^1)$ *be composable. Then* $\mathsf{C}(\mathbf{g}) \Rightarrow \mathsf{C}(\mathbf{g} \mathbin{;}_{\mathsf{b}} \mathbf{h})$.

In principle, one could store separate sets of supergraphs for use on the left/right in the RDGT, respectively. However, since all supergraphs need to be used on the right anyway, a simple flag is more efficient. We assign the label L to a supergraph to indicate that it is still useful on the left in the RDGT. If a supergraph satisfies condition C, then the L-label is removed. The algorithm counts the number of stored supergraphs that still carry the L-label. If this number drops to zero, then (1) it will remain zero (by Lemma 13), and

(2) no RDGT will ever find a counterexample: In this case, the algorithm can terminate early and report inclusion. In the special case where forward-simulation holds even between the *initial* states of $\mathcal{A}$ and $\mathcal{B}$, condition C is true for *every* generated supergraph. Thus, all L-labels are removed and the algorithm terminates immediately, reporting inclusion. Of course, condition C can also help in other cases where simulation does not hold between initial states but "more deeply" inside the automata.

The following lemma shows that if some supergraph $\mathbf{g}$ can find a counterexample when used on the left in the RDGT, then at least one of its 1-letter right-extensions can also find a counterexample. Intuitively, the counterexample has the form of a prefix followed by an infinite loop, and the prefix can always be extended by one step. E.g., the infinite words $xy(abc)^{\omega}$ and $xya(bca)^{\omega}$ are equivalent. This justifies the optimization in line 15 of our algorithm (cf. [2]).

Lemma 14. *Let* $\mathbf{g}, \mathbf{h} \in S^R$. *If* $\neg RDGT_{\mathsf{b}}(\mathbf{g}, \mathbf{h})$, *then there exists a* $\sqsubseteq_{\mathsf{fb}}$*-minimal supergraph* $\mathbf{f}$ *in* $Min(S^1)$ *and* $\mathbf{e} \in S^R$ *s.t.* $\neg RDGT_{\mathsf{b}}(\mathbf{g} \mathbin{⨟}_{\mathsf{b}} \mathbf{f}, \mathbf{e})$.[4]

6 Metagraphs and a New RDGT

Since many supergraphs share the same graph for $\mathcal{B}$, they can be more efficiently represented by a combined structure that we call a *metagraph*. Moreover, metagraphs allow to define a new RDGT where several $\mathcal{A}$-edges jointly witness a counterexample to inclusion, so that counterexamples can be found earlier than with individual supergraphs.

A metagraph is a structure (X, g) where $X \subseteq E_{\mathcal{A}}$ is a set of $\mathcal{A}$-edges and $g \in G_{\mathcal{B}}$. The metagraph (X, g) represents the set of all supergraphs $\langle x, g \rangle$ with $x \in X$. The L-labels of supergraphs then become labels of the elements of X since the graph g is the same.

We lift basic concepts from supergraphs to metagraphs. For every character $\sigma \in \Sigma$, there is exactly one single-letter metagraph $(E^{\sigma}_{\mathcal{A}}, E^{\sigma}_{\mathcal{B}})$. Let $M^1 = \{(E^{\sigma}_{\mathcal{A}}, E^{\sigma}_{\mathcal{B}}) | \ \sigma \in \Sigma\}$. Thus, the set of single-letter metagraphs M^1 represents all single-letter supergraphs in S^1. The function *RightExtend* defines the composition of two metagraphs such that $RightExtend((X, g), (Y, h)) = (X;Y, g \mathbin{⨟}_{\mathsf{b}} h)$, which is the metagraph containing the supergraphs that are $⨟_{\mathsf{b}}$-right extensions of supergraphs contained in (X, g) by supergraphs contained in (Y, h). The L-labels of the elements $z \in X;Y$ are assigned after testing condition C. The function Min_{f} is defined on sets $X \subseteq E_{\mathcal{A}}$ s.t. $Min_{\mathsf{f}}(X)$ contains the $\sqsubseteq_{\mathsf{f}}$-minimal edges of X. If some edges are $\sqsubseteq_{\mathsf{f}}$-equivalent, then $Min_{\mathsf{f}}(X)$ contains just any of them. Let $Min_M(X, g) = (Min_{\mathsf{f}}(X), Min(g))$. Thus, $Min_M(X, g)$ contains exactly one representative of every $\simeq_{\mathsf{fb}}$ equivalence class of the $\sqsubseteq_{\mathsf{fb}}$-minimal supergraphs in (X, g).

It is not meaningful to define subsumption for metagraphs. Instead, we need to remove certain supergraphs (i.e., $\mathcal{A}$-edges) from some metagraph if another metagraph contains a $\sqsubseteq_{\mathsf{fb}}$-smaller supergraph. If no $\mathcal{A}$-edge remains, i.e., $X = \emptyset$ in (X, g), then this metagraph can be discarded. This is the purpose of introducing the function *Clean*: It takes two metagraphs (X, g) and (Y, h), and it returns a metagraph (Z, g) that describes all supergraphs from (X, g) for which there is no $\sqsubseteq_{\mathsf{fb}}$-smaller supergraph in (Y, h). Formally, if $h \sqsubseteq_{\mathsf{fb}} g$, then $x \in Z$ iff $x \in X$ and $\nexists y \in Y$ s.t. $x \sqsubseteq_{\mathsf{f}} y$. Otherwise, if $h \not\sqsubseteq_{\mathsf{fb}} g$, then $Z = X$. Now we define a generalized RDGT on metagraphs.

[4] A slightly modified version, presented in [2], holds for the version of the RDGT mentioned in the footnote on Definition 4.

Algorithm 1. *Inclusion Checking with Metagraphs*

```
**Input**: BA $\mathcal{A} = (\Sigma, Q_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A}, \delta_\mathcal{A})$, $\mathcal{B} = (\Sigma, Q_\mathcal{B}, I_\mathcal{B}, F_\mathcal{B}, \delta_\mathcal{B})$, and the set $M^1_{\mathcal{A},\mathcal{B}}$.
**Output**: TRUE if $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. Otherwise, FALSE.
$Next := \{Min_M((X,g)) \mid (X,g) \in M^1_{\mathcal{A},\mathcal{B}}\}$; $Init := \emptyset$;
while $Next \neq \emptyset$ do
    Pick and remove a metagraph $(X,g)$ from $Next$;
    $Clean_1((X,g), Init)$;
    if $X \neq \emptyset$ then
        $Clean_3(Init, (X,g))$; Add $(X,g)$ to $Init$;
$Processed := \emptyset$; $Next := Init$;
foreach $(X,g) \in Next$ do
    foreach $x \in X$ do
        if $\neg \mathsf{C}(\langle x,g \rangle)$ then label $x$ with $L$
while $Next \neq \emptyset \wedge \exists (X,g) \in Next \cup Processed.\ \exists x \in X.L(x)$ do
    Pick a metagraph $(X,g)$ from $Next$ and remove $(X,g)$ from $Next$;
    if $\neg RDGT^M_\mathsf{b}((X,g),(X,g))$ then return FALSE;
    if $\exists (Y,h) \in Processed : \neg RDGT^M_\mathsf{b}((Y,h),(X,g)) \vee \neg RDGT^M_\mathsf{b}((X,g),(Y,h))$ then
    return FALSE;
    Create $(X',g)$ from $(X,g)$ by removing the $L$-labels from $X$ and add $(X',g)$ to
    $Processed$;
    foreach $(Y,h) \in Init$ do
        $(Z,f) := Min_M(RightExtend((X,g),(Y,h)))$;
        if $Z \neq \emptyset$ then $Clean_1((Z,f), Next)$;
        if $Z \neq \emptyset$ then $Clean_2((Z,f), Processed)$;
        if $Z \neq \emptyset$ then
            $Clean_3(Next,(Z,f))$; $Clean_3(Processed,(Z,f))$; Add $(Z,f)$ to $Next$;
return TRUE;
```

Definition 8. *A pair of sets of $\mathcal{A}$-edges $X,Y \subseteq E_\mathcal{A}$ passes the* forward-downward jumping lasso-finding test, *denoted $LFT_\mathsf{f}(X,Y)$, iff there is an arc $\langle p, a_0, q_0 \rangle$ in X (with the L-label) and an infinite sequence of arcs $\langle q'_0, a_1, q_1 \rangle, \langle q'_1, a_2, q_2 \rangle, \ldots$ in Y s.t. $p \in I_\mathcal{A}$, $q'_i \preceq_\mathsf{f} q_i$ for all $i \geq 0$, and $a_j = 1$ for infinitely many j's.*

Definition 9. *$RDGT^M_\mathsf{b}((X,g),(Y,h))$ iff, whenever $LFT_\mathsf{f}(X,Y)$, then $LFT_\mathsf{b}(g,h)$.*

The following lemma shows the soundness of the new RDGT.

Lemma 15. *Let $(X,g),(Y,h)$ be metagraphs where all contained supergraphs are in S^R. If $\neg RDGT^M_\mathsf{b}((X,g),(Y,h))$, then $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$.*

If there are $x \in X, y \in Y$ s.t. $\neg RDGT_\mathsf{b}(\langle x,g \rangle, \langle y,h \rangle)$, then $\neg RDGT^M_\mathsf{b}((X,g),(Y,h))$, by Definitions 4, 8, and 9. Thus the completeness of the new RDGT follows already from Lemmas 7 and 8. Checking $RDGT^M_\mathsf{b}((X,g),(Y,h))$ can be done very efficiently for large numbers of metagraphs, by using an abstraction technique that extracts test-relevant information from the metagraphs and stores it separately (cf. [2]).

7 The Main Algorithm

Algorithm 1 describes our inclusion testing algorithm. The function *Clean* is extended to sets of metagraphs in the standard way and implemented in procedures $Clean_1$ and $Clean_3$ in which the result overwrites the first argument (the two procedures differ in the role of the first argument, and $Clean_3$ in addition discards empty metagraphs). Lines 1-6 compute the metagraphs which contain the subsumption-minimal 1-letter supergraphs. Lines 7-10 initialize the set *Next* with these metagraphs and assign the correct labels by testing condition C. $L(x)$ denotes that the $\mathcal{A}$-arc x is labeled with L. Lines 11-21 describe the main loop. It runs until *Next* is empty or there are no more L-labels left. In the main loop, metagraphs are tested (lines 13-14) and then moved from *Next* to *Processed* without the L-label (line 15). Moreover, new metagraphs are created and some parts of them discarded by the *Clean* operation (lines 16-21). Extra bookkeeping is needed to handle the case where L-labels are regained by supergraphs in *Processed* in line 19 (see $Clean_2$ in [2]).

Theorem 1. *Algorithm 1 terminates. It returns* TRUE *iff* $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$.

8 Experimental Results

We have implemented the proposed inclusion-checking algorithm in Java (the implementation is available at `http://www.languageinclusion.org/CONCUR2011`) and tested it on automata derived from (1) mutual exclusion protocols [15] and (2) the Tabakov-Vardi model [18]. We have compared the performance of the new algorithm with the one in [1] (which only uses supergraphs, not metagraphs, and subsumption and minimization based on forward simulation on $\mathcal{A}$ and on $\mathcal{B}$), and found it better on average, and, in particular, on difficult instances where the inclusion holds. Below, we present a condensed version of the results. Full details can be found in [2].

In the first experiment, we inject artificial errors into models of several mutual exclusion protocols from [15][5], translate the modified versions into BA, and compare the sequences of program states (w.r.t. occupation of the critical section) of the two versions. For each protocol, we test language inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ of two variants $\mathcal{A}$ and $\mathcal{B}$. We use a timeout of 24 hours and a memory limit of 4GB. We record the running time and indicate a timeout by ">24h". We compare the algorithm from [1] against its various improvements proposed above. The basic new setting (denoted as "default" in the results) uses forward simulation as in [1] together with metagraphs from Section 6 (and some further small optimizations described in [2]). Then, we gradually add the use of backward simulation proposed in Section 4 (denoted by `-b` in the results) and forward simulation between $\mathcal{A}$ and $\mathcal{B}$ from Section 5 (denoted by `-c`, finally yielding the algorithm of Section 7). We also consider repeated quotienting w.r.t. forward/backward-simulation-equivalence before starting the actual inclusion checking (denoted by `-qr`), while the default does quotienting w.r.t. forward simulation only. In order to better show

[5] The models in [15] are based on guarded commands. We derive variants from them by randomly weakening or strengthening the guard of some commands.

Table 1. Language inclusion checking on mutual exclusion protocols. Forward simulation holds between initial states. The option `-c` is extremely effective in such cases.

Protocol	$\mathcal{A}$		$\mathcal{B}$		Algorithm	New Algorithm			
	Trans.	States	Trans.	States	of [1]	default	`-b`	`-b -qr`	`-b -qr -c`
Peterson	33	20	34	20	0.46s	0.39s	0.54	0.61s	0.03s
Phils	49	23	482	161	12h36m	11h3m	7h21m	7h23m	0.1s
Mcs	3222	1408	21503	7963	>24h	2m43s	2m32s	2m49s	1m24s
Bakery	2703	1510	2702	1509	>24h	>24h	>24h	>24h	12s
Fischer	1395	634	3850	1532	4h50m	2m38s	2m50s	27s	3.6s
FischerV2	147	56	147	56	13m15s	5m14s	1m26s	1m1s	0.1s

Table 2. Language inclusion checking on mutual exclusion protocols. Language inclusion holds, but forward simulation does not hold between initial states (we call this category "inclusion"). The new alg. is much better in FischerV3, due to metagraphs. Option `-b` is effective in FischerV4. BakeryV2 is a case where `-c` is useful even if simulation does not hold between initial states.

Protocol	$\mathcal{A}$		$\mathcal{B}$		Algorithm	New Algorithm			
	Trans.	States	Trans.	States	of [1]	default	`-b`	`-b -qr`	`-b -qr -c`
FischerV3	1400	637	1401	638	3h6m	45s	10s	11s	7s
FischerV4	147	56	1506	526	>24h	>24h	1h31m	2h12m	2h12m
BakeryV2	2090	1149	2091	1150	>24h	>24h	>24h	>24h	18s

Table 3. Language inclusion checking on mutual exclusion protocols. Language inclusion does not hold. Note that the new algorithm uses a different search strategy (BFS) than the alg. in [1].

Protocol	$\mathcal{A}$		$\mathcal{B}$		Algorithm	New Algorithm			
	Trans.	States	Trans.	States	of [1]	default	`-b`	`-b -qr`	`-b -qr -c`
BakeryV3	2090	1149	2697	1506	12m19s	5s	6s	16s	15s
FischerV5	3850	1532	1420	643	7h28m	1m6s	1m47s	39s	36s
PhilsV2	482	161	212	80	1.1s	0.7s	0.8s	1s	1s
PhilsV3	464	161	212	80	1s	0.7s	0.8s	1.2s	1.1s
PhilsV4	482	161	464	161	10.7s	3.8s	4.5s	4.8s	4.8s

the capability of the new techniques, the results are categorized into three classes, according to whether (1) simulation holds, (2) inclusion holds (but not simulation), and (3) inclusion does not hold. See, resp., Tables 1, 2, and 3. On average, the newly proposed approach using all the mentioned options produces the best result.

In the second experiment, we use the Tabakov-Vardi random model[6] with fixed alphabet size 2. There are two parameters, *transition density* (td; average number of tran-

[6] Note that automata generated by the Tabakov-Vardi model are very different from a control-flow graph of a program. They are almost unstructured, and thus on average the density of simulation is much lower. Hence, we believe it is not a fair evaluation benchmark for algorithms aimed at program verification. However, since it was used in the evaluation of most previous works on language inclusion testing, we also include it as one of the evaluation benchmarks.

Table 4. Results of the Tabakov-Vardi experiments on two selected configurations. In each case, we generated 100 random automata and set the timeout to one hour. The new algorithm found more cases with simulation between initial states because the option `-qr` (do fw/bw quotienting repeatedly) may change the forward simulation in each iteration. In the "Hard" case, most of the timeout instances probably belong to the category "inclusion" (Inc).

	Hard: td=2, ad=0.1, size=30		Easy, but nontrivial: td=3, ad=0.6, size=50	
	The Algorithm of [1]	New Algorithm	The Algorithm of [1]	New Algorithm
Sim	1%, 32m42s	2%, 0.025s	13%, 2m5s	21%, 0.14s
Inc	16%, 43m	20%, 30m42s	68%, 26m14s	64%, 6m12s
nInc	49%, 0.17s	49%, 0.2s	15%, 0.3s	15%, 0.3s
TO	34%	29%	4%	0%

sitions per state and alphabet symbol) and *acceptance density* (ad; percentage of accepting states). The results of a complete test for many parameter combinations and automata of size 15 can be found in [2]. Its results can be summarized as follows. In those cases where simulation holds between initial states, the time needed is negligible. Also the time needed to find counterexamples is very small. Only the "inclusion" cases are interesting. Based on the results presented in [2], we picked two configurations (Hard: td=2, ad=0.1, size=30) and (Easy, but nontrivial: td=3, ad=0.6, size=50) for an experiment with larger automata. Both configurations have a substantial percentage of the interesting "inclusion" cases. The results can be found in Table 4.

9 Conclusions

We have presented an efficient method for checking language inclusion for Büchi automata. It augments the basic Ramsey-based algorithm with several new techniques such as the use of weak subsumption relations based on combinations of forward and backward simulation, the use of simulation relations between automata in order to limit the search space, and methods for eliminating redundant tests in the search procedure. We have performed a wide set of experiments to evaluate our approach, showing its practical usefulness. An interesting direction for future work is to characterize the roles of the different optimizations in different application domains. Although their overall effect is to achieve a much better performance compared to existing methods, the contribution of each optimization will obviously vary from one application to another. Such a characterization would allow a portfolio approach in which one can predict which optimization would be the dominant factor on a given problem. In the future, we also plan to implement both the latest rank-based and Ramsey-based approaches in a uniform way and thoroughly investigate how they behave on different classes of automata.

References

1. Abdulla, P.A., Chen, Y.-F., Clemente, L., Holík, L., Hong, C.-D., Mayr, R., Vojnar, T.: Simulation subsumption in ramsey-based büchi automata universality and inclusion testing. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 132–147. Springer, Heidelberg (2010)

2. Abdulla, P.A., Chen, Y.-F., Clemente, L., Holík, L., Hong, C.D., Mayr, R., Vojnar, T.: Advanced Ramsey-based Büchi Automata Inclusion Testing. Technical report FIT-TR-2011-03, FIT BUT, Czech Republic (2011)
3. Abdulla, P.A., Chen, Y.-F., Holík, L., Mayr, R., Vojnar, T.: When Simulation Meets Antichains: On Checking Language Inclusion of Nondeterministic Finite (Tree) Automata. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 158–174. Springer, Heidelberg (2010)
4. Dill, D., Hu, A., Wong-Toi, H.: Checking for language inclusion using simulation preorders. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 255–265. Springer, Heidelberg (1992)
5. Doyen, L., Raskin, J.-F.: Improved Algorithms for the Automata-Based Approach to Model-Checking. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 451–465. Springer, Heidelberg (2007)
6. Etessami, K.: A Hierarchy of Polynomial-Time Computable Simulations for Automata. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 131–131. Springer, Heidelberg (2002)
7. Etessami, K., Wilke, T., Schuller, R.A.: Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. SIAM J. Comp. 34(5) (2005)
8. Fogarty, S.: Büchi Containment and Size-Change Termination. Master's Thesis (2008)
9. Fogarty, S., Vardi, M.Y.: Büchi Complementation and Size-Change Termination. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 16–30. Springer, Heidelberg (2009)
10. Fogarty, S., Vardi, M.Y.: Efficient Büchi Universality Checking. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 205–220. Springer, Heidelberg (2010)
11. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing Simulations on Finite and Infinite Graphs. In: Proc. FOCS 1995. IEEE CS, Los Alamitos (1995)
12. Jones, N.D., Lee, C.S., Ben-Amram, A.M.: The Size-Change Principle for Program Termination. In: Proc. of POPL 2001. ACM SIGPLAN, New York (2001)
13. Kupferman, O., Vardi, M.Y.: Verification of fair transition systems. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 372–384. Springer, Heidelberg (1996)
14. Kupferman, O., Vardi, M.Y.: Weak Alternating Automata Are Not That Weak. ACM Transactions on Computational Logic 2(2), 408–429 (2001)
15. Pelánek, R.: BEEM: Benchmarks for Explicit Model Checkers. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 263–267. Springer, Heidelberg (2007)
16. Sistla, A.P., Vardi, M.Y., Wolper, P.: The Complementation Problem for Büchi Automata with Applications to Temporal Logic. In: Brauer, W. (ed.) ICALP 1985. LNCS, vol. 194, pp. 465–474. Springer, Heidelberg (1985)
17. Somenzi, F., Bloem, R.: Efficient Büchi Automata from LTL Formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS. vol. 1855, pp. 248–263. Springer, Heidelberg (2000)
18. Tabakov, D., Vardi, M.Y.: Model Checking Büchi Specifications. In: Proc. of LATA 2007 (2007)
19. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proc. of LICS 1986. IEEE Comp. Soc. Press, Los Alamitos (1986)
20. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Antichains: A New Algorithm for Checking Universality of Finite Automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 17–30. Springer, Heidelberg (2006)