Existence of Reduction Hierarchies

Orna Kupferman
UC Berkeley*

Robert P. Kurshan
Bell Laboratories†

Mihalis Yannakakis
Bell Laboratories‡

July 30, 1997

Abstract

In the automata-theoretic approach to verification, we model programs and specifications by automata on infinite words. Correctness of a program with respect to a specification can then be reduced to the *language-containment* problem. In a concurrent setting, the program is typically a parallel composition of many coordinating processes, and the language-containment problem that corresponds to verification is

$$(\dagger) \quad \mathcal{L}(P_1) \cap \mathcal{L}(P_2) \cap \cdots \cap \mathcal{L}(P_n) \subseteq \mathcal{L}(T),$$

where $P_1, P_2, \ldots, P_n$ are automata that model the underlying coordinating processes, and T is the task they should perform. In 1994, Kurshan suggested the heuristic of *Reduction Hierarchies* for circumventing the exponential blow-up introduced by conventional methods that solve the problem (†). In the reduction-hierarchy heuristic, we solve the problem (†) by solving a sequence of easier problems, which involve only automata of tractable sizes. Complexity-theoretic conjectures (NP $\neq$ PSPACE) imply that there are settings in which the heuristic cannot circumvent the exponential blow-up. In this paper, we demonstrate the strength of the heuristic, study its properties, characterize settings in which it performs effectively, and suggest a method for searching for reduction hierarchies. In particular, we prove, independently of the NP $\neq$ PSPACE question, that reduction hierarchies of tractable sizes do not always exist.

*Address: EECS Department, Berkeley, CA 94720-1770, U.S.A. Email: orna@eecs.berkeley.edu
†Address: 700 Mountain Avenue, Murray Hill, NJ 07974, U.S.A. Email: k@research.bell-labs.com
‡Address: 700 Mountain Avenue, Murray Hill, NJ 07974, U.S.A. Email: mihalis@research.bell-labs.com

1 Introduction

In *program verification*, we check whether an *implementation* is correct with respect to a *specification*. In the *automata-theoretic approach* to verification, we model the specification and the implementation by automata on infinite words, and correctness amounts to *language containment*. We describe this approach in more detail. Consider a nonterminating finite-state program P over a finite set AP of atomic propositions. Each state of P can be associated with a set of atomic propositions that hold in the state. Then, each computation of P induces an infinite word in $(2^{AP})^\omega$, and P itself induces a language $\mathcal{L}(P)$ over the alphabet 2^{AP}. The language $\mathcal{L}(P)$ consists of all the computations of P, and it can be recognized by a finite-state automaton. A specification or a task T for P can also be viewed as a language $\mathcal{L}(T)$ over the alphabet 2^{AP}. The language $\mathcal{L}(T)$ consists of all the behaviors allowed to P. For almost all interesting tasks T, we can recognize $\mathcal{L}(T)$ by an automaton. In particular, we can translate linear-time temporal-logic formulas to automata [VW94]. Then, P is correct with respect to T iff the behavior of each of the computations of P is allowed, thus $\mathcal{L}(P) \subseteq \mathcal{L}(T)$. The automata-theoretic approach to verification has proven to be useful for verification of control-intensive hardware and software, and it is implemented in the verification tool COSPAN.

The advent of concurrent programming has made program verification significantly more necessary and difficult. In a concurrent setting, the program P is typically the parallel composition of many coordinating processes. Then, the language of P is the intersection of the languages of its underlying processes. Accordingly, the language-containment problem that corresponds, in this setting, to verification is

$$(\dagger) \quad \mathcal{L}(P_1) \cap \mathcal{L}(P_2) \cap \cdots \cap \mathcal{L}(P_n) \subseteq \mathcal{L}(T),$$

where $P_1, P_2, \ldots, P_n$ are the automata corresponding to the underlying coordinating processes, and T is the task they should perform. Conventional methods for solving this problem construct the product $P = P_1 \times P_2 \times \cdots \times P_n$ of the n automata and check whether $\mathcal{L}(P) \subseteq \mathcal{L}(T)$. Such methods are doomed to fail in the worst case, as the size of P is very big (exponential in n). Indeed, the language-containment problem (†) is PSPACE-complete already for a fixed-size T [Koz77]. Coping with this *state-explosion problem* is one of the most important issues in computer-aided verification and is the subject of much active research (cf. [CG87]). In [Kur94a], Kurshan suggested the heuristic of *Reduction Hierarchies* for coping with this problem.

Consider the language-containment problem (†). Let P_i and P_j be two automata in the intersection, and let P_{ij} be such that $\mathcal{L}(P_{ij}) = \mathcal{L}(P_i) \cap \mathcal{L}(P_j)$. Clearly, (†) holds iff the intersection of all automata, with P_{ij} replacing P_i and P_j, is contained in T. We can continue in this fashion and repeatedly replace two automata from the left by an automaton accepting their intersection. Eventually, we will end up with a single automaton P on the left for which (†) holds iff $\mathcal{L}(P) \subseteq \mathcal{L}(T)$. The above suggests a "replacement scheme" for solving the problem (†). Such a scheme, however, is not of much interest, as the automaton P we end-up with is simply the (too big) product of all the underlying automata. When we build a reduction hierarchy for the problem (†), we follow the above replacement scheme, but we perform each replacement

more wisely: we define the automaton P_{ij} to be such that $\mathcal{L}(P_{ij}) \supseteq \mathcal{L}(P_i) \cap \mathcal{L}(P_j)$. That is, an automaton replacing two other automata does not necessarily accept their intersection. Rather, it can accept a larger language, which strictly contains their intersection. It is easy to see that if we end up with an automaton P on the left for which $\mathcal{L}(P) \subseteq \mathcal{L}(T)$ hold, then (†) holds as well. Indeed, the language of the automaton P contains the intersection of the languages of the underlying processes. Intuitively, we only make it harder for the language of P to be contained in T. Why then do we relax the languages of the intermediate automata and allow them to accept words that are not necessarily in the intersection? Hopefully, this flexibility will enable us to keep the size of all automata small (polynomial, rather than exponential, in n). This idea, of relaxing the languages of the intermediate automata while keeping their size small, is the heart of the heuristic of reduction hierarchies.

In a related work [And95], Andersen suggests the method of *partial model checking*, which can be viewed as a restricted version of reduction hierarchies. There, one checks that a concurrent system satisfies a μ-calculus formula by gradually removing components of the system while transforming the formula accordingly, trying to keep it small. Thus, unlike reduction hierarchies, in which we can pair any two components of the system, in partial model checking one applies a list of "pairing rules" by which a component is paired with the specification. This does not support exploitation of abstraction of several coordinating components taken together, which is the main advantage of reduction hierarchies.

We demonstrate the strength of reduction hierarchies with the following simple example (more examples are given in the paper). Consider a parallel composition of n processes. The processes are defined over the set $AP = \{req_1, req_2, \ldots, req_n\}$ of atomic propositions, with req_i standing for "process P_i is sending a request". Each process may send at most one request. That is, the language of the automaton P_i consists of all words over 2^{AP} in which at most one letter contains req_i. It is easy to see that the size of each automaton P_i is fixed. The specification for the composition is that eventually no requests are sent. Thus, $\mathcal{L}(T)$ consists of all words over 2^{AP} with suffix $\emptyset^\omega$. Clearly, we can translate the specification into a fixed-size automaton. It is easy to see that the processes satisfy the specification, thus (†) holds. Nevertheless, though the size of all participating automata is fixed, the size of the intersection of the n automata, which is required for checking (†) naively, is exponential in n.

We can build a reduction hierarchy for (†) keeping the size of all intermediate automata polynomial in n. We can start, say, by replacing, the automata P_1 and P_2 by an automaton P_{12} that accepts all words in 2^{AP} in which at most two letters contain req_1 or req_2. Clearly, $\mathcal{L}(P_{12}) \supseteq \mathcal{L}(P_1) \cap \mathcal{L}(P_2)$. Continuing in this manner, we can replace m automata by an automaton that accepts all words in which at most m letters contain requests of the corresponding m processes. Each of the intermediate automata is of size $O(n)$, and the language of the automaton we end-up which is still contained in $\mathcal{L}(T)$. In fact, we can do better, and build for (†) a reduction hierarchy whose all intermediate automata are of a fixed size. For that, we proceed exactly as above, only that now we replace m automata by an automaton that accepts all words in which only finitely many letters contain requests of the corresponding m processes. That way, we end-up with an automaton P that accepts all words in which requests appear only finitely

often. The automaton P accepts many words that are not in the original intersection – this is what enables us to define it as a fixed-size automaton. Still, its language is contained in $\mathcal{L}(T)$, thus (†) holds.

So, reduction hierarchies suggest a very appealing solution for the state-explosion problem. Do reduction hierarchies of polynomial size always exist? When are they likely to exist? How can we search for, and build, a reduction hierarchy? This is the subject of this paper.

There is evidence from complexity theory that the answer to the first question is negative: namely, if NP $\neq$ PSPACE, then reduction hierarchies of polynomial size do not always exist. Although this complexity-theoretic conjecture is widely believed, it is valuable in such cases, whenever possible, to have an explicit *proof*, for several reasons. First, a proof of the above conjecture (as with other notable ones such as P $\neq$ NP), is nowhere near in sight; for example, despite long, extensive research in circuit complexity, there is a huge gap between what we suspect to be true regarding the size of the Boolean circuits and formulas that are needed to express NP-complete problems (namely exponential size) and what can be currently proved (essentially, only linear size for circuits, quadratic for formulas) [BS90]. Second, explicit constructions will give us a better insight and understanding as to which instances are difficult, in the sense that they do not admit an efficient reduction hierarchy, and hopefully provide some foundation to guide the design of heuristics that search for tractable hierarchies in cases where they might exist.

We present here a proof, independent of complexity-theoretic conjunctures, which refutes the hope to find reduction hierarchies for all problems. We present a family $(\dagger_i)$ of language-containment problems, each with i automata, such that for every i, a reduction hierarchy for solving $(\dagger_i)$ must contain an intermediate automaton of size exponential in i. For that, we study several properties of reduction hierarchies, which are of independent interest and which help us answering the two other questions.

2 Preliminaries

2.1 Streett Automata over Infinite Words

Given an alphabet Σ, an *infinite word over* Σ is an infinite sequence $w = c_0 \cdot c_1 \cdot c_2 \cdots$ of letters in Σ. A *Streett automaton over infinite words* is $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq 2^Q \times 2^Q$ is an acceptance condition. Intuitively, $\delta(q, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Since $\mathcal{A}$ may have several initial states and since the transition function may specify many possible transitions for each state and letter, $\mathcal{A}$ may be *nondeterministic*. If $|Q_0| = 1$ and δ is such that for every $q \in Q$ and $c \in \Sigma$, we have that $|\delta(q, c)| = 1$, then $\mathcal{A}$ is a *deterministic* automaton.

Given an input infinite word $w = c_0 \cdot c_1 \cdot c_2 \cdots \in \Sigma^\omega$, a *run* of $\mathcal{A}$ on w is a function $r : \mathbb{N} \to Q$ where $r(0) \in Q_0$ (i.e., the run starts in one of the initial states) and for every $i \geq 0$, we have

$r(i+1) \in \delta(r(i), c_i)$ (i.e., the run obeys the transition function). For each run r, let $Inf(r)$ be the set of states that r visits *infinitely often*, i.e.,

$$Inf(r) = \{q \in Q \ : \ \text{for infinitely many } i \in \mathbb{N}, \text{ we have } r(i) = q\}.$$

As Q is finite, it is guaranteed that $Inf(r) \neq \emptyset$. The run r is an *accepting* run of $\mathcal{A}$ iff for every pair $\langle G, B\rangle \in F$, we have that $Inf(r) \cap G = \emptyset$ or $Inf(r) \cap B \neq \emptyset$. Otherwise, the run r is a *rejecting*. Note that a nondeterministic automaton can have many runs on w. In contrast, a deterministic automaton has a single run on w. An automaton $\mathcal{A}$ accepts an input word w iff there exists an accepting run r of $\mathcal{A}$ on w. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of infinite words that $\mathcal{A}$ accepts. Thus, each word automaton defines a subset of Σ^ω. We denote by $\overline{\mathcal{L}(\mathcal{A})}$ the complement language of $\mathcal{A}$, that is the set of all words in $\Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.

Computations of a finite-state process can be viewed as infinite words over the alphabet 2^{AP}, where AP is the set of atomic propositions of the process. According to this view, each process corresponds to a language over the alphabet 2^{AP} and can be associated with an automaton. In the verification tool COSPAN, processes are associated with a particular subclass of deterministic Streett automata [Kur94b]. The alphabet of the automata in COSPAN consists of atoms of a Boolean algebra. Accordingly, we define the *size* of an automaton as the size of its state-space, and the size of the formulas in the Boolean algebra that are required for encoding its transitions, ignoring the size of its alphabet. Note that the size of an automaton is also independent of the number of pairs in its acceptance condition. Indeed, in practice, the automata used in COSPAN have very small acceptance conditions, and we assume that the number of pairs in each of these conditions is bounded by the size of the automaton. The results in the paper hold also for a definition of size that does count the number of pairs in the acceptance conditions.

2.2 Reduction Hierarchies

Consider a finite directed acyclic graph $G = \langle V, E\rangle$. For a node $v \in V$, let $id(v)$ denote the *in-degree* of v; that is, the number of nodes u for which $E(u, v)$. Similarly, let $out(v)$ denote the *out-degree* of v; that is, the number of nodes u for which $E(v, u)$. For a set $L \subset V$ and a node $t \in V$, we say that G is a *hierarchy for L rooted at t* if $in(t) = 0$ and $in(v) \neq 0$ for all $v \neq t$, for all $v \in L$ we have $out(v) = 0$, and for all $v \notin L$ we have $out(v) = 2$. If, in addition, $in(v) = 1$ for all $v \neq t$, we say that G is a *tree hierarchy*. We call t the *root* of the hierarchy and L, its *leaves*.

Consider the language-containment problem

$$(\dagger) \quad \mathcal{L}(P_1) \cap \mathcal{L}(P_2) \cap \cdots \cap \mathcal{L}(P_n) \subseteq \mathcal{L}(T),$$

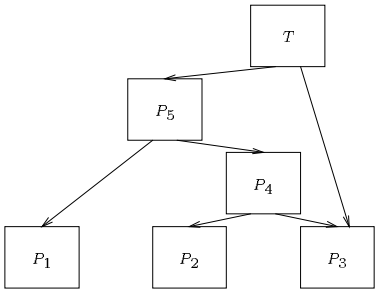
where the size of T and each of the P_i's is a constant function of n. Let β be in {*constant, linear, polynomial, exponential* }. A graph $G = \langle V, E\rangle$ is a *reduction hierarchy, of bound β*, for the problem $(\dagger)$ if the nodes of G are deterministic Streett automata and the following hold:

(R1) G is a hierarchy for L and t, with $L = \{P_1, P_2, \ldots, P_n\}$ and $t = T$;

(R2) For every $v \in V$, if $E(v, v_1)$ and $E(v, v_2)$, then $\mathcal{L}(v) \supseteq \mathcal{L}(v_1) \cap \mathcal{L}(v_2)$;

(R3) The size of all the nodes in G is β in n.

Example 2.1 The graph G below is a hierarchy for $\{P_1, P_2, P_3\}$ rooted at T. If $\mathcal{L}(P_4) \supseteq \mathcal{L}(P_2) \cap \mathcal{L}(P_3)$, $\mathcal{L}(P_5) \supseteq \mathcal{L}(P_1) \cap \mathcal{L}(P_4)$, and $\mathcal{L}(T) \supseteq \mathcal{L}(P_5) \cap \mathcal{L}(P_3)$, then G is also a reduction hierarchy for the problem $\mathcal{L}(P_1) \cap \mathcal{L}(P_2) \cap \mathcal{L}(P_3) \subseteq \mathcal{L}(T)$. Since $in(P_3) = 2$, the hierarchy G is not a tree hierarchy.



A common way to check (†) is to construct an automaton P such that $\mathcal{L}(P) = \mathcal{L}(P_1) \cap \mathcal{L}(P_2) \cap \cdots \cap \mathcal{L}(P_n) \cap \overline{\mathcal{L}(T)}$ and then check that $\mathcal{L}(P) = \emptyset$, with complexity which is exponential in n [Kur94b]. The motivation for constructing a reduction hierarchy is to reduce the complexity of checking (†) to $\beta \cdot |V|$. Clearly, if we succeed in constructing a reduction hierarchy for the problem (†), then (†) indeed holds.

Conversely, if (†) holds, we can always construct a tree reduction hierarchy of exponential bound for it, simply by putting above each consecutive pair of nodes P_{2i+1} and $P_{2(i+1)}$ a node P'_i with $\mathcal{L}(P'_i) = \mathcal{L}(P_{2i+1}) \cap \mathcal{L}(P_{2(i+1)})$, for $i = 1, \ldots, \lceil \frac{n-2}{2} \rceil$, and then repeating for subsequent levels. The significance of reduction hierarchy is that, besides the flexibility of choosing the order in which the P_i's are combined and the topology of the tree, it allows the languages of the internal nodes of the hierarchy to *strictly* contain the intersection of their successor's languages; this is a novel. This flexibility allows us often to succeed in keeping the sizes of the automata small, as demonstrated in the example below.

Example 2.2 Given an undirected graph $H = \langle U, R \rangle$, a *2-coloring* of H is a function $C : U \to \{W, B\}$ that "colors" each node by either W (white) or B (black). A 2-coloring C is *legal* iff all two adjacent nodes are colored in different colors. That is, formally, iff for every edge $(u_1, u_2) \in R$, we have $C(u_1) \neq C(u_2)$. Given a graph H, we say that H is 2-colorable iff there exists a legal 2-coloring of H. Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of n nodes. A graph $H = \langle U, R \rangle$ is a *ring* iff $R = \{(u_1, u_2), (u_2, u_3), \ldots, (u_{n-1}, u_n), (u_n, u_1)\}\}$. Given a ring $H = \langle U, R \rangle$, consider the 2-coloring C where for all i we have $C(u_i) = W$ iff i is odd. It is easy to see that when n is even, C is a legal 2-coloring. It is not hard to see that no legal 2-coloring exists for H in which n is odd. We show that the latter can be proved using a tree reduction hierarchy of a constant

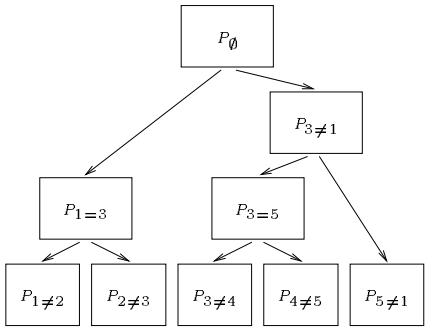
size. In particular, we show that the order in which leaves of the hierarchy are coupled together is of a great importance: a hierarchy of a constant bound exists only for some orders.

We first have to formalize 2-colorability in terms of automata. Let the alphabet be U (the set of nodes). Each (finite or infinite) word w over U induces a coloring C_w where for every node u, we have $C_w(u) = W$ if u appears in w, and $C_w(u) = B$ otherwise. Given two nodes u_i and u_j, we can easily define fixed-size automata $P_{i \neq j}$ and $P_{i=j}$, where

- $\mathcal{L}(P_{i \neq j}) = \{w \mid C_w(u_i) \neq C_w(u_j)\}$.
- $\mathcal{L}(P_{i=j}) = \{w \mid C_w(u_i) = C_w(u_j)\}$.

Proving that no 2-coloring exists now amounts to proving the emptiness of the intersection $\mathcal{L}(P_{1\neq 2}) \cap \mathcal{L}(P_{2\neq 3}) \cap \cdots \cap \mathcal{L}(P_{n\neq 1})$; that is, to proving the containment of the intersection in the language of an automaton $P_\emptyset$ that rejects all words.

In the figure below we present a tree reduction hierarchy of a constant bound for this emptiness problem, for the case $n = 5$. The idea behind the reduction is very simple: whenever we couple together automata that take care of edges with a joint node, we can "forget" the color of their joint node. Intuitively, each step in the reduction can be viewed as creating a new ring, with two less nodes. For example, replacing $P_{1\neq 2}$ and $P_{2\neq 3}$ by $P_{1=3}$, corresponds to a new ring with three nodes: the node u_2 is no longer a node in the new ring, and the nodes u_1 and u_3 are unified into a single new node. So, an order in which automata that take care of adjusted edges are coupled together leads to a tree reduction hierarchy of a constant bound.



We now show that coupling together automata that take care of adjacent edges is crucial: other orders are not necessarily successful, and might result in a tree reduction hierarchy of an exponential bound. Suppose we start by coupling together the automata $P_{1\neq 2}$ and $P_{3\neq 4}$. An automaton that replaces them must accept at least all words in which nodes u_1 and u_2 are colored differently and nodes u_3 and u_4 are colored differently. Can it accept more words? Not really. We can think of some unnatural extension to its language, but, unlike in the above orders, we can not "forget" any of the nodes u_1, u_2, u_3, or u_4. Releasing a requirements on a node u amounts to releasing the requirement of legal coloring with respect to the edges that have u as one of their nodes, leading to an illegal 2-coloring of the odd-size ring. Since there are exponentially many possible 2-coloring of nodes that are not related by edges, remembering them all induces a tree reduction hierarchy of an exponential bound.

Note that the alphabet of the automata in Example 2.2 is of size $O(n)$. Indeed, we do not restrict the problem (†) to automata with a fixed-size alphabet. Recall that in COSPAN, the

alphabet of the automata consists of atoms of a Boolean algebra. These atoms correspond to assignments to the program's variables, thus their number grows with n [Kur94b].

So, reduction hierarchies can be very efficient. Do reduction hierarchy always exist? When are they likely to exist? How can one find a reduction hierarchy? We will try to answer these questions, and we start with tree reduction hierarchies.

3 Properties of Tree Reduction Hierarchies

As discussed in [Kur94a], since the emptiness problem $\mathcal{L}(P_1) \cap \mathcal{L}(P_2) \cap \cdots \cap \mathcal{L}(P_n) = \emptyset$ is PSPACE-hard [Koz77], polynomial tree reduction hierarchies do not always exist, assuming NP $\neq$ PSPACE. To see this, assume, by way of contradiction, that for any $P_1, \ldots, P_n$ with an empty intersection, we can find a polynomial tree reduction hierarchy. We show that then, the emptiness problem is in NP. The number of nodes in a tree reduction hierarchy with n leaves is $2n$, and they are all of size polynomial in n. Thus, the whole hierarchy is of size polynomial in n. As the containment problem $\mathcal{L}(v) \supseteq \mathcal{L}(v_1) \cap \mathcal{L}(v_2)$ for Streett automata v, v_1, and v_2 is in NLOGSPACE [CDK93] (and therefore also in PTIME), we could solve the emptiness problem by guessing a polynomial hierarchy and checking it in polynomial time. A technical point in the above argument is that in the definition of the reduction hierarchy we required the P_i's to be of constant size, whereas the automata P_i in the PSPACE-hardness proof of [Koz77] are not of constant size. However, we can show that the PSPACE-hardness holds even for constant size (in fact, 2-state) automata.

Consider a tree reduction hierarchy $G = \langle V, E \rangle$ for L rooted at t. For a node $v \in V$ and a leaf $l \in L$, we say that v *controls* l if there exists a path from v to l. Otherwise, v *frees* l. We denote by $C(v)$ and $F(v)$ the set of leaves that v controls and frees, respectively. For a node v in the hierarchy, let $depth(v)$ denote the length of the longest path from v to a leaf in L. That is, for $v \in L$ we have $depth(v) = 0$ and for $v \notin L$ with $E(v, v_1)$ and $E(v, v_2)$, we have $depth(v) = 1 + \max\{depth(v_1), depth(v_2)\}$. For example, the node $P_{3\neq 1}$ in Example 2.2 has $depth(P_{3\neq 1}) = 2$, it controls the leaves $P_{3\neq 4}, P_{4\neq 5}$, and $P_{5\neq 1}$, and it frees the leaves $P_{1\neq 2}$ and $P_{2\neq 3}$.

Given a node v in a tree reduction hierarchy, we can easily come up with a nontrivial lower bound on the language of v. Indeed, every word that belongs to all the leaves in $C(v)$ must be a member of v as well. As we show in the lemma below, it is also possible to come with a nontrivial upper bound on the language of v. For every node v, let

$$LB(v) = \bigcap_{l \in C(v)} \mathcal{L}(l) \qquad \text{and} \qquad UB(v) = \mathcal{L}(t) \cup \bigcup_{l \in F(v)} \overline{\mathcal{L}(l)}.$$

Lemma 3.1 *Let $G = \langle V, E \rangle$ be a tree reduction hierarchy for L and t. For every node $v \in V$, we have $LB(v) \subseteq \mathcal{L}(v) \subseteq UB(v)$.*

Proof: We first prove, by induction on the depth of v, that $LB(v) \subseteq \mathcal{L}(v)$. If $depth(v) = 0$, then $v \in L$, $C(v) = \{v\}$, and we are done. Assume now that $LB(v) \subseteq \mathcal{L}(v)$ for nodes v of depth

at most d. Let v be a node of depth $d+1$ and let v_1 and v_2 be the two successors of v. By the induction hypothesis (which is applicable to v_1 and v_2), we have that $LB(v_1) \subseteq \mathcal{L}(v_1)$ and $LB(v_2) \subseteq \mathcal{L}(v_2)$. By requirement (R2) of reduction hierarchies, $\mathcal{L}(v_1) \cap \mathcal{L}(v_2) \subseteq \mathcal{L}(v)$. Hence, as $C(v) = C(v_1) \cup C(v_2)$, we are done.

We now prove that $\mathcal{L}(v) \subseteq UB(v)$. Intuitively, this follows from the fact that if w is in $\mathcal{L}(v)$, then either it is also in $\mathcal{L}(t)$, or that there exists a leaf $l \in F(v)$ such that $w \notin \mathcal{L}(l)$. Indeed, if w is a member of both $\mathcal{L}(v)$ and the languages of all the free leaves, then is must be a member of the languages of all the nodes in G that have a path to v, and in particular of $\mathcal{L}(t)$. Formally, consider the hierarchy G' obtained from G by taking out all the nodes reachable from v (excluding v). The leaves of G' are $\{v\} \cup F(v)$. Since t controls all leaves, we have a lower bound $\bigcap_{l \in \{v\} \cup F(v)} \mathcal{L}(l)$ on $\mathcal{L}(t)$. Hence, $\mathcal{L}(v) \subseteq \mathcal{L}(t) \cup \overline{(\bigcap_{l \in F(v)} \mathcal{L}(l))}$, and we are done. □

Lemma 3.1 has two applications. First, we use it as a "constraint" while constructing hierarchies. Consider a procedure that constructs tree reduction hierarchies. Such a procedure gets L and t as inputs and repeatedly replaces two nodes v_1 and v_2 by a node v for which $\mathcal{L}(v) \supseteq \mathcal{L}(v_1) \cap \mathcal{L}(v_2)$. Doing so, the procedure is eventually left with two nodes u_1 and u_2 and checks whether $\mathcal{L}(t) \supseteq \mathcal{L}(u_1) \cap \mathcal{L}(u_2)$. If this containment holds, then the original containment holds too. What if this containment does not hold? Then there are two possibilities. It might be that the original containment does not hold either. It might, however, also be that in some of its steps, the procedure "relaxes too much". That is, it replaces v_1 and v_2 with an automaton v that contains also words not in the upper bound of v. These words survive the intersections to the top of the hierarchy and cause the last containment check to fail. Using Lemma 3.1, we can bound each of the internal nodes of the hierarchy and prevent the latter possibility. Lemma 3.1 also helps us to estimate when tree reduction hierarchies are likely to exist. The bigger is the gap between the lower and the upper bounds of the internal nodes, the more likely we are to find a hierarchy. Intuitively, big gaps give us flexibility in defining the internal nodes. This flexibility is what we hope to "translate" to small automata. In Lemma 3.2 below, we formalize this intuition.

Lemma 3.2 *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two ω-regular languages over an alphabet Σ. If there exists a set $\{h_1, \ldots, h_k\} \subset \Sigma^*$ such that for every h_i and h_j with $i \neq j$ there exists $w \in \Sigma^\omega$ such that $h_i \cdot w \in \mathcal{L}_1$ and $h_j \cdot w \notin \mathcal{L}_2$, then any deterministic automaton $\mathcal{A}$ for which $\mathcal{L}_1 \subseteq \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}_2$ has at least k states.*

Proof: For a deterministic Streett automaton $\mathcal{A}$ and a finite word $h \in \Sigma^*$, let $r(h)$ be the state reached in $\mathcal{A}$ by reading h. Given a set $\{h_1, \ldots, h_k\}$ as above and a deterministic automaton $\mathcal{A}$, we show that if $\mathcal{L}_1 \subseteq \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}_2$, then for every $1 \leq i < j \leq k$, we have that $r(h_i) \neq r(h_j)$. Assume, by way of contradiction, that there exist $1 \leq i < j \leq k$ with $r(h_i) = r(h_j) = q$. Consider the automaton $\mathcal{A}'$ obtained from $\mathcal{A}$ by making q the initial state. Let w be the infinite tail distinguishing between h_i and h_j. That is, $h_i \cdot w \in \mathcal{L}_1$ and $h_j \cdot w \notin \mathcal{L}_2$. Since $h_i \cdot w \in \mathcal{L}_1$ and $\mathcal{L}_1 \subseteq \mathcal{L}(\mathcal{A})$, the automaton $\mathcal{A}'$ accepts w. On the other hand, since $h_j \cdot w \notin \mathcal{L}_2$ and $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}_2$, the automaton $\mathcal{A}'$ rejects w, and we reach a contradiction. □

Lemma 3.2 can be viewed as analogous to the tight bound of Myhill and Nerode on the size of the smallest automaton recognizing a certain regular language. We strengthen their result by handling ω-regular languages and by handling all automata whose language lies between two given languages. We weaken their result by providing only a lower bound on the size of the automaton. In Section 4, we use Lemmas 3.1 and 3.2 in order to prove that, independent of the NP=PSPACE question, tree reduction hierarchies of a polynomial bound do not always exist. For that , we also need the following lemma.

Lemma 3.3 *Let $G = \langle V, E\rangle$ be a tree reduction hierarchy for L and t. There exists a node $v \in V$ such that $\frac{|L|}{4} < C(v) \leq \frac{|L|}{2}$.*

Proof: Let $|L| = n$. Assume, by way of contradiction, that for all nodes v in G either $|C(v)| \leq \frac{n}{4}$ ("v is small") or $|C(v)| > \frac{n}{2}$ ("v is big"). We know that t is big, whereas all the leaves of G are small. Hence, there must be a node v in the hierarchy such that v is big and both its successors, v_1 and v_2, are small. Since, however, $C(v) = C(v_1) \cup C(v_2)$, such a node v satisfies $\frac{n}{2} < |C(v)| \leq |C(v_1)| + |C(v_2)| \leq \frac{n}{2}$, and we reach a contradiction. □

4 Existence and Nonexistence of Tree Reduction Hierarchies

In this section we prove an exponential lower bound on the size of tree hierarchies in the general case. Furthermore, we demonstrate the sensitivity of the existence of tractable tree reduction hierarchies. Specifically, we present two families of language-containment problems. The families are very similar, yet one of them admits tree reduction hierarchies of a constant bound, and the second admits only tree reduction hierarchies of an exponential bound. The second family also proves that tree reduction hierarchies of polynomial bound do not always exist.

Let $H = \langle U, R\rangle$ be an undirected graph such that the following hold.

1. $|U| = n$ is odd.
2. H has no self loops; i.e., there exists no $u \in U$ with $(u, u) \in R$.
3. H has maximal degree d, for some constant d; i.e., for every $u \in U$ there exist at most d nodes u' with $(u, u') \in R$.
4. H has expansion rate c, for some constant c; i.e., for every $S \subseteq U$ with $|S| \leq \frac{n}{2}$, we have $|cut(S)| \geq c \cdot |S|$, where $cut(S) = \{(u, u') \mid u \in S \text{ and } u' \notin S\}$.
5. R contains a Hamilton circuit $C = \{(u_1, u_2), (u_2, u_3), \ldots, (u_{n-1}, u_n), (u_n, u_1)\}\}$.

It is well known that such *expander graphs* exist for degree $d = 3$ or more (the expansion rate c is a constant that depends on the degree) that satisfy properties 2-4, see eg. [Ajt87]. The

usual definition of expander graphs does not require Hamiltonicity or an odd number of nodes (properties 1 and 5), but these are easy to achieve by simple modifications.

Let the alphabet be R, the set of H's edges. Each finite word [1] w over R defines a subgraph H_w where for all edges $e \in R$,

$$e \text{ is in } H_w \text{ iff } e \text{ has an odd number of occurrences in } w.$$

With each node $u_i \in U$, we associate an automaton P_i such that

$$\mathcal{L}(P_i) = \{w \mid \text{node } u_i \text{ has an odd degree in } H_w\}.$$

Since n is odd, there exists no subgraph of H in which all nodes have an odd degree (in any graph, there must be an even number of nodes with an odd degree). Hence, the intersection of the $\mathcal{L}(P_i)$ is empty. Also, $\mathcal{L}(P_i)$ can be recognized by a fixed-size automaton that keeps track of all edges incident to u_i whether they have appeared an odd or even number of times (note there is no need to distinguish between different edges: the automaton P_i accepts a word w iff the node u_i has an odd total number of occurrences of incident edges in w).

Proposition 4.1 *For every odd n, the emptiness problem $\mathcal{L}(P_1) \cap \mathcal{L}(P_2) \cap \cdots \cap \mathcal{L}(P_n) = \emptyset$ admits a tree reduction hierarchy of a constant bound.*

Proof: Recall that the automaton P_i records parity of the edges incident to u_i. We can combine the automata arbitrarily in a tree, letting the automaton at each node to record the parity of edges in the cut of the graph that corresponds to the leaves under its control. Thus, an automaton v with $C(v) = S$ accepts all words w with $\mathit{parity\_in\_w}(|\mathit{cut}(S)|) = \mathit{parity}(|S|)$. It is easy to see that all automata v in the hierarchy need only two states. If we continue building the hierarchy as above, we end up with two automata. One controls a set S of nodes and the second controls $U \setminus S$. Since n is odd, S and $U \setminus S$ differ in their parity. On the other hand, $\mathit{cut}(S) = \mathit{cut}(U \setminus S)$. Therefore, the intersection of the two automata is empty, and we are done. □

We now present a variant of the setting above for which no tree reduction hierarchy of a constant, or even polynomial, bound exists. Now, the subgraph H_w defined by each word w is such that for every edge $e \in R$,

$$e \text{ is in } H_w \text{ iff } e \text{ has two or more occurrences in } w.$$

With each node $u_i \in U$, we associate an automaton P_i such that

$$\mathcal{L}(P_i) = \{w \mid \text{node } u_i \text{ has an odd degree in } H_w\}.$$

[1] The examples we present consider finite words, and can be easily extended to consider infinite words as well; e.g., by concatenating a suffix $\#^\omega$ to all words.

Again, as n is odd, there exists no subgraph of H in which all nodes have an odd degree. Hence, the intersection of the $\mathcal{L}(P_i)$ is empty. The language $\mathcal{L}(P_i)$ can be recognized by an automaton that keeps track of each edge incident to u_i whether it has appeared 0, 1, or 2, or more times. Since the maximal degree of H is fixed, so are the sizes of the automata.

Proposition 4.2 *For every odd n, any tree reduction hierarchy for the emptiness problem $\mathcal{L}(P_1) \cap \mathcal{L}(P_2) \cap \cdots \cap \mathcal{L}(P_n) = \emptyset$ contains an automaton with at least $2^{(\frac{c \cdot n}{4})}$ states.*

Proof: Consider a tree reduction hierarchy $G = \langle V, E \rangle$ for the problem. By Lemma 3.3, there exists a node v of G such that $\frac{n}{4} < C(v) \leq \frac{n}{2}$. Let S be the set of nodes of H that corresponds to the leaves in $C(v)$. By the above, $\frac{n}{4} < |S| \leq \frac{n}{2}$.

For every set $Q \subseteq cut(S)$ of edges, let $x(Q)$ be a word which consists of the edges of Q, each once (in some arbitrary order). Let $K = \bigcup_{Q \subseteq cut(S)} \{x(Q)\}$ be the set of words corresponding to all subsets of edges in $cut(S)$. Since H has expansion rate c, the size of $cut(S)$ is at least $c \cdot |S|$, thus the size of K is at least $2^{c \cdot |S|}$. We show that the set K satisfies the requirements of Lemma 3.2 with $\mathcal{L}_1 = LB(v)$ and $\mathcal{L}_2 = UB(v)$. Hence, by Lemmas 3.1 and 3.2, the automaton v has at least $2^{c \cdot |S|}$ states.

Let $x(Q_1)$ and $x(Q_2)$ be two words in K. Let $e = (u, u')$ be an edge in the symmetric difference of the sets Q_1 and Q_2, say $e \in Q_2 \setminus Q_1$, with $u \in S$ and $u' \in U \setminus S$. Recall that H contains a Hamilton circuit C. Since n is odd, we can pick a set M of edges of C which induces a matching that covers all nodes of H except u'; i.e., in the subgraph M of H, all nodes have degree 1, except for u' that has degree 0.

Let t be a word over R that consists of the edge e once and the edges of M twice. Consider the words $x(Q_1) \cdot t$ and $x(Q_2) \cdot t$, and the subgraphs H_1 and H_2 corresponding to them. Note that $H_1 = M$ and $H_2 = M \cup \{e\}$. We prove that $x(Q_1) \cdot t \in LB(v)$ and $x(Q_2) \cdot t \notin UB(v)$. Consider first $x(Q_1) \cdot t$. Recall that in M, all nodes $u_i \in S$ have degree 1 (only u' has degree 0, but u' is not in S), therefore, $x(Q_1) \cdot t$ is accepted by all P_i for $u_i \in S$, hence $x(Q_1) \cdot t \in LB(v)$. Consider now $x(Q_2) \cdot t$. All nodes $u_i \in U \setminus S$ have degree 1 in $M \cup e$ (only u has degree 2, but u is in S). Therefore their corresponding automata P_i accept $x(Q_2) \cdot t$. Hence, $x(Q_2) \cdot t \notin UB(v)$. □

5 Discussion

The nonexistence example provides us with some directions as to when tree reduction hierarchies are likely to exist. We saw the importance of gaps between the lower and upper bounds on the languages of automata that are candidates for serving as intermediate nodes in a tree reduction hierarchy. Small gaps between the bounds prevent us from doing significantly better than defining the conventional product of the processes in the leaves. This observation provides us with directions in a search for tree reduction hierarchies. We are likely to get large gaps between the lower and the upper bounds when we couple together processes with shared variables.

Then, variables that both processes ignore induce the desired gap between the bounds (see also Example 2.2). The preceding example may suggest that many practical problems will have a tree reduction hierarchy of a constant bound. It would be of immense practical value to have an algorithm or even a good heuristic for finding such a hierarchy when it exists.

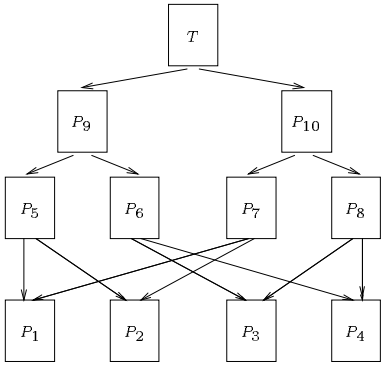
Since the size of a tree reduction hierarchy of a constant bound is polynomial in the size of its leaves and root, the problem of finding such a hierarchy is in NP (given a set of leaves and a root, simply guess a tree reduction hierarchy). We believe the problem is NP-complete, though we still have not come up with a lower bound.

Much less we know about DAG hierarchies. When we refer to the size of a DAG hierarchy, there are two parameters to consider. The first, as in tree hierarchies, is the size of the intermediate automata. In addition, we consider the number of intermediate automata. Unlike in tree hierarchies, where this number is bounded by the number of leaves, here there is no bound on the number of times a leaf can be "used", and thus, there is no bound on the number of nodes in the hierarchy. If we require both the number of the automata and their size to be bounded by polynomial, then by the same complexity-theoretic arguments as in the tree case, we know that if NP $\neq$ PSPACE, then in general there may not exist such DAG hierarchies. In this case however, we do not have an explicit proof, independent of the complexity-theoretic conjecture.

In order to see the strength of DAG reduction hierarchies with respect to tree reduction hierarchies, consider the upper bound proven in Lemma 3.1. The bound emerges from the fact that once we control a leaf, we can no longer use it again (that is, for every node v, we have $F(v) = L \setminus C(v)$). This is not the case with DAG hierarchies. There, a leaf can be used several times, $C(v)$ and $F(v)$ are not disjoint, and in order for our upper bound to hold, we should define

$$F(v) = \{l : \text{ there exists a path from } t \text{ to } l \text{ that does not meet } v\}.$$

It is not hard to see that with this definition of $F(v)$, Lemma 3.1 is valid also for DAG reduction hierarchies. Nevertheless, as $F(v)$ is now bigger, so is the upper bound it induces, and Lemma 3.1 is not of much help. To see this, consider the DAG reduction hierarchy below.



As $F(P_9) = F(P_{10}) = L$, the upper bound of all nodes is the trivial Σ^ω bound. The strength

of DAG reduction hierarchies raises several open problems. In particular, we do not even know whether or not DAG hierarchies of polynomial bound on the size of the automata always exist (note that we cannot use the PSPACE hardness of the problem (†) in order to refute their existence if the DAG is not restricted to have a polynomial number of nodes). Naturally, we are interested in finding an algorithm that "does its best" to construct DAG hierarchies, employing their extra flexibility with respect to tree reduction hierarchies.

From a logical point of view, reduction hierarchies can be viewed as a proof system for proving the containment (†). Generally, proof systems employ a set of axioms and inference rules to derive from a given set of assertions another assertion. In our case, an automaton P can be thought of as a representation for the assertion "a string w is accepted by P", where w is a variable string (in the context of verification, the string w encodes a computation). The containment (†) says that the conjunction of the assertions corresponding to the automata P_i imply the assertion corresponding to the task automaton T. Our proof system has just one inference rule: from automata P_1 and P_2 we can infer the automaton P_{12} provided that the intersection of $\mathcal{L}(P_1)$ and $\mathcal{L}(P_2)$ is a subset of $\mathcal{L}(P_{12})$. Note that this rule has the important property that it can be checked efficiently (i.e., in time polynomial in the size of the automata P_1, P_2, and P_{12}). Clearly, proofs in this proof system correspond exactly to DAG reduction hierarchies. The questions discussed in this paper in the context of computer-aided verification can accordingly be viewed as questions about proof complexity in this proof system, namely whether it is possible to find, for every containment (†) a proof in which the "length of the lines" in the proof (the size of the automata) is polynomially bounded. Notice that in terms of the more traditional measure of proof complexity, the number of lines of the proof, there is always a short proof, one that uses n (long) lines.

References

[Ajt87] M. Ajtai. Recursive construction for 3-regular expanders. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 295–304, 1987.

[And95] H.R. Andersen. Partial model checking. In *Proc. 10th IEEE Symposium on Logic in Computer Science*, San Diego, June 1995.

[BS90] R.B. Boppana and M. Sipser. The complexity of finite functions. *Handbook of Theoretical Computer Science*, pages 759–804, 1990.

[CDK93] E. M. Clarke, I. A. Draghicescu, and R. P. Kurshan. A unified approach for showing language containment and equivalence between various types of ω-automata. *Information Processing Letters* **46**, pages 301–308, (1993).

[CG87] E.M. Clarke and O. Grumberg. Avoiding the state explosion problem in temporal logic model-checking algorithms. In *Proc. 6th ACM Symposium on Principles of Distributed Computing*, pages 294–303, Vancouver, British Columbia, August 1987.

[Koz77] D. Kozen. Lower bounds for natural proof systems. In *Proc. 18th IEEE Symposium on Foundation of Computer Science*, pages 254–266, 1977.

[Kur94a] R. P. Kurshan. The complexity of verification. In *Proc. 26th ACM Symposium on Theory of Computing*, pages 365–371, Montreal, 1994.

[Kur94b] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.