

An Automaton Learning Approach to Solving Safety Games over Infinite Graphs

Daniel Neider

Department of Electrical Engineering
University of California at Los Angeles, USA

Ufuk Topcu

Department of Electrical and Systems Engineering
University of Pennsylvania, USA

Abstract—We propose a method to construct finite-state reactive controllers for systems whose interactions with their adversarial environment are modeled by infinite-duration two-player games over (possibly) infinite graphs. The proposed method targets safety games with infinitely many states or with such a large number of states that it would be impractical—if not impossible—for conventional synthesis techniques that work on the entire state space. We resort to constructing finite-state controllers for such systems through an automata learning approach, utilizing a symbolic representation of the underlying game that is based on finite automata. Throughout the learning process, the learner maintains an approximation of the winning region (represented as a finite automaton) and refines it using different types of counterexamples provided by the teacher until a satisfactory controller can be derived (if one exists). We present a symbolic representation of safety games (inspired by regular model checking), propose implementations of the learner and teacher, and evaluate their performance on examples motivated by robotic motion planning in dynamic environments.

I. INTRODUCTION

We propose an automata learning-based method to construct reactive controllers subject to safety specifications. We model the interaction between a controlled system and its possibly adversarial environment as a two-player game over a graph [1]. We consider games over *infinite graphs*. In this setting, the conventional techniques for reactive controller synthesis (e.g., fixed-point computations) are not applicable anymore. Therefore, we resort to a *learning*-based approach for constructing finite-state reactive controllers for the controlled system. The learning takes place in a setting akin to *counterexample-guided inductive synthesis* (CEGIS) [2] between a *teacher*, who has knowledge about the safety game in question, and a *learner*, whose objective is to identify a controller using information disclosed by the teacher in response to (incorrect) conjectures.

A natural context for the proposed method is one in which the interaction between the controlled system and its environment is so complex that it can be represented only by graphs with infinitely many vertices (e.g., motion planning over unbounded grid worlds) or “practically infinitely many” states (i.e., the number of possible configurations is so large that the game becomes impractical for conventional techniques). Additionally, in situations where a complete description of the game is not available in a format amenable to existing game solvers [3], [4], there may still exist human experts (or automated oracles, as in Section IV) who have sufficient insight into how the controlled system should behave and can act as teacher.

We focus on games with safety specifications, which already capture practically interesting properties (e.g., safety and bounded-horizon reachability). However, games over infinite graphs require special attention on the representation and manipulation of the underlying graph structure. Hence, one of our main contributions is a symbolic representation of safety games, called *rational safety games*, that follows the idea of *regular model checking* [5] in that it represents sets of vertices by regular languages and edges by so-called rational relations.

A straightforward approach to solve (rational) safety games is computing a *winning set* for the controlled system (i.e., a safe subset of the vertices in which the system can force to remain). Once a winning set is computed, a strategy for the system is determined by choosing its moves (in each of its turns) to stay inside the set, which is possible regardless of the moves of the environment. We use winning sets as a proxy for an actual controller, and the objective of the learning task is the construction of a winning set. In fact, learning a winning set rather than a controller results in more permissive strategies (and potentially smaller solutions) as the moves of the system do not need to be fixed during the learning process.

We develop a framework for learning winning sets for rational safety games and particular implementations of a teacher and learner. The actual learning works iteratively. In each iteration, the learner conjectures a winning set, represented as a deterministic finite automaton. The teacher performs a number of checks and returns, based on whether the conjecture passes the checks, a counterexample. Following the ICE learning framework [6] and partially deviating from the classical learning frameworks for regular languages [7], [8], the counterexample may be one of the following four types: positive, negative, existential implication and universal implication counterexamples. Based on the response from the teacher, the learner updates his conjecture. If the conjecture passes all checks (i.e., the teacher returns no counterexample), the learning process terminates with the desired controller.

A learning-based approach offers several advantages: First, even though the underlying game may be prohibitively large, the reactive controller necessary to realize the specifications often has a compact representation in practice; for example, depending on the given task specification in a robotic motion planning scenario, only a small subset of all possible rich interactions between the robot and its dynamic environment over a possibly large workspace is often relevant. Second, since

learning-based approaches usually identify “small” solutions (as they typically produce intermediate conjectures of increasing size), their runtime mainly depends on the size of the solution rather than the size of the underlying game. Third, learning-based approaches reduce the gap between human designers and construction of reactive controllers by hiding the complexity of the underlying game from the learner.

Finally, we demonstrate the use of our overall learning-based framework empirically on a series of examples motivated by robotic motion planning in dynamic environments.

Related Work: Games over infinite graphs have been studied in the past, predominantly in the case of games over pushdown graphs [9]. The games we consider here, however, are played over a richer class of graphs and require different techniques to be solved. Also, a constraint-based approach to solving games over infinite graphs has recently been proposed [10].

Learning-based techniques for games over infinite graphs have already been studied in the context of reachability games [11]; in fact, our symbolic representation of safety games is a generalization of the representation proposed there. In the context of safety games, recent work [12] has already demonstrated the ability of learning-based approaches to extract small reactive controllers from a priori constructed controllers with possibly large number of states. In this work, we by-pass the a priori construction of possibly large reactive controllers by learning (an appropriate representation of) a controller directly.

II. RATIONAL SAFETY GAMES

This section recaps infinite-duration, two-player safety games as well as basic concepts of automata theory and introduces rational safety games.

a) Safety Games: We consider safety games (i.e., infinite duration two-person games on graphs) as popularized by McNaughton [1]. A safety game is played on an *arena* $\mathfrak{A} = (V_0, V_1, E)$ consisting of two nonempty, disjoint sets V_0, V_1 of *vertices* (we denote their union by V) and a directed edge relation $E \subseteq V \times V$. In contrast to the classical (finite) setting, we allow V_0 and V_1 to be countable sets. As shorthand notation, we write the successors of a set $X \subseteq V$ of vertices as $E(X) = \{y \mid \exists x \in X: (x, y) \in E\}$.

We consider safety games with initial vertices, which are defined as triples $\mathfrak{G} = (\mathfrak{A}, F, I)$ consisting of an arena $\mathfrak{A} = (V_0, V_1, E)$, a set $F \subseteq V$ of *safe vertices*, and a set $I \subseteq F$ of *initial vertices*. Such safety games are played by two players, named Player 0 and Player 1, as follows: A token is placed on some initial vertex $v_0 \in I$ and, in each turn, the player owning the current vertex moves the token to a successor vertex of his choice. This process of moving the token is repeated ad infinitum, thereby forming an infinite sequence of vertices, which is called a *play*. Formally, a *play* is an infinite sequence $\pi = v_0 v_1 \dots \in V^\omega$ that satisfies $v_0 \in I$ and $(v_i, v_{i+1}) \in E$ for all $i \in \mathbb{N}$. The set F defines the winning condition of the game in the sense that a play $v_0 v_1 \dots$ is *winning for Player 0* if $v_i \in F$ for all $i \in \mathbb{N}$ —otherwise it is *winning for Player 1*.

A strategy for Player σ , $\sigma \in \{0, 1\}$, is a mapping $f_\sigma: V^* V_\sigma \rightarrow V$, which prescribes how to continue playing. A

strategy f_σ is called *winning* if any play $v_0 v_1 \dots$ that is played according to the strategy (i.e., that satisfies $v_{i+1} = f_\sigma(v_0 \dots v_i)$ for all $i \in \mathbb{N}$ and $v_i \in V_\sigma$) is winning for Player σ . A winning strategy for Player 0 straightforwardly translates into a controller satisfying the given safety specifications and, hence, we restrict ourselves to compute winning strategies for Player 0.

Computing a winning strategy for Player 0 is usually reduced to finding a so-called winning set.

Definition 1 (Winning set): For a safety game $\mathfrak{G} = (\mathfrak{A}, I, F)$ over the arena $\mathfrak{A} = (V_0, V_1, E)$, a winning set is a set $W \subseteq V$ satisfying (1) $I \subseteq W$, (2) $W \subseteq F$, (3) $E(\{v\}) \cap W \neq \emptyset$ for all $v \in W \cap V_0$ (existential closedness), and (4) $E(\{v\}) \subseteq W$ for all $v \in W \cap V_1$ (universal closedness).

By computing a winning set, one immediately obtains a strategy for Player 0: starting in an initial vertex, Player 0 simply moves to a successor vertex inside W whenever it is his turn. A straightforward induction over the length of plays proves that every play that is played according to this strategy stays inside F , no matter how Player 1 plays, and, hence, is won by Player 0 (since $I \subseteq W \subseteq F$). A winning set is what we want to compute—or, more precisely, *learn*.

Games over infinite arenas require a symbolic representation in order to work with them algorithmically. We follow the idea of *regular model checking* [5], an approach in verification, and represent sets of vertices by regular languages and edges by so-called rational relations. Before we can introduce our symbolic representation of safety games, however, we need to recap basic concepts and notations of automata theory.

b) Basics of Automata Theory: An *alphabet* Σ is a nonempty, finite set, whose elements are called *symbols*. A *word* over the alphabet Σ is a sequence $u = a_1 \dots a_n$ of symbols $a_i \in \Sigma$ for $i \in \{1, \dots, n\}$; the empty sequence is called *empty word* and denoted by ε . Given two words $u = a_1 \dots a_m$ and $v = b_1 \dots b_n$, the *concatenation of u and v* is the word $u \cdot v = uv = a_1 \dots a_m b_1 \dots b_n$. The set of all words over the alphabet Σ is denoted by Σ^* , and a subset $L \subseteq \Sigma^*$ is called a *language*. The set of prefixes of a language $L \subseteq \Sigma^*$ is the set $\text{Pref}(L) = \{u \in \Sigma^* \mid \exists v \in \Sigma^*: uv \in L\}$.

A *nondeterministic finite automaton (NFA)* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$ consisting of a nonempty, finite set Q of *states*, an *input alphabet* Σ , an *initial state* $q_0 \in Q$, a *transition relation* $\Delta \subseteq Q \times \Sigma \times Q$, and a set $F \subseteq Q$ of *final states*. A *run* of an NFA \mathcal{A} on a word $u = a_1 \dots a_n$ is a sequence of states q_0, \dots, q_n such that $(q_{i-1}, a_i, q_i) \in \Delta$ for $i \in \{1, \dots, n\}$. We denote this run by $\mathcal{A}: q_0 \xrightarrow{u} q_n$. An NFA \mathcal{A} *accepts* a word $u \in \Sigma^*$ if $\mathcal{A}: q_0 \xrightarrow{u} q$ with $q \in F$. The set $L(\mathcal{A}) = \{u \in \Sigma^* \mid \mathcal{A}: q_0 \xrightarrow{u} q, q \in F\}$ is called *language of \mathcal{A}* . A language L is said to be *regular* if there exists an NFA \mathcal{A} with $L(\mathcal{A}) = L$. Finally, NFA_Σ denotes the set of all NFAs over Σ .

A *deterministic finite automaton (DFA)* is an NFA in which $(p, a, q) \in \Delta$, $(p, a, q') \in \Delta$ implies $q = q'$. We replace the transition relation Δ with a transition function $\delta: Q \times \Sigma \rightarrow Q$.

We define rational relations by resorting to transducers. A *transducer* is an NFA $\mathcal{T} = (Q, \hat{\Sigma}, q_0, \Delta, F)$ over the alphabet $\hat{\Sigma} = (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})$ — Σ and Γ are both alphabets—that processes pairs $(u, v) \in \Sigma^* \times \Gamma^*$ of words. The *run* of

a transducer \mathcal{T} on a pair (u, v) is a sequence q_0, \dots, q_n of states such that $(q_{i-1}, (a_i, b_i), q_i) \in \Delta$ for all $i \in \{1, \dots, n\}$, $u = a_1 \dots a_n$, and $v = b_1 \dots b_n$; note that u and v do not need to be of equal length since any a_i or b_i can be ε . A pair (u, v) is said to be *accepted* by \mathcal{T} if there exists a run of \mathcal{T} on (u, v) that starts in the initial state and ends in a final state. As an acceptor of pairs of words, a transducer \mathcal{T} *defines* a relation, namely the relation consisting of exactly the pairs accepted by \mathcal{T} , which we denote by $R(\mathcal{T})$. Finally, a relation $R \subseteq \Sigma^* \times \Gamma^*$ is called *rational* if there exists a transducer \mathcal{T} with $R(\mathcal{T}) = R$. (This definition of rational relations is simplified from that in [13] but sufficient for our purpose.)

Our learning framework relies on the two well-known facts.

Lemma 1: *Let $R \subseteq \Sigma^* \times \Gamma^*$ be a rational relation and $X \subseteq \Sigma^*$ a regular set. Then, (1) the relation $R^{-1} = \{(y, x) \mid (x, y) \in R\}$ is again rational, and a transducer defining this set can be constructed in linear time; and (2) the set $R(X) = \{y \in \Gamma^* \mid \exists x \in X: (x, y) \in R\}$, called the image of X under R , is again regular, and an NFA accepting this set can be constructed effectively.*

c) **Rational Safety Games:** A rational safety game is a symbolic representation of a safety game in terms of regular languages and rational relations.

Definition 2: A rational arena over the alphabet Σ is an arena $\mathfrak{A}_\Sigma = (V_0, V_1, E)$ where $V_0, V_1 \subseteq \Sigma^*$ are regular languages and $E \subseteq V \times V$ is a rational relation.

The definition of rational safety games is now immediate.

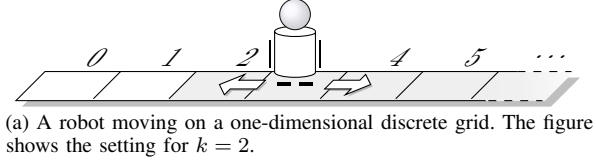
Definition 3: A rational safety game over the alphabet Σ is a safety game $\mathfrak{G}_\Sigma = (\mathfrak{A}_\Sigma, F, I)$ where \mathfrak{A}_Σ is a rational arena over Σ and $F, I \subseteq \Sigma^*$ are regular languages.

In the remainder, we assume regular languages to be given as NFAs and rational relations as transducers. In addition, we use these notions interchangeably when referring to rational arenas and rational safety games; for instance, we write a rational arena $\mathfrak{A}_\Sigma = (V_0, V_1, E)$ as $\mathfrak{A}_\Sigma = (\mathcal{A}_{V_0}, \mathcal{A}_{V_1}, \mathcal{T}_E)$ given that $L(\mathcal{A}_{V_0}) = V_0$, $L(\mathcal{A}_{V_1}) = V_1$, and $R(\mathcal{T}_E) = E$.

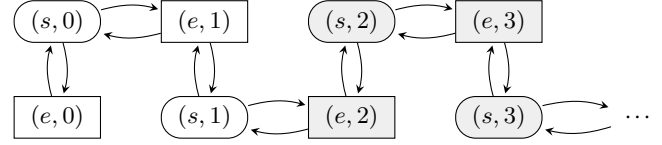
Let us illustrate rational safety games through an example.

Example 1: Consider a simple example motivated by motion planning, sketched in Figure 1a, in which a robot moves on an infinite, discrete one-dimensional grid that is “bounded on the left”. The robot can move left or right to an adjacent cell (provided that it has not reached edge of the grid) or it can stay at its current position. The grid is partitioned into a safe and an unsafe area, the former being shown shaded in Figure 1a. The safe area is parameterized by an integer $k \in \mathbb{N} \setminus \{0\}$ and consists of all position greater than or equal to k . The robot starts somewhere inside the safe area.

The robot’s movement is governed by two adversarial players, called *system* and *environment*; the system can move the robot to the right or keep it at its current position, whereas the environment can move the robot to the left (if the edge has not been reached) or keep it at its current position. The players move the robot in alternation, and the system moves first. The system’s objective is to stay within the safe area, whereas the environment wants to move the robot out of it. Note that the system can win, irrespective of k , by always moving right.



(a) A robot moving on a one-dimensional discrete grid. The figure shows the setting for $k = 2$.



(b) The safety game \mathfrak{G}_2 . Player 0 vertices are drawn as ellipses and Player 1 vertices are drawn as squares. Shaded vertices belong to F .

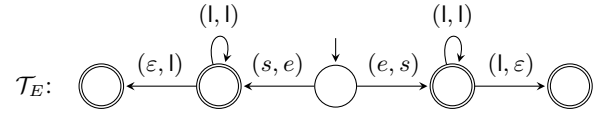
Fig. 1. Illustration of the safety game discussed in the introductory example.

A formalization as safety game is straightforward. Player 0 corresponds to the system and Player 1 corresponds to the environment. The arena $\mathfrak{A} = (V_0, V_1, E)$ consists of vertices $V_0 = \{s\} \times \mathbb{N}$ and $V_1 = \{e\} \times \mathbb{N}$ — s , respectively e , indicates the player moving next—as well as the edge relation $E = \{(s, i), (e, i+1) \mid i \in \mathbb{N}\} \cup \{(e, i+1), (s, i) \mid i \in \mathbb{N}\}$. The safety game itself is the triple $\mathfrak{G}_k = (\mathfrak{A}, F, I)$ with $F = \{s, e\} \times \{i \in \mathbb{N} \mid i \geq k\}$ and $I = \{s\} \times \{i \in \mathbb{N} \mid i \geq k\}$. Figure 1b sketches the game \mathfrak{G}_k for the case $k = 2$.

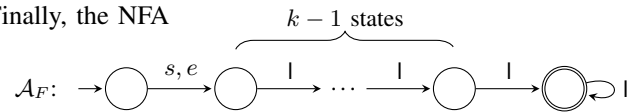
We now turn \mathfrak{G}_k into a rational safety game. To this end, we label each vertex uniquely with a finite word. In our example, we choose $\Sigma = \{s, e, l\}$ and associate the vertex $(x, i) \in \{s, e\} \times \mathbb{N}$ with the word xl^i where l^i is the encoding of i in unary. We represent the sets V_0 and V_1 by the following NFAs:



Moreover, we represent the edges by the following transducer:



Finally, the NFA



represents the set F ; similarly, I is represented by a copy of \mathcal{A}_F in which the transition labeled with e is omitted. \square

It is worth mentioning that rational arenas not only subsume finite arenas but also a rich class of infinite arenas, including such encoding computations of Turing machines. Hence, the problem of determining the winner of a rational safety game is undecidable, and any algorithm for computing a winning set can at best be a semi-algorithm (i.e., an algorithm that, on termination, gives the correct answer but does not guarantee to halt). The algorithm we design in this paper is of this kind and guarantees to learn a winning set if one exists. To ease description, we always assume that a winning set exists.

III. THE LEARNING FRAMEWORK

Our learning framework is an extension of the ICE framework proposed by Garg et. al. [6], which deals with learning loop invariants from positive and negative data as well as

implications. The learning takes place between a *teacher*, who has (explicit or implicit) knowledge about the rational safety game in question, and a *learner*, whose objective is to learn a DFA accepting a winning set, but who is agnostic to the game. We assume that the teacher announces the alphabet of the game before the actual learning starts.

The learning proceeds in a CEGIS-style loop [2]. In every iteration, the learner conjectures a DFA, let us call it \mathcal{C} , and the teacher checks whether $L(\mathcal{C})$ is a winning set—this kind of query is often called *equivalence* or *correctness query*. Although the teacher does not know a winning set (the overall objective is to learn one after all), he can resort to Conditions (1)–(4) of Definition 1 in order to decide whether $L(\mathcal{C})$ is a winning set. If $L(\mathcal{C})$ satisfies Conditions (1)–(4) (i.e., $L(\mathcal{C})$ is a winning set), then the teacher replies “yes” and the learning ends. If this is not the case, the teacher returns a *counterexample* witnessing the violation of one of these conditions, and the learning continues with the next iteration. The definition below fixes the protocol between the teacher and the learner and defines counterexamples.

Definition 4 (Teacher for rational safety games): Let $\mathfrak{G}_\Sigma = (\mathfrak{A}_\Sigma, F, I)$ be a rational safety game over the rational arena $\mathfrak{A}_\Sigma = (V_0, V_1, E)$. Confronted with a DFA \mathcal{C} , a teacher for \mathfrak{G}_Σ replies as follows:

- 1) If $I \not\subseteq L(\mathcal{C})$, then the teacher returns a positive counterexample $u \in I \setminus L(\mathcal{C})$.
- 2) If $L(\mathcal{C}) \not\subseteq F$, then the teacher returns a negative counterexample $u \in L(\mathcal{C}) \setminus F$.
- 3) If there exists $u \in L(\mathcal{C}) \cap V_0$ such that $E(\{u\}) \cap L(\mathcal{C}) = \emptyset$, then the teacher picks such a u and returns an existential implication counterexample $(u, \mathcal{A}) \in \Sigma^* \times \text{NFA}_\Sigma$ where $L(\mathcal{A}) = E(\{u\})$.
- 4) If there exists $u \in L(\mathcal{C}) \cap V_1$ such that $E(\{u\}) \not\subseteq L(\mathcal{C})$, then the teacher picks such a u and returns a universal implication counterexample $(u, \mathcal{A}) \in \Sigma^* \times \text{NFA}_\Sigma$ where $L(\mathcal{A}) = E(\{u\})$.

If \mathcal{C} passes all four checks, the teacher replies “yes”. The order in which the teacher performs these checks is arbitrary.

It is easy to see that the language of a conjecture is indeed a winning set if the teacher replies “yes” (since it satisfies all conditions of Definition 1). The meaning of a positive counterexample is that any conjecture needs to accept it, but it was rejected. Similarly, a negative counterexample indicates that any conjecture has to reject it but it was accepted. An existential implication counterexample (u, \mathcal{A}) means that any conjecture accepting u has to accept at least one $v \in L(\mathcal{A})$, which was violated by the current conjecture. Finally, a universal implication counterexample (u, \mathcal{A}) means that any conjecture accepting u needs to accept all $v \in L(\mathcal{A})$. At this point, it is important to note that Definition 4 is sound (in particular, both types of implication counterexamples are well-defined due to Lemma 1 Part 2) and every counterexample is a finite object.

Let us illustrate this learning framework through an example.

Example 2: We revisit the setting of Example 1 for the case $k = 2$ and describe how the learner learns a winning set.

Suppose that the learner conjectures the DFA \mathcal{C}_0 with $L(\mathcal{C}_0) = \emptyset$. As \mathcal{C}_0 fails Check 1 (it passes all other checks), the teacher returns a positive counterexample, say $u = sll \in I$.

Next, suppose the learner conjectures the DFA \mathcal{C}_1 with $L(\mathcal{C}_1) = \{s^n \mid n \geq 2\}$, which passes all checks but Check 3 (as the players alternate but $L(\mathcal{C}_1)$ does not contain a vertex of the environment). The teacher replies with an existential implication counterexample, say (sll, \mathcal{A}) with $L(\mathcal{A}) = \{ell, elll\}$.

In the next round, let us assume that the learner conjectures the DFA \mathcal{C}_2 with $L(\mathcal{C}_2) = \{s^n \mid n \geq 2\} \cup \{e^m \mid m \geq 3\}$. This conjecture passes all checks (i.e., $L(\mathcal{C}_2)$ is a winning set), the teacher replies “yes”, and the learning ends. \dashv

It is important to note that classical learning frameworks for regular languages that involve learning from positive and negative data only, such as Gold’s passive learning [7] or Angluin’s active learning [8], are insufficient in our setting. If the learner provides a conjecture \mathcal{C} that violates Condition (3) or (4) of Definition 1, the teacher is stuck. For instance, if \mathcal{C} does not satisfy Condition (4), the teacher does not know whether to exclude u or to include $E(\{u\})$. Returning an implication counterexample, however, resolves this problem in that it communicates exactly why the conjecture is incorrect and, hence, allows the learner to make progress.¹

IV. A GENERIC TEACHER

We now present a generic teacher that, taking a rational safety game as input, answers queries according to Definition 4. For the remainder of this section, fix a rational safety game $\mathfrak{G}_\Sigma = (\mathfrak{A}_\Sigma, \mathcal{A}_F, \mathcal{A}_I)$ over the rational arena $\mathfrak{A}_\Sigma = (\mathcal{A}_{V_0}, \mathcal{A}_{V_1}, \mathcal{T}_E)$, and let \mathcal{C} be a DFA conjectured by the learner.

To answer a query, the teacher performs Checks 1 to 4 of Definition 4 as described below. If the conjecture passes all checks, the teacher returns “yes”; otherwise, he returns a corresponding counterexample, as described next.

Check 1 (initial vertices): The teacher computes an NFA \mathcal{B} with $L(\mathcal{B}) = L(\mathcal{A}_I) \setminus L(\mathcal{C})$. If $L(\mathcal{B}) \neq \emptyset$, he returns a positive counterexample $u \in L(\mathcal{B})$.

Check 2 (safe vertices): The teacher computes an NFA \mathcal{B} with $L(\mathcal{B}) = L(\mathcal{C}) \setminus L(\mathcal{A}_F)$. If $L(\mathcal{B}) \neq \emptyset$, he returns a negative counterexample $u \in L(\mathcal{B})$.

Check 3 (existential closure): To check existential closure, the teacher successively computes three NFAs:

- 1) An NFA \mathcal{B}_1 with $L(\mathcal{B}_1) = R(\mathcal{T}_E)^{-1}(L(\mathcal{C}))$; the language $L(\mathcal{B}_1)$ contains all vertices that have a successor in $L(\mathcal{C})$.
- 2) An NFA \mathcal{B}_2 with $L(\mathcal{B}_2) = L(\mathcal{A}_{V_0}) \setminus L(\mathcal{B}_1)$; the language $L(\mathcal{B}_2)$ contains all vertices of Player 0 that have no successor in $L(\mathcal{C})$.
- 3) An NFA \mathcal{B}_3 with $L(\mathcal{B}_3) = L(\mathcal{C}) \cap L(\mathcal{B}_2)$; the language $L(\mathcal{B}_3)$ contains all vertices of Player 0 that belong to $L(\mathcal{C})$ and have no successor in $L(\mathcal{C})$.

Every $u \in L(\mathcal{B}_3)$ is a witness that \mathcal{C} is not existentially closed. Hence, if $L(\mathcal{B}_3) \neq \emptyset$, the teacher picks an arbitrary $u \in L(\mathcal{B}_3)$

¹Garg et. al. [6] argue comprehensively why implications needed in a robust invariant learning framework. Their arguments also apply to our setting as one obtains a setting similar to Garg et. al.’s by considering a solitary game with Player 1 as the only player.

and returns the existential implication counterexample (u, \mathcal{A}) where $L(\mathcal{A}) = R(\mathcal{T}_E)(\{u\})$.

Check 4 (universal closure): To check universal closure, the teacher, again, computes three NFAs:

- 1) An NFA \mathcal{B}_1 with $L(\mathcal{B}_1) = (L(\mathcal{A}_{V_0}) \cup L(\mathcal{A}_{V_1})) \setminus L(\mathcal{C})$; the language $L(\mathcal{B}_1)$ contains all vertices not in $L(\mathcal{C})$.
- 2) An NFA \mathcal{B}_2 with $L(\mathcal{B}_2) = R(\mathcal{T}_E)^{-1}(L(\mathcal{B}_1))$; the language $L(\mathcal{B}_2)$ contains all vertices that have a successor not belonging to $L(\mathcal{C})$.
- 3) An NFA \mathcal{B}_3 with $L(\mathcal{B}_3) = L(\mathcal{A}_{V_1}) \cap L(\mathcal{C}) \cap L(\mathcal{B}_2)$; the language $L(\mathcal{B}_3)$ contains all vertices of Player 1 that are in $L(\mathcal{C})$ and have at least one successor not in $L(\mathcal{C})$.

Every $u \in L(\mathcal{B}_3)$ is a witness that \mathcal{C} is not universally closed. Hence, if $L(\mathcal{B}_3) \neq \emptyset$, the teacher picks an arbitrary $u \in L(\mathcal{B}_3)$ and returns the universal implication counterexample (u, \mathcal{A}) where $L(\mathcal{A}) = R(\mathcal{T}_E)(\{u\})$.

All checks can be performed using standard methods of automata theory, including product constructions, projections, determinizing automata, and emptiness checks (see Lemma 1).

V. A LEARNER FOR RATIONAL SAFETY GAMES

We design our learner with two key features: (1) the learner always conjectures a DFA consistent with the counterexamples received so far (we make this precise shortly), and (2) the learner always conjectures a minimal consistent DFA (i.e., a DFA with the least number of states among all DFAs that are consistent with the received counterexamples). The first design goal prevents the learner from making the same mistake twice, while the second design goal facilitates convergence of the overall learning (assuming that a winning set exists).

To meet these goals, our learner stores counterexamples in a data structure, which we call *sample*. Formally, a *sample* is a four-tuple $\mathcal{S} = (Pos, Neg, Ex, Uni)$ consisting of a finite set $Pos \subset \Sigma^*$ of positive words, a finite set $Neg \subset \Sigma^*$ of negative words, a finite set $Ex \subset \Sigma^* \times NFA_\Sigma$ of existential implications, and a finite set $Uni \subset \Sigma^* \times NFA_\Sigma$ of universal implications. We encourage the reader to think of a sample as a finite approximation of the safety game learned thus far.

In every iteration, our learner constructs a minimal DFA *consistent* with the current sample. A DFA \mathcal{B} is called *consistent* with a sample $\mathcal{S} = (Pos, Neg, Ex, Uni)$ if

- 1) $Pos \subseteq L(\mathcal{B})$;
- 2) $Neg \cap L(\mathcal{B}) = \emptyset$;
- 3) $u \in L(\mathcal{B})$ implies $L(\mathcal{B}) \cap L(\mathcal{A}) \neq \emptyset$ for each $(u, \mathcal{A}) \in Ex$;
- 4) $u \in L(\mathcal{B})$ implies $L(\mathcal{A}) \subseteq L(\mathcal{B})$ for each $(u, \mathcal{A}) \in Uni$.

Constructing a DFA that is consistent with a sample is possible only if the sample does not contain contradictory information. Contradictions can arise in two ways: first, Pos and Neg are not disjoint; second, the (alternating) transitive closure of the implications in Ex and Uni contains a pair (u, v) with $u \in Pos$ and $v \in Neg$. This observation justifies to introduce the notion of *contradiction-free* samples: a sample \mathcal{S} is called *contradiction-free* if a DFA that is consistent with \mathcal{S} exists. Since we assume that Player 0 wins from set I , a

Algorithm 1: A learner for rational safety games

```

1 Initialize an empty sample  $\mathcal{S} = (Pos, Neg, Ex, Uni)$  with
    $Pos = \emptyset$ ,  $Neg = \emptyset$ ,  $Ex = \emptyset$ , and  $Uni = \emptyset$ ;
2 repeat
3   Construct a minimal DFA  $\mathcal{A}_\mathcal{S}$  consistent with  $\mathcal{S}$ ;
4   Submit  $\mathcal{A}_\mathcal{S}$  to an equivalence query;
5   if the teacher returns a counterexample then
6     | Add the counterexample to  $\mathcal{S}$ ;
7   end
8 until the teacher replies “yes” to an equivalence query;
9 return  $\mathcal{A}_\mathcal{S}$ ;

```

winning set exists and the counterexamples returned by the teacher always form contradiction-free samples.²

After having constructed a minimal consistent DFA, the learner conjectures it to the teacher. If the teacher replies “yes”, the learning terminates. If the teacher returns a counterexample, on the other hand, the learner adds it to the appropriate set in \mathcal{S} and iterates. This procedure is sketched as Algorithm 1. Note that, by definition of the teacher, a conjecture is guaranteed to accept a winning set once the learning terminates.

It is left to describe how the learner actually constructs a minimal DFA that is consistent with the current sample. However, this task, known as *passive learning*, is computationally hard (i.e., the corresponding decision problem is NP-complete) already in the absence of implications [7]. Our strategy to approach this hurdle is to translate the original problem into a sequence of satisfiability problems of formulas in propositional Boolean logic and use highly optimized constraint solvers as a practically effective means to solve the resulting formulas (note that a translation into a logical formulation is a popular and effective strategy). More precisely, our learner creates and solves propositional Boolean formulas φ_n^S , for increasing values of $n \in \mathbb{N}$, $n \geq 1$, with the following two properties:

- 1) The formula φ_n^S is satisfiable if and only if there exists a DFA with n states that is consistent with \mathcal{S} .
- 2) A model \mathfrak{M} of φ_n^S (i.e., a satisfying assignment of the variables in φ_n^S) contains sufficient information to construct a DFA, denoted by $\mathcal{A}_\mathfrak{M}$, that has n states and is consistent with \mathcal{S} .

If φ_n^S is satisfiable, then Property 2 enables us to construct a consistent DFA from a model. However, if the formula is unsatisfiable, then the parameter n has been chosen too small and the learner increments it (e.g., by one or using a binary search). This procedure is summarized as Algorithm 2. We show its correctness shortly in Section V-B.

The key idea of the formula φ_n^S is to encode a DFA with n states by means of Boolean variables and to pose constraints on those variables. Our encoding relies on a simple observation: for every DFA there exists an isomorphic (hence, equivalent) DFA over the state set $Q = \{0, \dots, n-1\}$ with initial state

²In fact, checking for contradictions equips the learner with a means to detect that the game is won by Player 1. However, since determining the winner of a rational safety game is undecidable, any sample obtained during the learning might be contradiction-free despite the fact that Player 1 wins.

Algorithm 2: Computing a minimal consistent DFA.

Input: A contradiction-free sample \mathcal{S}

Output: A minimal DFA that is consistent with \mathcal{S}

```

1  $n \leftarrow 0$ ;
2 repeat
3    $n \leftarrow n + 1$ ;
4   Construct and solve  $\varphi_n^S$ ;
5 until  $\varphi_n^S$  is satisfiable, say with model  $\mathfrak{M}$ ;
6 return  $\mathcal{A}_{\mathfrak{M}}$ ;
```

$q_0 = 0$; moreover, given that Q and q_0 are fixed, any DFA with n states is uniquely determined by its transitions and final states. Therefore, we can fix the state set of the prospective DFA as $Q = \{0, \dots, n-1\}$ and the initial state as $q_0 = 0$; the alphabet Σ is announced by the teacher.

Our encoding of transitions and final states follows an idea from [14] (independently due to [15]). We introduce Boolean variables $d_{p,a,q}$ and f_q where $p, q \in Q$ and $a \in \Sigma$, which have the following meaning: setting $d_{p,a,q}$ to *true* means that the transition $\delta(p, a) = q$ exists in the prospective DFA, and setting f_q to *true* means that q is a final state.

To make sure that the variables $d_{p,a,q}$ encode a deterministic transition function, we impose two constraints:

$$\bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \bigwedge_{q, q' \in Q, q \neq q'} \neg d_{p,a,q} \vee \neg d_{p,a,q'} \quad (1)$$

$$\bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \bigvee_{q \in Q} d_{p,a,q} \quad (2)$$

Let φ_n^{DFA} be the conjunction of Formulas (1) and (2). Given a model \mathfrak{M} of φ_n^{DFA} (we assume a model to be a map from the variables of a formula to the set $\{\text{true}, \text{false}\}$), deriving the encoded DFA is straightforward, as shown next.

Definition 5: Let \mathfrak{M} be a model of φ_n^{DFA} . We define the DFA $\mathcal{A}_{\mathfrak{M}} = (Q, \Sigma, q_0, \delta, F)$ by (1) $\delta(p, a) = q$ for the unique $q \in Q$ with $\mathfrak{M}(d_{p,a,q}) = \text{true}$; and (2) $F = \{q \in Q \mid \mathfrak{M}(f_q) = \text{true}\}$. (Recall that we fixed $Q = \{0, \dots, n-1\}$ and $q_0 = 0$.)

To enforce that $\mathcal{A}_{\mathfrak{M}}$ is consistent with the given sample $\mathcal{S} = (\text{Pos}, \text{Neg}, \text{Ex}, \text{Uni})$, we impose further constraints, corresponding to the four requirements of consistent DFAs:

- a formula φ_n^{Pos} asserting $\text{Pos} \subseteq L(\mathcal{A}_{\mathfrak{M}})$;
- a formula φ_n^{Neg} asserting $\text{Neg} \cap L(\mathcal{A}_{\mathfrak{M}}) = \emptyset$;
- a formula φ_n^{Ex} asserting that $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}_{\mathfrak{M}}) \cap L(\mathcal{A}) \neq \emptyset$ for each $(u, A) \in \text{Ex}$; and
- a formula φ_n^{Uni} asserting that $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}_{\mathfrak{M}}) \subseteq L(\mathcal{A})$ for each $(u, A) \in \text{Uni}$.

Then, $\varphi_n^S := \varphi_n^{\text{DFA}} \wedge \varphi_n^{\text{Pos}} \wedge \varphi_n^{\text{Neg}} \wedge \varphi_n^{\text{Ex}} \wedge \varphi_n^{\text{Uni}}$. We here sketch formula φ_n^{Uni} and refer the reader to Appendix A for a detailed presentation of the remaining formulas. A description of φ_n^{Pos} and φ_n^{Neg} can also be found in [14].

A. The formula φ_n^{Uni}

We break the construction of φ_n^{Uni} down into smaller parts. Roughly speaking, we construct for each universal implication

$\iota = (u, \mathcal{A}) \in \text{Uni}$ a formula φ_n^{ι} that asserts $L(\mathcal{A}) \subseteq L(\mathcal{A}_{\mathfrak{M}})$ if $u \in L(\mathcal{A}_{\mathfrak{M}})$. The formulas φ_n^{Uni} is then the finite conjunction $\bigwedge_{\iota \in \text{Uni}} \varphi_n^{\iota}$. For the remainder, let us fix a universal implication $\iota \in \text{Uni}$, say $\iota = (u, \mathcal{A})$ with $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, q_0^{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$, and let $\text{Ante}(\text{Uni}) = \{u \mid (u, \mathcal{A}) \in \text{Uni}\}$ be the set of all words occurring as antecedent of a universal implication.

As a preparatory step, we introduce auxiliary Boolean variables that track the runs of $\mathcal{A}_{\mathfrak{M}}$ on words of $\text{Pref}(\text{Ante}(\text{Uni}))$ in order to detect when $\mathcal{A}_{\mathfrak{M}}$ accepts the antecedent of a universal implication. More precisely, we introduce variables $x_{u,q}$ where $u \in \text{Pref}(\text{Ante}(\text{Uni}))$ and $q \in Q$, which have the meaning that $x_{u,q}$ is set to *true* if $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{u} q$ (i.e., $\mathcal{A}_{\mathfrak{M}}$ reaches state q on reading u):

$$x_{\varepsilon, q_0} \quad (3)$$

$$\bigwedge_{u \in \text{Pref}(\text{Ante}(\text{Uni}))} \bigwedge_{q \neq q' \in Q} \neg x_{u,q} \vee \neg x_{u,q'} \quad (4)$$

$$\bigwedge_{ua \in \text{Pref}(\text{Ante}(\text{Uni}))} \bigwedge_{p, q \in Q} (x_{u,p} \wedge d_{p,a,q}) \rightarrow x_{ua,q} \quad (5)$$

Formula (3) asserts that x_{ε, q_0} is set to *true* since any run starts in the initial state q_0 . Formula (4) enforces that for every $u \in \text{Pref}(\text{Ante}(\text{Uni}))$ there exists at most one $q \in Q$ such that $x_{u,q}$ is set to *true* (in fact, the conjunction of Formulas (2)–(5) implies that there exists a unique such state). Finally, Formula (5) prescribes how the run of $\mathcal{A}_{\mathfrak{M}}$ on a word $u \in \text{Pref}(\text{Ante}(\text{Uni}))$ proceeds: if $\mathcal{A}_{\mathfrak{M}}$ reaches state p on reading u (i.e., $x_{u,p}$ is set to *true*) and there exists a transition from p to state q on reading the symbol $a \in \Sigma$ (i.e., $d_{p,a,q}$ is set to *true*), then $\mathcal{A}_{\mathfrak{M}}$ reaches state q on reading ua and $x_{ua,q}$ needs to be set to *true*.

We now define φ_n^{ι} . The formula ranges, in addition to $d_{p,a,q}$, f_q , and $x_{u,q}$, over Boolean variables $y_{q,q'}^{\iota}$ where $q \in Q$ and $q' \in Q_{\mathcal{A}}$, which track runs of \mathcal{A} and $\mathcal{A}_{\mathfrak{M}}$. Their precise meaning is the following: if there exists a word $u \in \Sigma^*$ with $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{u} q$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{u} q'$, then $y_{q,q'}^{\iota}$ is set to *true*:

$$y_{q_0, q_0^{\mathcal{A}}}^{\iota} \quad (6)$$

$$\bigwedge_{p, q \in Q} \bigwedge_{(p', a, q') \in \Delta_{\mathcal{A}}} (y_{p,p'}^{\iota} \wedge d_{p,a,q}) \rightarrow y_{q,q'}^{\iota} \quad (7)$$

Formula (6) enforces $y_{q_0, q_0^{\mathcal{A}}}^{\iota}$ to be set to *true* because $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{\varepsilon} q_0$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{\varepsilon} q_0^{\mathcal{A}}$. Formula (7) is similar to Formula (5) and describes how the runs of $\mathcal{A}_{\mathfrak{M}}$ and \mathcal{A} proceed: if there exists a word v such that $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} p$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{v} p'$ (i.e., $y_{p,p'}^{\iota}$ is set to *true*) and there are transitions $(p', a, q') \in \Delta_{\mathcal{A}}$ and $\delta(p, a) = q$ in $\mathcal{A}_{\mathfrak{M}}$, then $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{va} q$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{va} q'$, which requires $y_{q,q'}^{\iota}$ to be set to *true*.

Finally, the next constraint ensures that whenever $\mathcal{A}_{\mathfrak{M}}$ accepts u (i.e., the antecedent is *true*), then all words that lead to an accepting state in \mathcal{A} also lead to an accepting state in $\mathcal{A}_{\mathfrak{M}}$ (i.e., the consequent is *true*).

$$\left(\bigvee_{q \in Q} x_{u,q} \wedge f_q \right) \rightarrow \left(\bigwedge_{q \in Q} \bigwedge_{q' \in F_{\mathcal{A}}} y_{q,q'}^{\iota} \rightarrow f_q \right) \quad (8)$$

Let $\varphi_n^{\text{Ante}(\text{Uni})}$ be the conjunction of Formulas (3), (4), and (5) as well as φ_n^{ι} the conjunction of Formulas (6), (7), and (8). Then, φ_n^{Uni} is the (finite) conjunction $\varphi_n^{\text{Ante}(\text{Uni})} \wedge \bigwedge_{\iota \in \text{Uni}} \varphi_n^{\iota}$.

B. Correctness of the Learner

We now sketch a correctness proof of the learner—we refer the reader to Appendix B for a detailed proof. First, we state that φ_n^S has the desired properties.

Lemma 2: Let S be a sample, $n \geq 1$, and φ_n^S be as defined above. Then, the following statements hold: (1) If $\mathfrak{M} \models \varphi_n^S$, then $A_{\mathfrak{M}}$ is a DFA with n states that is consistent with S . (2) If there exists a DFA that has n states and is consistent with S , then φ_n^S is satisfiable.

Next, let us show the correctness of Algorithm 2.

Theorem 1: Given a contradiction free-sample S , Algorithm 2 returns a minimal DFA (in terms of the number of states) that is consistent with S . If a minimal consistent DFA has k states, then Algorithm 2 terminates after k iterations.

Proof: Given a sample S , suppose that there exists a DFA that has k states and is consistent with S . Then, φ_n^S is satisfiable for all $n \geq k$ (see Lemma 2). Moreover, if \mathfrak{M} is a model of φ_n^S , then $A_{\mathfrak{M}}$ is a DFA with n states that is consistent with S . Since Algorithm 2 increases the parameter n by one in every iteration (starting with $n = 1$), the algorithm eventually finds the smallest value for which φ_n^S is satisfiable (after k iterations) and, hence, a consistent DFA of minimal size. \square

Finally, we can prove the correctness of our learner.

Theorem 2: Given a teacher, Algorithm 1, equipped with Algorithm 2 to construct conjectures, terminates and returns a (minimal) DFA accepting a winning set if one exists.

Proof: Theorem 2 follows from three observations about the learner: (1) The learner never conjectures the same DFA twice (due to Theorem 1 and the fact that counterexamples are added to the sample). (2) The conjectures grow monotonically in size (due to minimality of conjectures) with increasing n , and (3) adding counterexamples to a sample does not rule out any solution (as every DFA accepting a winning set is consistent with any sample produced during the learning). Now, suppose a DFA accepting a winning set exists, say with k states. Due to Observations 1 and 2, the learner eventually conjectures a DFA with k states and, moreover, cannot conjecture a larger DFA (due to Observation 3 and the minimality of conjectures). Hence, the learner eventually conjectures a DFA with k states that accepts a winning set, and the learning terminates. \square

VI. EXPERIMENTS

In order to demonstrate the feasibility of our learning approach, we implemented a Java prototype using the BRICS automaton library [16] and Microsoft’s Z3 [17] constraint solver. The source code, including the games used in the experiments, is available at <http://preview.tinyurl.com/n7a7byj>.

In addition to the learner of Section V, we implemented a learner based on the popular RPNI algorithm [18], which is a polynomial time algorithm for learning DFAs from positive and negative words. For this learner, we modified the RPNI algorithm such that it constructs a consistent DFA from existential and universal implications in addition to positive and negative words (a detailed presentation can be found in Appendix C). In contrast to Algorithm 2, our modified version of RPNI cannot guarantee to find smallest consistent DFAs

and, hence, the resulting learner is a fast heuristic that is sound but in general not complete. Another limitation is that it can only handle implication counterexamples of the form (u, \mathcal{A}) where $L(\mathcal{A})$ is finite. We refer to the learner of Section V as *SAT learner* and the RPNI-based learner as *RPNI learner*.

Our experiments are on a slightly restricted type of games:

- 1) Edge relations are *automatic*. Automatic relations are defined by transducers that do not possess transitions of the form (a, ε) and (ε, b) but rather use a dedicated padding symbol to balance the length of their input-words.³
- 2) Each vertex of an arena has a finite (but not necessarily bounded) number of outgoing edges.

Restriction 1 simplifies the implementation of the teacher. Restriction 2 is due to the limitation of the RPNI learner.

We use two benchmark suits: the first suite serves to demonstrate the feasibility of our techniques for various examples, predominantly taken from the area of motion planning; the second suite serves to assess the performance of our techniques when confronted with games of increasing “complexity”. All games were given as finite automata, and we employed the teacher described in Section IV. We conducted all experiments on an Intel Core i7-4510U CPU (running Microsoft Windows 8.1) with a memory limit of 4 GiB and a runtime limit of 300 s.

A. Examples

We consider the following examples.

Diagonal game: A robot moves on an infinite, discrete two-dimensional grid world from one cell to an adjacent cell. Player 0 controls the robot’s vertical movement, whereas Player 1 controls the horizontal movement. Both players move the robot in alternation, and Player 0’s objective is to stay inside a margin of two cells around the diagonal.

Box game: A version of the diagonal game in which Player 0’s objective is to stay within a horizontal stripe of width three.

Solitary box game: A version of the box game in which Player 0 is the only player and has control over both the horizontal and the vertical movement.

Evasion game: Two robots move in alternation on an infinite, two-dimensional grid. Each robot is controlled by a player. Player 0’s objective is to avoid collision with Player 1’s robot.

Follow game: A version of the evasion game in which Player 0’s objective is to keep his robot within a distance of two cells (in the Manhattan distance) from Player 1’s robot.

Program-repair game: A finitely-branching version of the program-repair game described by Beyene et al. [10].

Table I lists the overall time taken by each of the two learners to learn a winning set (including the time taken by the teacher) as well as further statistics of the learning process. The second column $|\mathcal{G}|$ corresponds to sum of states of all automata constituting a game (*size of the game*), which serves as measure for the complexity of a game. The remaining columns list the number of iterations, the number of states of the learned DFA, and the cardinality of each set of the final sample.

³Automatic relations constitute a proper subset of rational relations, but are still expressive enough to encode computations of Turing machines.

TABLE I
RESULTS OF THE FIRST BENCHMARK SUITE

Game	$ \mathcal{G} $	SAT learner							RPNI learner						
		Time in s	Iter.	Size	$ Pos $	$ Neg $	$ Ex $	$ Uni $	Time in s	Iter.	Size	$ Pos $	$ Neg $	$ Ex $	$ Uni $
Diagonal	29	1.352	62	4	1	55	2	3	1.000	77	6	1	54	10	11
Box	25	0.516	32	4	1	30	0	0	0.188	15	5	1	10	1	2
Solitary Box	22	4.289	81	6	1	77	2	0	0.156	16	6	1	13	1	0
Follow	53	165.670	294	7	2	269	10	12	timeout (> 300 s)						
Evasion	56	140.888	255	7	2	232	11	9	2.316	142	12	1	115	14	11
Program-repair	41	1.948	62	3	2	55	4	0	0.438	31	4	1	20	9	0

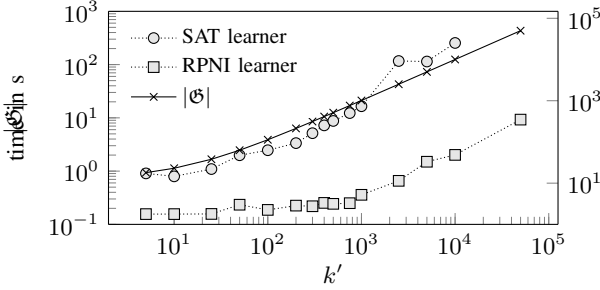


Fig. 2. Results of the scalability benchmark.

As Table I shows, the SAT learner computed the winning sets for all games, whereas the RPNI learner computed the winning sets for all but the Follow game. Since the RPNI learner does not compute minimal consistent DFAs, we expected that it is on average faster than the SAT learner, which turned out to be the case. However, the RPNI learner fails to terminate within the time limit on the Follow game, and the large number of iterations seem to indicate that the learner in fact diverges.

Finally, it is important to note that the teacher replied implication counterexamples in all but one experiment. This observation highlights that classical learning algorithms, which learn from positive and negative words only, are insufficient to learn winning sets (since the learning would be stuck at that point) and one has to move to a richer learning framework.

B. Scalability Benchmarks

To assess the scalability of our technique when confronted with inputs of increasing size, we modified the game of Example 1 such that the safe region is now determined by two parameters, namely k and k' , and contains all positions in the interval $[k, k']$ (we assume $k < k'$ and fix $k = 1$). In this new setting, the number of states of the automaton \mathcal{A}_F increases when k' increases as the automaton needs to count in unary to check the position of the robot.

Figure 2 depicts the overall time taken to learn a winning set, depending on the parameter k' . To put the runtimes into perspective, it also shows the size of the games.

On the scalability benchmark suite, the RPNI learner was about one order of magnitude faster than the SAT learner and can compute a winning set for games up to a combined size of 50 000. The SAT learner, on the other hand, computed a winning set for games up to a combined size of 10 000 but did not terminate for game with $k' = 50\,000$. While a thorough assessment remains as part of future work, our results promise

applicability to practically interesting problem instances.

VII. CONCLUSION

We developed an automata learning method to construct finite-state reactive controllers for systems whose interactions with their environment are modeled by infinite-state games. We focused on the practically interesting family of safety games, utilized a symbolic representation of the underlying game, developed specific implementations of the learner and the teacher, and demonstrated the feasibility of the method on a set of problems motivated by robotic motion planning.

REFERENCES

- [1] R. McNaughton, “Infinite games played on finite graphs,” *Ann. Pure Appl. Logic*, vol. 65, no. 2, pp. 149–184, 1993.
- [2] S. Itzhaky, S. Gulwani, N. Immerman, and M. Sagiv, “A simple inductive synthesis methodology and its applications,” in *OOPSLA 2010*. ACM, 2010, pp. 36–46.
- [3] R. Ehlers, V. Raman, and C. Finucane, “Slugs GR(1) synthesizer,” 2014, available at <https://github.com/LTLMoP/slugs/>.
- [4] A. Bohy, V. Bruyère, E. Filiot, N. Jin, and J.-F. Raskin, “Acacia+, a tool for ltl synthesis,” in *CAV*, 2012, pp. 652–657.
- [5] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili, “Regular model checking,” in *CAV 2000*, ser. LNCS, vol. 1855. Springer, 2000, pp. 403–418.
- [6] P. Garg, C. Löding, P. Madhusudan, and D. Neider, “ICE: A robust framework for learning invariants,” in *CAV 2014*, ser. LNCS, vol. 8559. Springer, 2014, pp. 69–87.
- [7] E. M. Gold, “Complexity of automaton identification from given data,” *Information and Control*, vol. 37, no. 3, pp. 302–320, 1978.
- [8] D. Angluin, “Learning regular sets from queries and counterexamples,” *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.
- [9] O. Kupferman, N. Piterman, and M. Y. Vardi, “An automata-theoretic approach to infinite-state systems,” in *Time for Verification, Essays in Memory of Amir Pnueli*, ser. LNCS, vol. 6200. Springer, 2010, pp. 202–259.
- [10] T. A. Beyene, S. Chaudhuri, C. Popeea, and A. Rybalchenko, “A constraint-based approach to solving games on infinite graphs,” in *POPL 2014*. ACM, 2014, pp. 221–234.
- [11] D. Neider, “Reachability games on automatic graphs,” in *CIAA 2010*, ser. LNCS, vol. 6482. Springer, 2010, pp. 222–230.
- [12] —, “Small strategies for safety games,” in *ATVA 2011*, ser. LNCS, vol. 6996. Springer, 2011, pp. 306–320.
- [13] A. Blumensath and E. Grädel, “Finite presentations of infinite structures: Automata and interpretations,” *Theory Comput. Syst.*, vol. 37, no. 6, pp. 641–674, 2004.
- [14] D. Neider and N. Jansen, “Regular model checking using solver technologies and automata learning,” in *NFM 2013*, ser. LNCS, vol. 7871. Springer, 2013, pp. 16–31.
- [15] M. Heule and S. Verwer, “Exact DFA identification using SAT solvers,” in *ICGI 2010*, ser. LNCS, vol. 6339. Springer, 2010, pp. 66–79.
- [16] A. Möller, “dk.brics.automaton – finite-state automata and regular expressions for Java,” 2010, <http://www.brics.dk/automaton/>.
- [17] L. M. de Moura and N. Bjørner, “Z3: an efficient SMT solver,” in *TACAS 2008*, ser. LNCS, vol. 4963. Springer, 2008, pp. 337–340.

- [18] J. Oncina and P. Garcia, "Inferring regular languages in polynomial update time," in *Pattern Recognition & Image Analysis*, 1992, pp. 49–61.

APPENDIX A
CONSTRUCTING CONSISTENT DFAS
USING CONSTRAINT SOLVERS

The key building block of our learner is an algorithm that, given a sample \mathcal{S} , produces a smallest DFA that is consistent with \mathcal{S} . Recall that the learner translates this problem into a series of satisfiability problem of propositional Boolean formulas $\varphi_n^{\mathcal{S}}$ and uses a constraint solver to check their satisfiability.

In the following, we describe in detail how the formula $\varphi_n^{\mathcal{S}}$ is constructed. For the sake of a self-contained presentation, we repeat parts of Section V; as a beneficial side-effect, this repetition allows us to provide further explanations of the formulas presented in Section V. Moreover, to facilitate a more concise and accessible description, we define $\varphi_n^{\mathcal{S}}$ slightly different. In particular, we introduce a formula φ_n^W , which tracks the run of $\mathcal{A}_{\mathfrak{M}}$ on words occurring in the sample (in Pos , Neg , and as antecedent of an implication). In contrast to Section V (where we defined the formula φ_n^{Uni} to track the run of $\mathcal{A}_{\mathfrak{M}}$ on the set $Ante(Uni)$) this approach results in more concise and easier to understand formulas since (a prefix of) a word can occur more than once in a sample. As a consequence, however, the formula φ_n^{Uni} has to be changed in comparison to Section V.

Recapping the main ideas and encoding of states and transitions

The key idea of the formula $\varphi_n^{\mathcal{S}}$ is to encode a DFA with n states by means of Boolean variables and to pose constraints on those variables in order to obtain a DFA that is consistent with the given sample. Our encoding relies on a simple observation: if we fix the alphabet, the set of states and the initial state, then any DFA with n states is uniquely determined (up to isomorphism) by its transitions and final states. Hence, we can without loss of generality fix the state set of the prospective DFA to be $Q = \{0, \dots, n-1\}$ and the initial state to be $q_0 = 0$; the alphabet Σ is determined by the given game.

To encode the transitions and the final states, we introduce Boolean variables $d_{p,a,q}$ and f_q where $p, q \in Q$ and $a \in \Sigma$, which have the following meaning: assigning *true* to $d_{p,a,q}$ means that the transition $\delta(p, a) = q$ exists in the prospective DFA, and assigning *true* to f_q means that q is a final state.

To make sure that the variables $d_{p,a,q}$ indeed encode a deterministic transition function, we impose the following constraints.

$$\bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \bigwedge_{q, q' \in Q, q \neq q'} \neg d_{p,a,q} \vee \neg d_{p,a,q'} \quad (9)$$

$$\bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \bigvee_{q \in Q} d_{p,a,q} \quad (10)$$

Formula (9) and (10) are the same as Formula (1) and (2) of Section V, respectively: Formula (9) enforces that $d_{p,a,q}$ encode a deterministic function, while Formula (10) asserts that the function is total.

Let $\varphi_n^{\text{DFA}}(\bar{d}, \bar{f})$ be the conjunction of Formulas (9) and (10) where \bar{d} denotes the list of variables $d_{p,a,q}$ and \bar{f} denotes the

list of variables f_q for $p, q \in Q$ and $a \in \Sigma$. Given a model \mathfrak{M} of φ_n^{DFA} , deriving the encoded DFA is straightforward, as shown next.

Definition 6: Let $\mathfrak{M} \models \varphi_n^{\text{DFA}}(\bar{d}, \bar{f})$. We define the DFA $\mathcal{A}_{\mathfrak{M}} = (Q, \Sigma, q_0, \delta, F)$ by

- $\delta(p, a) = q$ for the unique $q \in Q$ with $\mathfrak{M}(d_{p,a,q}) = \text{true}$; and
- $F = \{q \in Q \mid \mathfrak{M}(f_q) = \text{true}\}$.

(Recall that we fixed $Q = \{0, \dots, n-1\}$ and $q_0 = 0$.)

To produce a DFA that is consistent with a sample, we add further constraints:

- a formula φ_n^{Pos} asserting $Pos \subseteq L(\mathcal{A}_{\mathfrak{M}})$;
- a formula φ_n^{Neg} asserting $Neg \cap L(\mathcal{A}_{\mathfrak{M}}) = \emptyset$;
- a formula φ_n^{Ex} asserting for each $(u, A) \in Ex$ that $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}_{\mathfrak{M}}) \cap L(A) \neq \emptyset$; and
- a formula φ_n^{Uni} asserting for each $(u, A) \in Ex$ that $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}_{\mathfrak{M}}) \subseteq L(A)$.

Moreover, we add an auxiliary formula φ_n^W , which we discuss shortly. Then,

$$\varphi_n^{\mathcal{S}} := \varphi_n^{\text{DFA}} \wedge \varphi_n^W \wedge \varphi_n^{Pos} \wedge \varphi_n^{Neg} \wedge \varphi_n^{Ex} \wedge \varphi_n^{Uni}$$

is the desired formula.

The pivotal idea of these formulas is to impose constraints on the variables $d_{p,a,q}$ and f_q , which, in turn, determine the DFA $\mathcal{A}_{\mathfrak{M}}$. Having this in mind, it is easier to describe the effects of these constraints by referring to \mathfrak{M} rather than to the variables themselves. However, we thereby implicitly assume that the formula is satisfiable and that the valuation \mathfrak{M} is a model.

—THE FORMULA φ_n^W —

To ensure that the prospective automaton $\mathcal{A}_{\mathfrak{M}}$ is consistent with the given sample, we need a mechanism to determine whether $\mathcal{A}_{\mathfrak{M}}$ accepts or rejects the words occurring in the sample. The idea is to track the run of $\mathcal{A}_{\mathfrak{M}}$ on all prefixes of the set

$$W = Pos \cup Neg \cup Ante(Ex) \cup Ante(Uni),$$

which contains all positive and negative words as well as all words that occur as antecedent of an existential or universal implication. The idea is to introduce auxiliary Boolean variables $x_{u,q}$ where $u \in Pref(W)$ and $q \in Q$; the intended meaning of these variables is that if the prospective DFA $\mathcal{A}_{\mathfrak{M}}$ reaches state q on reading the word u , then $x_{u,q}$ is set to *true*. The following constraints enforce this.

$$x_{\varepsilon, q_0} \quad (11)$$

$$\bigwedge_{u \in Pref(W)} \bigwedge_{q \neq q' \in Q} \neg x_{u,q} \vee \neg x_{u,q'} \quad (12)$$

$$\bigwedge_{ua \in Pref(W)} \bigwedge_{p, q \in Q} (x_{u,p} \wedge d_{p,a,q}) \rightarrow x_{ua,q} \quad (13)$$

Since any run starts in the initial state q_0 , Formula (11) asserts that x_{ε, q_0} is set to *true*. Formula (12) enforces that for every $u \in Pref(W)$ there exists at most one $q \in Q$ such that

$x_{u,q}$ is set to *true* (in fact, the conjunction of Formulas (10)–(13) implies that there exists a unique such state). Finally, Formula (13) prescribes how the run of $\mathcal{A}_{\mathfrak{M}}$ on a word $u \in \text{Pref}(W)$ proceeds: if $\mathcal{A}_{\mathfrak{M}}$ reaches state p on reading u (i.e., $x_{u,p}$ is set to *true*) and there exists a transition from p to state q on reading the symbol $a \in \Sigma$ (i.e., $d_{p,a,q}$ is set to *true*), then $\mathcal{A}_{\mathfrak{M}}$ reaches state q on reading ua and x_{ua} is set to *true*.

Let $\varphi_n^W(\bar{d}, \bar{f}, \bar{x})$ be the conjunction of Formulas (11), (12), and (13) where \bar{d} and \bar{f} are as above and \bar{x} is the list of variables $x_{u,q}$ for $u \in \text{Pref}(W)$ and $q \in Q$. Then a straightforward induction proves the following lemma (see, e.g., Neider and Jansen [14]).

Lemma 3: Let $n \geq 1$, \mathfrak{M} a model of

$$\varphi_n^{\text{DFA}}(\bar{d}, \bar{f}) \wedge \varphi_n^W(\bar{d}, \bar{f}, \bar{x}),$$

and $\mathcal{A}_{\mathfrak{M}}$ the DFA defined according to Definition 6. Then, $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{u} q$ implies $\mathfrak{M}(x_{u,q}) = \text{true}$ for all $u \in \text{Pref}(W)$.

—————THE FORMULAS φ_n^{Pos} AND φ_n^{Neg} —————

Having introduced the formula φ_n^W , it is straightforward to enforce a correct behavior of $\mathcal{A}_{\mathfrak{M}}$ on *Pos* and *Neg*. To assert that $\mathcal{A}_{\mathfrak{M}}$ accepts all words in *Pos*, we impose the constraint

$$\bigwedge_{u \in \text{Pos}} \bigwedge_{q \in Q} x_{u,q} \rightarrow f_q, \quad (14)$$

which ensures that state q is a final state if $\mathcal{A}_{\mathfrak{M}}$ reaches q on reading a word $u \in \text{Pos}$. Similarly, the constraint

$$\bigwedge_{u \in \text{Neg}} \bigwedge_{q \in Q} x_{u,q} \rightarrow \neg f_q \quad (15)$$

makes sure that state q is not a final state if $\mathcal{A}_{\mathfrak{M}}$ reaches q on reading a word $u \in \text{Neg}$, hence, asserting that all words of *Neg* are rejected.

Let $\varphi_n^{\text{Pos}}(\bar{d}, \bar{f}, \bar{x})$ denote Formula (14) and $\varphi_n^{\text{Neg}}(\bar{d}, \bar{f}, \bar{x})$ denote Formula (15) where \bar{d} , \bar{f} , and \bar{x} are as above. Then, we obtain the following results.

Lemma 4: Let $\mathcal{S} = (\text{Pos}, \text{Neg}, \text{Ex}, \text{Uni})$ be a sample, $n \geq 1$, and

$$\psi_n^{\text{Pos}}(\bar{d}, \bar{f}, \bar{x}) := \varphi_n^{\text{DFA}}(\bar{d}, \bar{f}) \wedge \varphi_n^W(\bar{d}, \bar{f}, \bar{x}) \wedge \varphi_n^{\text{Pos}}(\bar{d}, \bar{f}, \bar{x}).$$

Then, the following statements hold:

- 1) If $\mathfrak{M} \models \psi_n^{\text{Pos}}$, then $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that satisfies $\text{Pos} \subseteq L(\mathcal{A}_{\mathfrak{M}})$.
- 2) If a DFA \mathcal{B} with n states exists that satisfies $\text{Pos} \subseteq L(\mathcal{B})$, then ψ_n^{Pos} is satisfiable.

Lemma 5: Let $\mathcal{S} = (\text{Pos}, \text{Neg}, \text{Ex}, \text{Uni})$ be a sample, $n \geq 1$, and

$$\psi_n^{\text{Neg}}(\bar{d}, \bar{f}, \bar{x}) := \varphi_n^{\text{DFA}}(\bar{d}, \bar{f}) \wedge \varphi_n^W(\bar{d}, \bar{f}, \bar{x}) \wedge \varphi_n^{\text{Neg}}(\bar{d}, \bar{f}, \bar{x}).$$

Then, the following statements hold:

- 1) If $\mathfrak{M} \models \psi_n^{\text{Neg}}$, then $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that satisfies $\text{Neg} \cap L(\mathcal{A}_{\mathfrak{M}}) = \emptyset$.
- 2) If a DFA \mathcal{B} with n states exists that satisfies $\text{Neg} \cap L(\mathcal{B}) = \emptyset$, then ψ_n^{Neg} is satisfiable.

Let us now prove Lemma 4. The proof of Lemma 5 is analogous.

Proof of Lemma 4: To prove the Statement 1, assume $\mathfrak{M} \models \psi_n^{\text{Pos}}$ and let $\mathcal{A}_{\mathfrak{M}}$ be the DFA constructed according to Definition 6. Furthermore, pick an arbitrary $u \in \text{Pos}$. Then, Lemma 3 implies that if $\mathcal{A}_{\mathfrak{M}}$ reaches state q on reading u , then $\mathfrak{M}(x_{u,q}) = \text{true}$. Additionally, Formula (14) asserts that q is a final state and, therefore, $\mathcal{A}_{\mathfrak{M}}$ accepts u by Definition 6. Since this is true for all $u \in \text{Pos}$, we obtain $\text{Pos} \subseteq L(\mathcal{A}_{\mathfrak{M}})$.

To prove the second statement, let $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, q_0^{\mathcal{B}}, \delta_{\mathcal{B}}, F_{\mathcal{B}})$ be a DFA with n states that satisfies $\text{Pos} \subseteq L(\mathcal{B})$. The key idea is to translate \mathcal{B} into a valuation \mathfrak{V} that satisfies ψ_n^{Pos} . To simplify this translation a bit, we assume without loss of generality that the sets of states of \mathcal{B} and $\mathcal{A}_{\mathfrak{M}}$ coincide (i.e., $Q_{\mathcal{B}} = Q$); one can easily achieve this by renaming states. The definition of \mathfrak{V} is as follows:

- For each $p, q \in Q_{\mathcal{B}}$ and $a \in \Sigma$, we set $d_{p,a,q}$ to *true* if and only if $\delta_{\mathcal{B}}(p, a) = q$.
- For each $q \in Q_{\mathcal{B}}$, we set f_q to *true* if and only if $q \in F_{\mathcal{B}}$.
- For each $u \in W$, we set $x_{u,q}$ to *true* if and only if $\mathcal{B}: q_0^{\mathcal{B}} \xrightarrow{u} q$.

It is not hard to verify that \mathfrak{V} indeed satisfies ψ_n^{Pos} since $\mathfrak{V}(x_{u,q})$ is defined according to the runs of \mathcal{B} on the inputs $u \in W$. \square

—————THE FORMULA φ_n^{Uni} —————

The formula φ_n^{Uni} needs to enforce that $L(\mathcal{A}_{\mathfrak{M}})$ respects all universal implications in *Uni*. (Recall that the learner stores universal and existential implication as a pair (u, \mathcal{A}) where $u \in \Sigma^*$ is a word and \mathcal{A} is an NFA over Σ .) To achieve this, we construct for each universal implication $\iota = (u, \mathcal{A}) \in \text{Uni}$ a formula φ_n^{ι} that asserts $L(\mathcal{A}) \subseteq L(\mathcal{A}_{\mathfrak{M}})$ if $u \in L(\mathcal{A}_{\mathfrak{M}})$. The formulas φ_n^{Uni} is then the (finite) conjunction $\bigwedge_{\iota \in \text{Uni}} \varphi_n^{\iota}$.

Given a universal implication $\iota \in \text{Uni}$, say $\iota = (u, \mathcal{A})$ with $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, q_0^{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$, the key idea of the formula φ_n^{ι} is to track the runs of $\mathcal{A}_{\mathfrak{M}}$ and \mathcal{A} in parallel. To this end, we introduce new auxiliary variables $y_{q,q'}^{\iota}$ where $q \in Q$ and $q' \in Q_{\mathcal{A}}$, which have the following meaning: the variable $y_{q,q'}^{\iota}$ is set to *true* if there exists a word $v \in \Sigma^*$ such that $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} q$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{v} q'$. The following constraints assert this.

$$y_{q_0, q_0^{\mathcal{A}}}^{\iota} \quad (16)$$

$$\bigwedge_{p, q \in Q} \bigwedge_{(p', a, q') \in \Delta_{\mathcal{A}}} (y_{p, p'}^{\iota} \wedge d_{p, a, q}) \rightarrow y_{q, q'}^{\iota} \quad (17)$$

Formula (16) enforces $y_{q_0, q_0^{\mathcal{A}}}^{\iota}$ to be set to *true* because $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{\varepsilon} q_0$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{\varepsilon} q_0^{\mathcal{A}}$. Formula (17) is similar to Formula (13) and describes how the runs of $\mathcal{A}_{\mathfrak{M}}$ and \mathcal{A} proceed: if there exists a word v such that $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} p$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{v} p'$ (i.e., $y_{p, p'}^{\iota}$ is set to *true*) and there are transitions $(p', a, q') \in \Delta_{\mathcal{A}}$ and $\delta(p, a) = q$ in $\mathcal{A}_{\mathfrak{M}}$, then $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{va} q$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{va} q'$, which requires that $y_{q, q'}^{\iota}$ has to be set to *true* as well.

Note that the variables $y_{q, q'}^{\iota}$ do not track runs exactly: it is possible that a variable $y_{q, q'}^{\iota}$ is set to *true* even without the

existence of a word $v \in \Sigma^*$ that induces the runs $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} q$ and $\mathcal{A}: q_0^A \xrightarrow{v} q'$. This inaccuracy, however, is sufficient to obtain the desired result.

In order to express that $\mathcal{A}_{\mathfrak{M}}$ indeed respects the universal implication ι , we add the implication

$$\left(\bigvee_{q \in Q} x_{u,q} \wedge f_q \right) \rightarrow \left(\bigwedge_{q \in Q} \bigwedge_{q' \in F_A} y_{q,q'}^t \rightarrow f_q \right). \quad (18)$$

This formula ensures that whenever $\mathcal{A}_{\mathfrak{M}}$ accepts u (i.e., the antecedent is *true*), then all words that lead to an accepting state in \mathcal{A} also lead to an accepting state in $\mathcal{A}_{\mathfrak{M}}$ (i.e., the consequent is *true*).

Let $\varphi_n^t(\bar{d}, \bar{f}, \bar{x}, \bar{y}^t)$ be the conjunction of Formulas (16), (17), and (18) where \bar{d}, \bar{f} , as well as \bar{x} are as above and \bar{y}^t is the list of all $y_{q,q'}^t$ for $q \in Q$ and $q' \in Q_A$. Additionally, let φ_n^{Ex} be the conjunction

$$\varphi_n^{Uni}(\bar{d}, \bar{f}, \bar{x}, \bar{y}) := \bigwedge_{\iota \in Uni} \varphi_n^t(\bar{d}, \bar{f}, \bar{x}, \bar{y}^t),$$

where \bar{y} denotes the list of all variables occurring in \bar{y}^t for each $\iota \in Uni$. Then, the following holds.

Lemma 6: Let $\mathcal{S} = (Pos, Neg, Ex, Uni)$ be a sample, $n \geq 1$, and

$$\psi_n^{Uni}(\bar{d}, \bar{f}, \bar{x}, \bar{y}) := \varphi_n^{DFA}(\bar{d}, \bar{f}) \wedge \varphi_n^W(\bar{d}, \bar{f}, \bar{x}) \wedge \varphi_n^{Uni}(\bar{d}, \bar{f}, \bar{x}, \bar{y}).$$

Then, the following statements hold:

- 1) If $\mathcal{M} \models \psi_n^{Uni}$, then $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that satisfies for all $(u, \mathcal{A}) \in Uni$ that $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}) \subseteq L(\mathcal{A}_{\mathfrak{M}})$.
- 2) If a DFA with n states exists that satisfies for all $(u, \mathcal{A}) \in Uni$ that $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}) \subseteq L(\mathcal{A}_{\mathfrak{M}})$, then ψ_n^{Uni} is satisfiable.

Proof: We split the proof in two parts: we first show Statement 1 and subsequently Statement 2.

To prove Statement 1, we show that for an universal implication $\iota = (u, \mathcal{A}) \in Uni$, a model \mathfrak{M} of the formula

$$\psi_n^t(\bar{d}, \bar{f}, \bar{x}, \bar{y}^t) := \varphi_n^{DFA}(\bar{d}, \bar{f}) \wedge \varphi_n^W(\bar{d}, \bar{f}, \bar{x}) \wedge \varphi_n^t(\bar{d}, \bar{f}, \bar{x}, \bar{y}^t)$$

results in an automaton $\mathcal{A}_{\mathfrak{M}}$ that respects ι (i.e., $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}) \subseteq L(\mathcal{A}_{\mathfrak{M}})$). The claim of Statement 1 then follows immediately because φ_n^{Uni} is the conjunction of the individual formulas φ_n^t . In the following, fix an universal implication $\iota = (u, \mathcal{A}) \in Uni$, assume $\mathfrak{M} \models \psi_n^t$, and let $\mathcal{A}_{\mathfrak{M}}$ be the DFA constructed according to Definition 6.

Given an universal implication $\iota = (u, \mathcal{A})$, say with $\mathcal{A} = (Q_A, \Sigma, q_0, \Delta_A, F_A)$, we first show by induction over the length of inputs $v \in \Sigma^*$ that the variables $y_{q,q'}^t$ have indeed the desired meaning (i.e., $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} q$ and $\mathcal{A}: q_0^A \xrightarrow{v} q'$ imply $\mathfrak{M}(y_{q,q'}^t) = \text{true}$).

Base case ($v = \varepsilon$) Both $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{\varepsilon} q_0$ and $\mathcal{A}: q_0^A \xrightarrow{\varepsilon} q_0^A$ hold by definition of runs. Moreover, Formula (16) enforces $\mathfrak{M}(y_{q_0, q_0^A}^t) = \text{true}$. Thus, the claim holds.

Induction step ($v = v'a$) Assume $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v'} p \xrightarrow{a} q$ and $\mathcal{A}: q_0^A \xrightarrow{v'} p' \xrightarrow{a} q'$. Thus, there exists transitions $(p', a, q') \in \Delta_A$ and $\delta(p, a) = q$; the latter means $\mathfrak{M}(d_{p,a,q}) = \text{true}$ by Definition 6. Moreover, applying the induction hypothesis yields $\mathfrak{M}(y_{p,p'}^t) = \text{true}$. In this situation, Formula (17) enforces $\mathfrak{M}(y_{q,q'}^t) = \text{true}$, which proves the claim.

Having established the meaning of the variables $y_{q,q'}^t$, it is now straightforward to prove that $\mathcal{A}_{\mathfrak{M}}$ satisfies $L(\mathcal{A}) \subseteq L(\mathcal{A}_{\mathfrak{M}})$ if $u \in L(\mathcal{A}_{\mathfrak{M}})$. If $\mathcal{A}_{\mathfrak{M}}$ accepts u , say $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{u} q$ with $q \in F$, then we know that $\mathfrak{M}(x_{u,q}) = \text{true}$ (by Lemma 3) and $\mathfrak{M}(f_q) = \text{true}$ (by Definition 6). In this situation, the antecedent of Formula (18) is satisfied. Thus, its consequent is necessarily satisfied as well because \mathfrak{M} is a satisfying assignment of ψ_n^{Uni} . This, in turn, ensures that whenever \mathcal{A} accepts a word $v \in \Sigma^*$, say $\mathcal{A}: q_0^A \xrightarrow{v} q'$ with $q' \in F_A$, then the run $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} q$ is also accepting: the induction above shows that $\mathfrak{M}(y_{q,q'}^t) = \text{true}$ and, since the consequent of Formula (18) ensures that $\mathfrak{M}(y_{q,q'}^t) = \text{true}$ implies $\mathfrak{M}(f_q) = \text{true}$ for all $q \in Q$ and $q' \in F_A$, also $\mathfrak{M}(f_q) = \text{true}$ holds. Hence, $L(\mathcal{A}) \subseteq L(\mathcal{A}_{\mathfrak{M}})$ because v was chosen arbitrarily. Since these arguments are true for all $\iota \in Uni$, the DFA $\mathcal{A}_{\mathfrak{M}}$ respects all implications in Uni .

To prove Statement 2, suppose that $\mathcal{B} = (Q_B, \Sigma, q_0^B, \delta_B, F_B)$ is a DFA with n states that respects all universal implications in Uni . Similar to the proof of Lemma 3, we translate this DFA into a assignment \mathfrak{V} that satisfies ψ_n^{Uni} . For the sake of this translation, we assume without loss of generality that the state sets of \mathcal{B} and $\mathcal{A}_{\mathfrak{M}}$ coincide (i.e., $Q_B = Q$).

The translation is as follows:

- For each $p, q \in Q_B$ and $a \in \Sigma$, we set $\mathfrak{V}(d_{p,a,q}) = \text{true}$ is and only if $\delta_B(p, a) = q$.
- For each $q \in Q_B$, we set $\mathfrak{V}(f_q) = \text{true}$ if and only if $q \in F_B$.
- For each $u \in W$ and $q \in Q_B$, we set $\mathfrak{V}(x_{u,q}) = \text{true}$ if and only if $\mathcal{B}: q_0^B \xrightarrow{u} q$.
- For each universal implication $\iota = (u, \mathcal{A}) \in Uni$ with $\mathcal{A} = (Q_A, \Sigma, q_0^A, \Delta_A, F_A)$, $q \in Q_B$, and $q' \in Q_A$, we set $\mathfrak{V}(y_{q,q'}^t) = \text{true}$ if a $v \in \Sigma^*$ exists such that $\mathcal{B}: q_0 \xrightarrow{v} q$ and $\mathcal{A}: q_0^A \xrightarrow{v} q'$.

It is not hard to verify that \mathfrak{V} satisfies $\varphi_n^{DFA} \wedge \varphi_n^W$. To show that it also satisfies φ_n^{Uni} , fix a universal implication $\iota = (u, \mathcal{A})$, say with $\mathcal{A} = (Q_A, \Sigma, q_0^A, \Delta_A, F_A)$. We first observe that \mathfrak{V} satisfies Formulas (16) and (17) since the variables $y_{q,q'}^t$ track the runs of both automata on inputs $v \in \Sigma^*$. Second, if $u \notin L(\mathcal{B})$, then \mathfrak{V} does not satisfy the antecedent of Formula (18) and, hence, satisfies Formula (18). If $u \in L(\mathcal{B})$, on the other hand, consider the runs $\mathcal{B}: q_0^B \xrightarrow{u} q$ and $\mathcal{A}: q_0^A \xrightarrow{u} q'$ on some input $v \in \Sigma^*$. Then, $\mathfrak{V}(y_{q,q'}^t) = \text{true}$ by definition of \mathfrak{V} . Moreover, if \mathcal{A} accepts v (i.e., $q' \in F_A$), then \mathcal{B} accepts v as well (i.e., $q \in F_B$) because \mathcal{B} respects all implications in Uni . Hence, $\mathfrak{V}(f_q) = \text{true}$ by definition of \mathfrak{V} . Thus, the valuation \mathfrak{V} satisfies the consequent of Formula (18) (since v was chosen arbitrary), which implies that \mathfrak{V} satisfies Formula (18). Finally,

we note that these arguments are true for each $\iota \in Uni$ and, thus, \mathfrak{V} satisfies φ_n^{Uni} . \square

THE FORMULA φ_n^{Ex}

The formula φ_n^{Ex} needs to enforce that $L(\mathcal{A}_{\mathfrak{M}})$ respects all existential implications in Ex . Similar to the previous formula, we construct for each existential implication $\iota = (u, \mathcal{A}) \in Ex$ a formula ϕ_n^ι that asserts $L(\mathcal{A}_{\mathfrak{M}}) \cap L(\mathcal{A}) \neq \emptyset$ if $u \in L(\mathcal{A}_{\mathfrak{M}})$. The formulas φ_n^{Ex} is then the (finite) conjunction $\bigwedge_{\iota \in Ex} \phi_n^\iota$.

The formulas ϕ_n^ι work similar to the formulas φ_n^{Uni} introduced above. Given an existential implication $\iota = (u, \mathcal{A})$, say with $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, q_0^{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$, the key idea is again to track the runs of $\mathcal{A}_{\mathfrak{M}}$ and \mathcal{A} in parallel. In contrast to φ_n^{Uni} , however, it is no longer sufficient to build upon the variables $y_{q,q'}$ as they do not track the runs exactly; recall that $y_{q,q'}$ might be set to *true* even without the existence of a word that induces runs to the state $q \in \mathcal{A}_{\mathfrak{M}}$ and $q' \in \mathcal{A}$. This fact prevents us from enforcing the existence of a word in the intersection $L(\mathcal{A}_{\mathfrak{M}}) \cap L(\mathcal{A})$ based on the variables $y_{q,q'}$ (should this be necessary due to $\mathcal{A}_{\mathfrak{M}}$ accepting the antecedent of ι).

We approach this problem by tracking the parallel runs of $\mathcal{A}_{\mathfrak{M}}$ and \mathcal{A} exactly, exploiting the following simple fact about finite automata.

Observation 1: Let $\mathcal{B}_1 = (Q_{\mathcal{B}_1}, \Sigma, q_0^{\mathcal{B}_1}, \Delta_{\mathcal{B}_1}, F_{\mathcal{B}_1})$ and $\mathcal{B}_2 = (Q_{\mathcal{B}_2}, \Sigma, q_0^{\mathcal{B}_2}, \Delta_{\mathcal{B}_2}, F_{\mathcal{B}_2})$ be two NFAs. Then, a word $w \in \Sigma^*$ with $\mathcal{B}_1: q_0^{\mathcal{B}_1} \xrightarrow{w} q$ and $\mathcal{B}_2: q_0^{\mathcal{B}_2} \xrightarrow{w} q'$ exists if and only if a word $w' \in \Sigma^*$ of length at most $|Q_{\mathcal{B}_1}| |Q_{\mathcal{B}_2}| - 1$ with $\mathcal{B}_1: q_0^{\mathcal{B}_1} \xrightarrow{w'} q$ and $\mathcal{B}_2: q_0^{\mathcal{B}_2} \xrightarrow{w'} q'$ exists.

To see why Observation 1 is true, suppose there exists an input $w \in \Sigma^*$ of length greater than $k = |Q_{\mathcal{B}_1}| |Q_{\mathcal{B}_2}| - 1$ with $\mathcal{B}_1: q_0^{\mathcal{B}_1} \xrightarrow{w} q$ and $\mathcal{B}_2: q_0^{\mathcal{B}_2} \xrightarrow{w} q'$. Then, there has to be a pair of states occurring in these runs that repeats at least once. The (nonempty) part of w in between this repetition can be removed, resulting in a word w' with $\mathcal{B}_1: q_0^{\mathcal{B}_1} \xrightarrow{w'} q$ and $\mathcal{B}_2: q_0^{\mathcal{B}_2} \xrightarrow{w'} q'$. By repeating this argument successively, one obtains a word of length less or equal to k that leads to state q in \mathcal{B}_1 and state q' in \mathcal{B}_2 .

As Observation 1 shows, it is indeed enough to consider words of length at most $k = n|\mathcal{A}| - 1$ in order to track the parallel runs of $\mathcal{A}_{\mathfrak{M}}$ and \mathcal{A} exactly. We do so by means of new auxiliary variables $z_{q,q',\ell}^\iota$ where $q \in Q$, $q' \in Q_{\mathcal{A}}$, and $\ell \in \{0, \dots, k\}$, which have the following meaning: the variable $z_{q,q',\ell}^\iota$ is set to *true* if and only if there exists a word $v \in \Sigma^*$ with $|v| = \ell$ such that $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} q$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{v} q'$. The following formulas constrain the variables $z_{q,q',\ell}$ as described.

$$z_{q_0, q_0^{\mathcal{A}}, 0}^\iota \wedge \bigwedge_{(q, q') \in Q \times Q_{\mathcal{A}} \setminus \{(q_0, q_0^{\mathcal{A}})\}} \neg z_{q, q', 0}^\iota \quad (19)$$

$$\bigwedge_{p, q \in Q} \bigwedge_{(p', a, q') \in \Delta_{\mathcal{A}}} \bigwedge_{\ell \in \{0, \dots, k-1\}} (z_{p, p', \ell}^\iota \wedge d_{p, a, q}) \rightarrow z_{q, q', \ell+1}^\iota \quad (20)$$

$$\bigwedge_{q \in Q} \bigwedge_{q' \in Q_{\mathcal{A}}} \bigwedge_{\ell \in \{1, \dots, k\}} z_{q, q', \ell}^\iota \rightarrow \bigvee_{p \in Q} \bigvee_{(p', a, q') \in \Delta_{\mathcal{A}}} d_{p, a, q} \wedge z_{p, p', \ell-1}^\iota \quad (21)$$

Formula (19) makes sure that $z_{q_0, q_0^{\mathcal{A}}, 0}^\iota$ is set to *true*, whereas all other variables $z_{q, q', 0}^\iota$ are set to *false*, since $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{\epsilon} q_0$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{\epsilon} q_0^{\mathcal{A}}$ are the only runs on the empty word. Formula (20) is similar to Formula (13) and describes how the runs of both automata proceed: if there exists a word $v \in \Sigma^*$ with $|v| < k$ that induces the runs $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} q$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{v} q'$ (i.e., $z_{q, q', |v|}^\iota$ is set to *true*) and there exists transitions $(p', a, q') \in \Delta_{\mathcal{A}}$ and $\delta(p, a) = q$ (i.e., $d_{p, a, q}$ is set to *true*), then the word va induces the runs $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{va} q$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{va} q'$, which implies that $z_{q, q', |va|}^\iota$ has to be set to *true* as well. In a similar manner, Formula (21) prevents $z_{q, q', \ell}^\iota$ from being set to *true* if there exists no input of length ℓ that leads to the states q in $\mathcal{A}_{\mathfrak{M}}$ and state q' in \mathcal{A} ; an exemption to this constraint is the pair of initial states.

Finally, adding the implication

$$\left(\bigvee_{q \in Q} x_{u, q} \wedge f_q \right) \rightarrow \left(\bigvee_{q \in Q} \bigvee_{q' \in F_{\mathcal{A}}} \bigvee_{\ell \in \{0, \dots, k\}} z_{q, q', \ell}^\iota \wedge f_q \right) \quad (22)$$

enforces that $L(\mathcal{A}_{\mathfrak{M}})$ indeed respects the implication $\iota = (u, \mathcal{A})$: if $\mathcal{A}_{\mathfrak{M}}$ accepts u (signaled by the antecedent being *true*), then there also has to exist an input on which both automata reach final states (indicated by the consequent being set to *true*), hence, proving $L(\mathcal{A}_{\mathfrak{M}}) \cap L(\mathcal{A}) \neq \emptyset$.

Let $\phi_n^\iota(\bar{d}, \bar{f}, \bar{x}, \bar{z}^\iota)$ be the conjunction of Formulas (19)–(22) where \bar{d} , \bar{f} , and \bar{x} are as above and \bar{z}^ι is a list of variables $z_{q, q', \ell}^\iota$ for $q \in Q$, $q' \in Q_{\mathcal{A}}$, and $\ell \in \{0, \dots, k\}$. Moreover, let φ_n^{Ex} be the conjunction

$$\varphi_n^{Ex}(\bar{d}, \bar{f}, \bar{x}, \bar{z}) := \bigwedge_{\iota \in Ex} \phi_n^\iota(\bar{d}, \bar{f}, \bar{x}, \bar{z}^\iota),$$

where \bar{z} denotes the list of all variables occurring in \bar{z}^ι . Then, the following holds.

Lemma 7: Let $\mathcal{S} = (Pos, Neg, Ex, Uni)$ be a sample, $n \geq 1$, and

$$\psi_n^{Ex}(\bar{d}, \bar{f}, \bar{x}, \bar{z}) := \varphi_n^{DFA}(\bar{d}, \bar{f}) \wedge \varphi_n^W(\bar{d}, \bar{f}, \bar{x}) \wedge \varphi_n^{Ex}(\bar{d}, \bar{f}, \bar{x}, \bar{z}).$$

Then, the following statements hold:

- 1) If $\mathcal{M} \models \psi_n^{Ex}$, then $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that satisfies for all $(u, \mathcal{A}) \in Ex$ that $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}_{\mathfrak{M}}) \cap L(\mathcal{A}) \neq \emptyset$.
- 2) If a DFA with n states exists that satisfies for all $(u, \mathcal{A}) \in Ex$ that $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}_{\mathfrak{M}}) \cap L(\mathcal{A}) \neq \emptyset$, then ψ_n^{Ex} is satisfiable.

Proof of Lemma 7: This proof is similar to the proof of Lemma 6. Again, we split this proof into two part: we first prove Statement 1 and subsequently Statement 2.

To prove Statement 1, we show that for an existential implication $\iota = (u, \mathcal{A}) \in Ex$, a model of the formula

$$\psi_n^t(\bar{d}, \bar{f}, \bar{x}, \bar{z}) := \varphi_n^{\text{DFA}}(\bar{d}, \bar{f}) \wedge \varphi_n^W(\bar{d}, \bar{f}, \bar{x}) \wedge \phi_n^t(\bar{d}, \bar{f}, \bar{x}, \bar{z}^t)$$

results in an automaton $\mathcal{A}_{\mathfrak{M}}$ that respects ι (i.e., $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}_{\mathfrak{M}}) \cap L(\mathcal{A}) \neq \emptyset$). The claim of Statement 1 then follows immediately because φ_n^{Ex} is the conjunction of the individual formulas ϕ_n^t . In the following, fix an existential implication $\iota = (u, \mathcal{A}) \in Ex$, assume $\mathfrak{M} \models \psi_n^t$, let $\mathcal{A}_{\mathfrak{M}}$ be the DFA constructed according to Definition 6 and $k = n|Q_{\mathcal{A}}| - 1$.

We first prove that the variable $z_{q,q',\ell}^t$, where $\ell \in \{0, \dots, k\}$, is set to *true* if and only if there exists a $v \in \Sigma^*$ with $|v| \leq \ell$ such that $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} q$ and $\mathcal{A}: q_0^A \xrightarrow{v} q'$. This proof proceeds by induction over ℓ .

Base case ($\ell = 0$) The empty word ε is the unique word $v \in \Sigma^*$ with $|v| = 0$. By definition of runs, $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{\varepsilon} q_0$ and $\mathcal{A}: q_0^A \xrightarrow{\varepsilon} q_0^A$. Moreover, Formula (19) makes sure that $z_{q_0, q_0^A, 0}^t$ is set to *true*, whereas $z_{q, q', 0}^t$ is set to *false* for all other pairs of states. In addition, Formula (21) does not restrict any variable in the case $\ell = 0$. Hence, the claim holds.

Induction step ($\ell = \ell' + 1$) To prove the direction from left to right, assume $\mathfrak{M}(y_{q,q',\ell}^t) = \text{true}$. Then, Formula (21) asserts that there exists a state $p \in Q$ and a transition $(p', a, q') \in \Delta$ such that $\mathfrak{M}(z_{p,p',\ell'}^t) = \text{true}$ and $\mathfrak{M}(d_{p,a,q}) = \text{true}$ (the latter means that $\mathcal{A}_{\mathfrak{M}}$ contains the transition $\delta(p, a) = q$). In addition, applying the induction hypothesis yields that there exists a word $v' \in \Sigma^*$ with $|v'| = \ell'$ such that $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v'} p$ and $\mathcal{A}: q_0^A \xrightarrow{v'} p'$. Thus, $v = v'a$ is a word of length ℓ satisfying $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} q$ and $\mathcal{A}: q_0^A \xrightarrow{v} q'$, which proves the claim.

To prove the reverse direction, let $v = v'a \in \Sigma^*$ be a word of length ℓ and assume that $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} p \xrightarrow{a} q$ and $\mathcal{A}: q_0^A \xrightarrow{v} p' \xrightarrow{a} q'$. Thus, we know that $(p', a, q') \in \Delta_{\mathcal{A}}$ and $\delta(p, a) = q$ (the latter implying $\mathfrak{M}(d_{p,a,q}) = \text{true}$). In addition, applying the induction hypothesis yields $\mathfrak{M}(z_{p,p',\ell'}^t) = \text{true}$. In this situation, Formula (20) enforces that $z_{q,q',\ell}$ has to be set to *true*, which proves the claim.

Having established the correct meaning of the variables $z_{q,q',\ell}$, proving that $\mathcal{A}_{\mathfrak{M}}$ satisfies $L(\mathcal{A}_{\mathfrak{M}}) \cap L(\mathcal{A}) \neq \emptyset$ if $u \in L(\mathcal{A}_{\mathfrak{M}})$ is now straightforward: If $u \in L(\mathcal{A}_{\mathfrak{M}})$, say $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{u} q$ with $q \in F$, then we know that $x_{u,q}$ is set to *true* (by Lemma 3) and that $\mathfrak{M}(f_q) = \text{true}$ (by Definition 6). In this situation, the antecedent of Formula (19) is satisfied, which implies that its consequent is satisfied as well (since \mathfrak{M} is a model of ψ_n^t). This means that there exist $q \in Q$, $q' \in F_{\mathcal{A}}$, and $\ell \in \{0, \dots, k\}$ such that both $\mathfrak{M}(z_{q,q',\ell}^t) = \text{true}$ and $\mathfrak{M}(f_q) = \text{true}$. The former asserts that there exists a word $v \in \Sigma^*$ (of length ℓ) such that $\mathcal{A}_{\mathfrak{M}}: q_0 \xrightarrow{v} q$ and $\mathcal{A}: q_0^A \xrightarrow{v} q'$ (according to the induction above); on the other hand, the latter means $q \in F$. Hence v is accepted by both automata and, consequently, $u \in L(\mathcal{A}_{\mathfrak{M}})$ implies $L(\mathcal{A}_{\mathfrak{M}}) \cap L(\mathcal{A}) \neq \emptyset$.

To prove Statement 2, let $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, q_0^{\mathcal{B}}, \delta_{\mathcal{B}}, F_{\mathcal{B}})$ be a DFA with n states that satisfies $L(\mathcal{B}) \cap L(\mathcal{A}) \neq \emptyset$ if $u \in L(\mathcal{B})$ for

all $(u, \mathcal{A}) \in Ex$. Similar to the previous proofs, we translate \mathcal{B} into a satisfying valuation \mathfrak{V} of the variables \bar{d} , \bar{f} , \bar{x} , and \bar{z} . For the sake of this translation, we once more assume without loss of generality that the sets of states of \mathcal{B} and $\mathcal{A}_{\mathfrak{M}}$ coincide (i.e., $Q_{\mathcal{B}} = Q$). The definition of \mathfrak{V} then is as follows:

- For each $p, q \in Q_{\mathcal{B}}$ and $a \in \Sigma$, we set $\mathfrak{V}(d_{p,a,q}) = \text{true}$ if and only if $\delta_{\mathcal{B}}(p, a) = q$.
- For each $q \in Q_{\mathcal{B}}$, we set $\mathfrak{V}(f_q) = \text{true}$ if and only if $q \in F_{\mathcal{B}}$.
- For each $u \in W$ and $q \in Q_{\mathcal{B}}$, we set $\mathfrak{V}(x_{u,q}) = \text{true}$ if and only if $\mathcal{B}: q_0^{\mathcal{B}} \xrightarrow{u} q$.
- For each $\iota = (u, \mathcal{A}) \in Ex$, where $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, q_0^{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$, $q \in Q_{\mathcal{B}}$, and $q' \in Q_{\mathcal{A}}$, we set $\mathfrak{V}(z_{q,q',\ell}^t) = \text{true}$ if and only if there exists a word $v \in \Sigma^*$ with length $\ell \leq n|Q_{\mathcal{A}}| - 1$ such that $\mathcal{B}: q_0^{\mathcal{B}} \xrightarrow{v} q$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{v} q'$.

It is not hard to verify that \mathfrak{V} satisfies $\varphi_n^{\text{DFA}} \wedge \varphi_n^W$. To see why it also satisfies φ_n^{Ex} , pick a universal implication $(u, \mathcal{A}) \in Ex$, say with $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, q_0^{\mathcal{A}}, \Delta_{\mathcal{A}}, F_{\mathcal{A}})$, and let $k = |Q_{\mathcal{B}}||Q_{\mathcal{A}}|$ (recall that $|Q_{\mathcal{B}}| = n = |Q|$). First, it is not hard to see that \mathfrak{V} satisfies Formulas (19) to (21) since these formulas exactly describe the runs of \mathcal{B} and \mathcal{A} on words of length at most k . Second, if $u \notin L(\mathcal{B})$, then \mathfrak{V} does not satisfy the antecedent of Formula (22) and, hence, satisfies Formula (22). If $u \in L(\mathcal{B})$, on the other hand, we know that $L(\mathcal{B}) \cap L(\mathcal{A}) \neq \emptyset$.

In other words, there exists a word $v \in L(\mathcal{B}) \cap L(\mathcal{A})$ such that $\mathcal{B}: q_0^{\mathcal{B}} \xrightarrow{v} q$ and $\mathcal{A}: q_0^{\mathcal{A}} \xrightarrow{v} q'$ where $q \in F_{\mathcal{B}}$ and $q' \in F_{\mathcal{A}}$. Moreover, Observation 1 allows us to assume without loss of generality that $|v| \leq k$. In this situation, $\mathfrak{V}(z_{q,q',|v|}^t) = \text{true}$ and $\mathfrak{V}(f_q) = \text{true}$ holds by definition of \mathfrak{V} . Hence, \mathfrak{V} satisfies the consequent of Formula (22), which implies that \mathfrak{V} satisfies Formula (22) as well. Finally, since these arguments are true for each $\iota \in Ex$, the valuation \mathfrak{V} satisfies φ_n^{Ex} . \square

APPENDIX B

CORRECTNESS OF THE SAT LEARNER

The fact that formula φ_n^S has the desired properties is a straightforward corollary of Lemmas 4 to 7.

Corollary 1: Let $\mathcal{S} = (\text{Pos}, \text{Neg}, \text{Ex}, \text{Uni})$ be a sample, $n \geq 1$, and

$$\begin{aligned} \varphi_n^S(\bar{d}, \bar{f}, \bar{x}, \bar{y}, \bar{z}) := & \varphi_n^{\text{DFA}}(\bar{d}, \bar{f}) \wedge \varphi_n^W(\bar{d}, \bar{f}, \bar{x}) \wedge \varphi_n^{\text{Pos}}(\bar{d}, \bar{f}, \bar{x}) \\ & \wedge \varphi_n^{\text{Neg}}(\bar{d}, \bar{f}, \bar{x}) \wedge \varphi_n^{\text{Uni}}(\bar{d}, \bar{f}, \bar{x}, \bar{y}) \wedge \varphi_n^{\text{Ex}}(\bar{d}, \bar{f}, \bar{x}, \bar{z}). \end{aligned}$$

Then, the following statements hold:

- 1) If $\mathcal{M} \models \varphi_n^S$, then $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that is consistent with \mathcal{S} .
- 2) If a DFA with n states exists that is consistent with \mathcal{S} , then φ_n^S is satisfiable.

Having established that formula φ_n^S has the desired properties, we can now show that Algorithm 2 computes a smallest DFA that is consistent with a given sample.

Theorem 3: Given a contradiction free-sample \mathcal{S} , Algorithm 2 returns a minimal DFA (in terms of the number of states) that is consistent with \mathcal{S} . In addition, if a minimal consistent DFA has k states, then Algorithm 2 terminates after k iterations.

Proof of Theorem 3: Theorem 3 follows directly from the properties of the formula φ_n^S (see Corollary 1): Given a sample \mathcal{S} , suppose that a DFA with k states that is consistent with \mathcal{S} exists. Then, the formula φ_n^S is satisfiable for all $n \geq k$. Moreover, if $\mathfrak{M} \models \varphi_n^S$, then $\mathcal{A}_{\mathfrak{M}}$ is a DFA with n states that is consistent with \mathcal{S} . Since Algorithm 2 increases the parameter n by one in every iteration (starting with $n = 1$), the algorithm eventually finds the smallest value for which φ_n^S is satisfiable (after k iterations) and, thus, a consistent DFA of minimal size. \square

We are now ready to prove the correctness of the SAT learner.

Theorem 4: Given a teacher for a rational safety game, Algorithm 1, equipped with Algorithm 2 to construct conjectures, terminates and returns a (minimal) DFA accepting a winning set if one exists.

Proof of Theorem 4: Due to the way the teacher answers queries, it is clear that the DFA returned by the SAT learner accepts a winning set. Thus, it is left to show that the SAT learner terminates (given that a winning set exists) and that its result is of minimal size. To this end, we first make three observations:

- 1) The SAT learner never conjectures the same DFA twice. This is due to the fact that the SAT learner only conjectures DFAs that are consistent with the sample of the iteration in which it was constructed. Moreover, a simple proof by contradiction shows that the conjecture of the current iteration is also consistent with the samples of all previous iterations since a new sample results from adding a counterexample (i.e., a word or an implication) to the sample of the previous iteration. Hence, the conjectures \mathcal{A}_i of iteration i and \mathcal{A}_j of iteration $j < i$ differ at least on the counterexample added in iteration j .
- 2) The SAT learner conjectures DFAs that grow monotonically in size. To see why, suppose that conjecture \mathcal{A}_{i+1} of iteration $i + 1$ has less states than the conjecture \mathcal{A}_i of iteration i . As argued above, \mathcal{A}_{i+1} is also consistent with the sample \mathcal{S}_i , but has fewer states than \mathcal{A}_i . This, however, contradicts the fact that Algorithm 2 always constructs consistent DFAs of minimal size (see Theorem 3).
- 3) Any DFA accepting a winning set is consistent with any sample produced during the learning. In other words, adding counterexamples does not rule out solutions.

Theorem 4 can now be proven as follows. Suppose that a winning set exists and let \mathcal{A} be a smallest DFA, say with k states, that accepts a winning set. Since no smaller DFA accepting a winning set exists and due to Observations 1 and 2, we know that the SAT learner eventually conjectures a DFA with at least k states. Towards a contradiction, assume that the SAT learner does not conjecture a DFA with k accepting a winning set. This means that the learner eventually conjectures a DFA with more than k states. Then, however, Observation 3 in connection with the fact that the SAT learner always produces smallest consistent DFAs implies that there exists no DFA with k states accepting a winning set. This is a contradiction. Hence, the SAT learner eventually conjectures a minimal DFA

accepting a winning set, which passes the teacher's query, and terminates. \square

APPENDIX C RPNI LEARNER

The RPNI learner works in a restricted setting in which every vertex of the arena has a finite (but not necessarily bounded) number of outgoing edges (i.e., $E(\{v\})$ is finite for all $v \in V$). This implies that implication counterexamples are of the form (u, \mathcal{A}) with $L(\mathcal{A})$ being finite.

The RPNI learner works identical to the SAT learner, but uses a different method to construct a consistent DFA from a sample. While the SAT learner uses a constraint solver for this task (see Algorithm 2), the RPNI learner employs a modified version of the popular *RPNI algorithm* [18], which is a polynomial time heuristic for learning DFAs from positive and negative words (we adapted the RPNI algorithm such that it now learns DFAs not only from positive and negative words but also from existential and universal implications). In contrast to Algorithm 2, however, the modified RPNI algorithm does not, in general, produce minimal consistent DFAs but is much faster. Hence, we encourage the reader to think of the RPNI learner as a heuristic, which uses a faster means to construct conjectures but can no longer guarantee to terminate given that a winning set exists.

As a preparatory step, we first present the original RPNI algorithm. Then, we show how to modify the RPNI algorithm such that it can handle existential and universal in addition to positive and negative words. Finally, we present the RPNI learner and

A. The RPNI Algorithm

The RPNI algorithm is a so-called *passive learning algorithm* for regular languages. It takes two disjoint, finite sets $Pos \subset \Sigma^*$ and $Neg \subset \Sigma^*$ as input and constructs a DFA \mathcal{A} that satisfies $Pos \subseteq L(\mathcal{A})$ and $Neg \cap L(\mathcal{A}) = \emptyset$. The algorithm runs in time and space polynomial in $|Pref(Pos \cup Neg)|$ and, hence, the constructed DFA can, in general, not be minimal (as the problem it solves is NP-complete, see Gold [7]). It turns out, however, that the RPNI algorithm often produces “small” automata in practice.

The RPNI algorithm operates on given sets Pos and Neg as follows. It first constructs the prefix-tree acceptor of the set Pos (i.e., the tree-like automaton that accepts exactly the set Pos). Then, it successively tries to merge states of this automaton (in a fixed order), where a merge is considered to be successful if the resulting DFA still rejects all words in Neg . If a merge was successful, RPNI proceeds to merge further states of the resulting automaton. If it was not successful, the merged automaton is discarded and RPNI proceeds with the automaton of the last successful merge. The algorithm stops once there are no more merges left.

For our purpose, it is helpful to view the RPNI algorithm as a concrete instance of a generic state-merging algorithm,

which is sketched in pseudo code as Algorithm 3.⁴ In this more abstract setting, the learning algorithm takes a finite collection κ of data as input and outputs a DFA that satisfies a given (decidable) property p (which usually refers to κ); in the case of RPNI, κ is the pair (Pos, Neg) and the property p states that the resulting DFA has to accept all words in Pos and to reject all words in Neg . The pivotal idea of Algorithm 3 is to start with a potentially large initial DFA that satisfies property p and then reduce its size by merging states, thereby discarding merges that result in a DFA that violates p . Since merging states of a DFA increase its language, we encourage the reader to think of merging as a means of generalization.

Algorithm 3: Generic state-merging algorithm

Input: A collection of data κ
Output: A DFA machine \mathcal{A} that passes $test(\mathcal{A})$

```

1  $\mathcal{A}_{init} = (Q, \Sigma, q_0, \delta, f) \leftarrow init(\kappa);$ 
2  $(q_0, \dots, q_n) \leftarrow order(Q);$ 
3  $\sim_0 \leftarrow \{(q, q) \mid q \in Q\};$ 
4 for  $i = 1, \dots, n$  do
5   if  $q_i \not\sim_{i-1} q_j$  for all  $j \in \{0, \dots, i-1\}$  then
6      $j \leftarrow 0;$ 
7     repeat
8       Let  $\sim$  be the smallest congruence that
       contains  $\sim_{i-1}$  and the pair  $(q_i, q_j);$ 
9        $j \leftarrow j + 1;$ 
10    until  $test(\mathcal{A}_{init}/\sim);$ 
11     $\sim_i \leftarrow \sim;$ 
12  else
13     $\sim_i \leftarrow \sim_{i-1};$ 
14
15 end
16 return  $\mathcal{A}_{init}/\sim_n;$ 
```

Algorithm 3 uses three functions *init*, *order*, and *test*, which have the following effects:

- The function *init* receives a finite collection of data as input and returns a (potentially large) DFA that satisfies property p (assuming that this is possible).
- The function *order* receives a finite set Q as input and returns an ordered sequence of the elements of Q .
- The function *test* receives a DFA as input and returns a Boolean value indicating whether this DFA satisfies property p .

(We shortly introduce implementations of these functions that allows us to compute a DFA that is consistent with a given finite sample.)

Algorithm 3 runs in two consecutive phases. In the first phase (Lines 1 and 2), it calls the function *init* with parameter κ to construct an initial DFA \mathcal{A}_{init} that satisfies p (recall that we assume that this is possible). Then, it fixes an order

q_0, \dots, q_n of the states of \mathcal{A}_{init} by calling the function *order* with parameter Q .

The actual merging takes place in the second phase (Lines 3 to 15), according to the order determined in the first phase. For $i = 1, \dots, n$ and $j = 0, \dots, i-1$, the algorithm tries to merge state q_i with state q_j if state q_i has not already been merged with a smaller state; since a merge might introduce nondeterminism, the algorithm merges additional states until determinism is restored. Note that we represent merging of states abstractly as constructing a congruence relation $\sim \subseteq Q \times Q$ (i.e., an equivalence relation that is compatible with the transition function) and the result of the merging as the quotient automaton \mathcal{A}_{init}/\sim , which is defined in the usual way. A merge is kept only if the resulting automaton passes *test* (otherwise it is discarded). This preserves the invariant that any intermediate DFA $\mathcal{A}_{init}/\sim_k$ satisfies property p (since $\mathcal{A}_{init}/\sim_0 = \mathcal{A}_{init}$ satisfies p by definition of *init*). Hence, the final DFA is guaranteed to satisfy p as well.

B. Adapting the Generic State Merging Algorithm

In our setting, the collection κ corresponds to a sample $S = (Pos, Neg, Ex, Uni)$, and the property p is consistency with S . We now describe how to implement the functions *init*, *order*, and *test* such that the output of Algorithm 3 is a DFA that is consistent with the input-sample S .

a) *Creating an initial DFA:* Given a sample S , we need to construct a DFA satisfying p (i.e., a DFA consistent with S). To this end, we follow the idea of the RPNI algorithm, namely to construct the prefixtree acceptor of the set Pos . The prefix tree acceptor of a finite set $X \subseteq \Sigma^*$ is a partial DFA⁵ that accepts exactly the set X . It is defined as follows.

Definition 7: Given an alphabet Σ and finite set $X \subseteq \Sigma^*$, the *prefix tree acceptor* is the partial DFA $\mathcal{A}_X = (Q, \Sigma, q_0, \delta, F)$ defined by

- $Q = Pref(X);$
- $q_0 = \varepsilon;$
- $F = X;$ and
- $\delta(u, a) = \begin{cases} ua & \text{if } ua \in Pref(X) \text{ and;} \\ \text{undefined} & \text{otherwise.} \end{cases}$

A straightforward induction over the length of input-words proves $L(\mathcal{A}_X) = X$.

However, just starting with the prefix tree acceptor \mathcal{A}_{Pos} is not sufficient as \mathcal{A}_{Pos} is not necessarily consistent with S : an implication (u, \mathcal{A}) might require to accept a word $v \in L(\mathcal{A})$ (because $u \in L(\mathcal{A}_{Pos})$) that is not an element of Pos and, hence, is rejected by \mathcal{A}_{Pos} . In the case of universal implications, the problem is easy to resolve by (temporarily) adding $L(\mathcal{A})$ to Pos (recall that $L(\mathcal{A})$ is finite). However, the problem becomes more involved in the presence of existential implications as it is no longer apparent which word $v \in L(\mathcal{A})$ one should add to Pos in order to obtain a consistent (and preferable small) prefix tree acceptor.

⁴The description here closely follows the more general description by Garg et al. [6].

⁵A DFA is called *partial* if not all transitions are defined. Runs that cannot be continue due to missing transition are considered to be rejecting.

We approach this problem by using a straightforward translation into a satisfiability problem of formulas in propositional Boolean logic (the resulting satisfiability problem is much simpler than those generated by the SAT learner as it does not involve finding a minimal solution). Given a sample $\mathcal{S} = (Pos, Neg, Ex, Uni)$, we introduce a Boolean variable x_w for each word w of the set

$$V = Pos \cup Neg \cup Ante(Ex) \cup Ante(Uni) \\ \cup \left(\bigcup_{(u, \mathcal{A}) \in Ex} L(\mathcal{A}) \right) \cup \left(\bigcup_{(u, \mathcal{A}) \in Uni} L(\mathcal{A}) \right),$$

which consists of all words occurring (explicitly and implicitly) in \mathcal{S} . Since the languages of the automata occurring in \mathcal{S} is finite, V is a finite set and, hence, the number of variables is finite as well.

The desired meaning of the variables is the following: x_w is set to *true* if w either belongs to Pos or it is needed to be added to Pos in order to satisfy the implications. The following constraints enforce this meaning.

$$\left(\bigwedge_{w \in Pos} x_w \right) \wedge \left(\bigwedge_{w \in Neg} \neg x_w \right) \quad (23)$$

$$\bigwedge_{(u, \mathcal{A}) \in Ex} \left(x_u \Rightarrow \bigvee_{v \in L(\mathcal{A})} x_v \right) \quad (24)$$

$$\bigwedge_{(u, \mathcal{A}) \in Uni} \left(x_u \Rightarrow \bigwedge_{v \in L(\mathcal{A})} x_v \right) \quad (25)$$

Let $\chi(\bar{x})$ be the conjunction of Formulas (23), (24), and (25) where \bar{x} is the list of all variables $w \in V$. Then, $\chi(\bar{x})$ is satisfiable since we assume any sample to be contradiction-free. Moreover, if \mathfrak{M} is a model of $\chi(\bar{x})$, then the prefix tree acceptor $\mathcal{A}_{Pos'}$ of the set

$$Pos' = \{w \in V \mid \mathfrak{M}(w) = \text{true}\}$$

is consistent with \mathcal{S} (i.e., satisfies p), which is formalized by the lemma below. This automaton is what the function *init* returns.

Lemma 8: Let $\mathcal{S} = (Pos, Neg, Ex, Uni)$ a contradiction-free sample. Then, the following holds:

- 1) The formula $\chi(\bar{x})$ is satisfiable.
- 2) If \mathfrak{M} a model of $\chi(\bar{x})$ and

$$Pos' = \{w \in V \mid \mathfrak{M}(w) = \text{true}\},$$

then the prefix tree acceptor $\mathcal{A}_{Pos'}$ is consistent with \mathcal{S} .

Proof of Lemma 8: Since \mathcal{S} is contradiction-free, there exists a DFA, let us denote it by \mathcal{B} , that is consistent with \mathcal{S} . If we assign *true* to the variable x_w if and only if $w \in L(\mathcal{B})$, then this assignment satisfies $\chi(\bar{x})$. This proves the first claim.

The proof of the second claim relies on the fact that the prefix tree acceptor of a set $X \subseteq \Sigma^*$ indeed accepts exactly the set X , which can be shown by a simple induction. Given

this fact, we first observe that $\mathcal{A}_{Pos'}$ accepts all words in Pos since $L(\mathcal{A}_{Pos'}) = Pos'$ and Formula (23) ensures that $Pos \subseteq Pos'$; moreover, a similar argument shows that $\mathcal{A}_{Pos'}$ rejects all words in Neg . Second, Formula (24) asserts for each existential implication $(u, \mathcal{A}) \in Ex$ that $u \in Pos'$ implies the existence of a $v \in L(\mathcal{A})$ with $v \in Pos' = L(\mathcal{A}_{Pos'})$. Hence, $\mathcal{A}_{Pos'}$ respects all existential implications. Moreover, one can establish the fact that $\mathcal{A}_{Pos'}$ respects all universal implications in an analogous manner by referring to Formula (25). \square

b) Choosing the Merging Order: The function *init* returns a DFA whose set of states consists of words over the alphabet Σ . The order function *order* takes this set and orders it according to the canonical order of words⁶. This order is also used by RPNI.

c) Implementing the Test: The function *test* needs to check whether a given automaton \mathcal{A} is consistent with the finite sample \mathcal{S} . Since \mathcal{S} is a finite collection of words, consistency can be decided easily by computing the runs of \mathcal{A} on those words and checking whether all four conditions (i.e., acceptance of all words in Pos , rejection of all words in Neg , and respecting both types of implications) are fulfilled.

C. Correctness of the RPNI learner

The correctness of the RPNI learner relies on the correctness of Algorithm 3, which is stated in the next lemma.

Lemma 9: Given a contradiction-free sample \mathcal{S} , Algorithm 3 modified as described in Appendix C-B constructs a DFA that is consistent with \mathcal{S} . The resulting automaton comprises at most $|V|$ states.

Proof of Lemma 9: Proving that Algorithm 3 constructs a DFA that is consistent with the given sample \mathcal{S} is straightforward: the function *init* constructs an initial DFA that is consistent with \mathcal{S} (see Lemma 8), and a merge is only kept if the merged DFA passes the check *test* (i.e., it is still consistent); hence, the final DFA is guaranteed to be consistent as well. Since the initial DFA has $|V|$ states and merging of states reduces the number of states, the final DFA has at most $|V|$ states. \square

The correctness of the RPNI learner immediately follows from the fact that the learning terminates only if the learner proposes a DFA accepting a winning set. In contrast to the SAT learner, however, the RPNI learner uses an algorithm to derive conjectures that does not necessarily produce consistent DFAs of minimal size. As a consequence, termination of the RPNI learner is not guaranteed even if a DFA accepting a winning set exists. The following theorem summarizes the main result.

Theorem 5: Given a teacher for a rational safety game over a finitely branching arena, the RPNI learner (i.e., Algorithm 1 equipped with Algorithm 3 to construct conjectures) on termination returns a DFA accepting a winning set.

⁶Given an alphabet Σ and a total order $<_{\Sigma} \subseteq \Sigma \times \Sigma$, the *canonical order of words* $< \subseteq \Sigma^* \times \Sigma^*$ is defined by $a_1 \dots a_m < b_1 \dots b_n$ if and only if $m < n$ or there exists an $i \in \{1, \dots, m\}$ such that $a_i <_{\Sigma} b_i$ and $a_j = b_j$ for all $j \in \{1, \dots, i-1\}$.