Computing Inductive Invariants of Regular Abstraction Frameworks

Philipp Czerner
Technical University of Munich, Germany

Javier Esparza
Technical University of Munich, Germany

Valentin Krasotin
Technical University of Munich, Germany

Christoph Welzel-Mohr
Technical University of Munich, Germany

Abstract

Regular transition systems (RTS) are a popular formalism for modeling infinite-state systems in general, and parameterised systems in particular. In a CONCUR 22 paper, Esparza et al. introduce a novel approach to the verification of RTS, based on inductive invariants. The approach computes the intersection of all inductive invariants of a given RTS that can be expressed as CNF formulas with a bounded number of clauses, and uses it to construct an automaton recognising an overapproximation of the reachable configurations. The paper shows that the problem of deciding if the language of this automaton intersects a given regular set of unsafe configurations is in EXPSPACE and PSPACE-hard.

We introduce *regular abstraction frameworks*, a generalisation of the approach of Esparza et al., very similar to the regular abstractions of Hong and Lin. A framework consists of a regular language of *constraints*, and a transducer, called the *interpretation*, that assigns to each constraint the set of configurations of the RTS satisfying it. Examples of regular abstraction frameworks include the formulas of Esparza et al., octagons, bounded difference matrices, and views. We show that the generalisation of the decision problem above to regular abstraction frameworks remains in EXPSPACE, and prove a matching (highly non-trivial) EXPSPACE-hardness bound.

EXPSPACE-hardness implies that, in the worst case, the automaton recognising the overapproximation of the reachable configurations has a double-exponential number of states. We introduce a learning algorithm that computes this automaton in a lazy manner, stopping whenever the current hypothesis is already strong enough to prove safety. We report on an implementation and show that our experimental results improve on those of Esparza et al.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Problems, reductions and completeness; Theory of computation → Verification by model checking

Keywords and phrases Regular model checking, abstraction, inductive invariants

1 Introduction

Regular transition systems (RTS) are a popular formalism for modelling infinite-state systems satisfying the following conditions: configurations can be encoded as words, the set of initial configurations is recognised by a finite automaton, and the transition relation is recognised by a transducer. Model checking RTS has been intensely studied under the name of *regular model checking* (see [24, 14, 25, 11] and the surveys [5, 1, 6, 2]). Most regular model checking algorithms address the *safety problem*: given a regular set of unsafe configurations, decide if its intersection with the set of reachable configurations is empty or not. They combine algorithms for the computation of increasingly larger regular subsets of the reachable configurations with acceleration, abstraction, and widening techniques [14, 24, 17, 4, 11, 13, 15, 12, 28, 16].

Recently, Esparza et al. have introduced a novel approach that, starting with the set of all configurations of the RTS, computes increasingly smaller inductive invariants, that is, inductive supersets of the reachable configurations. More precisely, [19] considers invariants given by Boolean formulas in conjunctive normal form with at most b clauses. The paper proves that, for every bound $b \geq 0$, the intersection of *all* inductive b-invariants of the system is recognised by a DFA of double exponential size in $\mathcal{I}$ and $\mathcal{T}$. As a corollary, they obtain that, for every $b \geq 0$, deciding if this intersection contains some unsafe configuration is in EXPSPACE. They also show that the problem is PSPACE-hard, and leave the question of closing the gap open.

In [20] (a revised version of [19]), the EXPSPACE proof is conducted in a more general setting than in [19]. Inspired by this, in our first contribution we show that the approach of [19] can be vastly generalised to arbitrary *regular abstraction frameworks*, consisting of a regular language of *constraints*, and an *interpretation*. Interpretations are functions, represented by transducers, that assign to each constraint a set of configurations, viewed as the set of configurations that *satisfy* the constraint. Examples of regular abstraction frameworks include the formulas of [19] for every $b \geq 0$, views [3], and families of Presburger arithmetic formulas like octagons [29] or bounded difference matrices [27, 8]. A framework induces an abstract interpretation, in which, loosely speaking, the word encoding a constraint is the abstraction of the set of configurations satisfying the constraint. Just as regular model checking started with the observation that different classes of *systems* could be uniformly modeled as RTSs [5, 1, 6, 2], we add the observation, also made in [22], that different classes of *abstractions* can be uniformly modeled as regular abstraction frameworks. We show that the generalisation of the verification problem of [19, 20] to arbitrary regular abstraction frameworks remains in EXPSPACE.

In our second contribution we show that our problem is also EXPSPACE-hard. The reduction (from the acceptance problem for exponentially bounded Turing machines) is surprisingly involved. Loosely speaking, it requires to characterise the set of prefixes of the run of a Turing machine on a given word as an intersection of inductive invariants of a very restrictive kind. We think that this construction can be of independent interest.

Our third and final contribution is motivated by the EXPSPACE-hardness result. A consequence of this lower bound is that the automaton recognising the overapproximation of the reachable configurations must necessarily have a double-exponential number of states in the worst case. We present an approach, based on automata learning, that constructs increasingly larger automata that recognise increasingly smaller overapproximations, and checks whether they are precise enough to prove safety. A key to the approach is solving the separability problem: given a pair (c, c') of configurations, is there an inductive constraint that *separates* c and c', i.e. is satisfied by c but not by c'? We show that the problem is PSPACE-complete and NP-complete for interpretations captured by length-preserving transducers. We provide an implementation on top of a SAT solver for the latter case (this is the only case considered in [19, 20]). An experimental comparison shows that this approach beats the one of [19, 20].

Related work. As mentioned above, our first contribution is a reformulation of results of [20] into a more ambitious formalism; it is a conceptual but not a technical novelty. The second and third contributions are new technical results.

Our regular abstraction frameworks are in the same spirit as the regular abstractions of Hong and Lin [22], which use regular languages as abstract objects. In this paper we concentrate on the inductive invariant approach of [19], and in particular on its complexity. This is unlike the approach of [22], which on the one hand is more general, since it also considers liveness properties, but on the other hand does not contain complexity results.

Automata learning has been explored for the verification of regular transition systems multiple times [30, 33, 16, 34, 31]. Roughly speaking, all these approaches formulate a learning process to obtain a *regular* inductive invariant of the system that proves a safety property. Since it is impossible to algorithmically identify the cases where such regular inductive invariant exists, timeouts [16] and resource limits [30] are used as heuristics. In contrast, our approach is designed to always terminate. In particular, we either provide a regular set of constraints that suffices to establish the safety property or a pair of configurations that cannot be separated by inductive constraints of the considered framework. This information can be used to design a more precise framework by adding a new type of constraints.

2 Preliminaries and regular transition systems

Automata. Let Σ be an alphabet. A *nondeterministic finite automaton (NFA)* over Σ is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$ where Q is a finite set of *states*, $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the *transition function*, $Q_0 \subseteq Q$ is the set of *initial states*, and $F \subseteq Q$ is the set of *final states*. A *run* of A on a word $w = w_1 \cdots w_l \in \Sigma^l$ is a sequence $q_0 q_1 \cdots q_l$ of states where $q_0 \in Q_0$ and $\forall i \in [l] : q_i \in \delta(q_{i-1}, w_i)$. A run on w is *accepting* if $q_l \in F$, and A *accepts* w if there exists an accepting run of A on w. The language *recognised* by A, denoted $L(A)$ or L_A, is the set of words accepted by A. If $|Q_0| = 1$ and $|\delta(q, a)| = 1$ for every $q \in Q, a \in \Sigma$ $|Q_0| = 1$, then A is a deterministic finite automaton *(DFA)*. In this case, we write $\delta(q, a) = q'$ instead of $\delta(q, a) = \{q'\}$ and have a single initial state q_0 instead of a set Q_0.

Relations. Let $R \subseteq X \times Y$ be a relation. The *complement* of R is the relation $\overline{R} := \{(x, y) \in X \times Y \mid (u, w) \notin R\}$. The *inverse* of R is the relation $R^{-1} := \{(y, x) \in Y \times X \mid (x, y) \in R\}$. The *projections* of R onto its first and second components are the sets $R|_1 := \{x \in X \mid \exists y \in Y : (x, y) \in R\}$ and $R|_2 := \{y \in Y \mid \exists x \in X : (x, y) \in R\}$. The *join* of two relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ is the relation $R \circ S := \{(x, z) \in X \times Z \mid \exists y \in Y : (x, y) \in R, (y, z) \in S\}$. The *post-image* of a set $X' \subseteq X$ under a relation $R \subseteq X \times Y$, denoted $X' \circ R$ or $R(X')$, is the set $\{y \in Y \mid \exists x \in X' : (x, y) \in R\}$; the *pre-image*, denoted $R \circ Y$ or $R^{-1}(Y)$, is defined analogously. Throughout this paper, we only consider relations where $X = \Sigma^*$ and $Y = \Gamma^*$ for some alphabets Σ, Γ. We just call them relations. A relation $R \subseteq \Sigma^* \times \Gamma^*$ is *length-preserving* if $(u, w) \in R$ implies $|u| = |w|$.

Convolutions and transducers. Let Σ, Γ be alphabets, let $\# \notin \Sigma \cup \Gamma$ be a padding symbol, and let $\Sigma_\# := \Sigma \cup \{\#\}$ and $\Gamma_\# := \Gamma \cup \{\#\}$. The *convolution* of two words $u = a_1 \ldots a_k \in \Sigma^*$ and $w = b_1 \ldots b_l \in \Gamma^*$, denoted $\left[\begin{smallmatrix} u \\ w \end{smallmatrix}\right]$, is the word over the alphabet $\Sigma_\# \times \Gamma_\#$ defined as follows. Intuitively, $\left[\begin{smallmatrix} u \\ w \end{smallmatrix}\right]$ is the result of putting u on top of w, aligned left, and padding the shorter of u and w with $\#$. Formally, if $k \leq l$, then $\left[\begin{smallmatrix} u \\ w \end{smallmatrix}\right] = \left[\begin{smallmatrix} a_1 \\ b_1 \end{smallmatrix}\right] \cdots \left[\begin{smallmatrix} a_k \\ b_k \end{smallmatrix}\right] \left[\begin{smallmatrix} \# \\ b_{k+1} \end{smallmatrix}\right] \cdots \left[\begin{smallmatrix} \# \\ b_l \end{smallmatrix}\right]$, and otherwise $\left[\begin{smallmatrix} u \\ w \end{smallmatrix}\right] = \left[\begin{smallmatrix} a_1 \\ b_1 \end{smallmatrix}\right] \cdots \left[\begin{smallmatrix} a_l \\ b_l \end{smallmatrix}\right] \left[\begin{smallmatrix} a_{l+1} \\ \# \end{smallmatrix}\right] \cdots \left[\begin{smallmatrix} a_k \\ \# \end{smallmatrix}\right]$. The convolution of a tuple of words $u_1 \in \Sigma_1^*, \ldots, u_k \in \Sigma_k^*$ is defined analogously, putting all k words on top of each other, aligned left, and padding the shorter words with $\#$.

A *transducer* over $\Sigma \times \Gamma$ is an NFA over $\Sigma_\# \times \Gamma_\#$. The binary relation recognised by a transducer T over $\Sigma \times \Gamma$, denoted $\mathsf{R}(T)$, is the set of pairs $(u, w) \in \Sigma^* \times \Gamma^*$ such that T accepts $\left[\begin{smallmatrix} u \\ w \end{smallmatrix}\right]$. The definition is generalised to relations of higher arity in the obvious way. In the paper transducers recognise binary relations unless mentioned otherwise. A relation is *regular* if it is recognised by some transducer. A transducer is *length-preserving* if it recognises a length-preserving relation.

Complexity of operations on automata and transducers. Given NFAs A_1, A_2 over Σ with n_1 and n_2 states, DFAs B_1, B_2 over Σ with m_1 and m_2 states, and transducers T_1 over $\Sigma \times \Gamma$ and T_2 over $\Gamma \times \Sigma$ with l_1 and l_2 states, the following facts are well known (see e.g. chapters 3 and 5 of [18]):

- there exist NFAs for $\mathsf{L}(A_1) \cup \mathsf{L}(A_2)$, $\mathsf{L}(A_1) \cap \mathsf{L}(A_2)$, and $\overline{\mathsf{L}(A_1)}$ with at most $n_1 + n_2$, $n_1 n_2$, and 2^{n_1} states, respectively;
- there exist DFAs for $\mathsf{L}(B_1) \cup \mathsf{L}(B_2)$, $\mathsf{L}(B_1) \cap \mathsf{L}(B_2)$, and $\overline{\mathsf{L}(B_1)}$ with at most $m_1 m_2$, $m_1 m_2$, and m_1 states, respectively;
- there exist NFAs for $\mathsf{R}(T_1)|_1$ and $\mathsf{R}(T_1)|_2$ and a transducer for $\mathsf{R}(T_1)^{-1}$ with at most l_1 states;
- there exists a transducer for $\mathsf{R}(T_1) \circ \mathsf{R}(T_2)$ with at most $l_1 l_2$ states; and
- there exist NFAs for $\mathsf{L}(A_1) \circ \mathsf{R}(T_1)$ and $\mathsf{R}(T_1) \circ \mathsf{L}(A_2)$ with at most $n_1 l_1$ and $l_1 n_2$ states, respectively.

2.1 Regular transition systems

We recall standard notions about regular transition systems and fix some notations. A *transition system* is a pair $\mathcal{S} = (\mathcal{C}, \Delta)$ where $\mathcal{C}$ is the set of all possible *configurations* of the system, and $\Delta \subseteq \mathcal{C} \times \mathcal{C}$ is a *transition relation*. The *reachability relation Reach* is the reflexive and transitive closure of Δ. Observe that, by our definition of post-set, $\Delta(C)$ and $\mathit{Reach}(C)$ are the sets of configurations reachable in one step and in arbitrarily many steps from C, respectively.

Regular transition systems are transition systems where Δ can be finitely represented by a transducer. Formally:

▶ **Definition 1.** *A transition system $\mathcal{S} = (\mathcal{C}, \Delta)$ is* regular *if $\mathcal{C}$ is a regular language over some alphabet Σ, and Δ is a regular relation. We abbreviate regular transition system to RTS.*

RTSs are often used to model parameterised systems [5, 1, 6, 2]. In this case, Σ is the set of possible *states* of a process, the set of configurations is $\mathcal{C} = \Sigma^* \setminus \{\varepsilon\}$, and a configuration $a_1 \cdots a_n \in \Sigma^*$ describes the global state of an *array* consisting of n identical copies of the process, with the i-th process in state a_i for every $1 \le i \le n$. The transition relation Δ describes the possible transitions of all arrays, of any length.

▶ **Example 2** (Token passing [5])**.** We use a version of the well-known token passing algorithm as running example. We have an array of processes of arbitrary length. At each moment in time, a process either has a token (t) or not (n). Initially, only the first process has a token. A process that has a token can pass it to the process to the right if that process does not have one. We set $\Sigma = \{t, n\}$, and so $\mathcal{C} = \{t, n\}^* \setminus \{\epsilon\}$. We have $c_2 \in \Delta(c_1)$ iff the word $\left[\begin{smallmatrix} c_1 \\ c_2 \end{smallmatrix}\right]$ belongs to the regular expression $\left(\left[\begin{smallmatrix} n \\ n \end{smallmatrix}\right] + \left[\begin{smallmatrix} t \\ t \end{smallmatrix}\right]\right)^* \left(\left[\begin{smallmatrix} t \\ n \end{smallmatrix}\right]\left[\begin{smallmatrix} n \\ t \end{smallmatrix}\right]\right) \left(\left[\begin{smallmatrix} n \\ n \end{smallmatrix}\right] + \left[\begin{smallmatrix} t \\ t \end{smallmatrix}\right]\right)^*$. For the set of initial configurations $C_I := tn^*$ where only the first processs has a token, the set of reachable configurations is $\mathit{Reach}(C_I) = n^*\, t\, n^*$.

3 Regular abstraction frameworks

In the same way that RTSs can model multiple classes of *systems* (e.g. parameterised systems with synchronous/asynchronous, binary/multiway/broadcast communication), regular abstraction frameworks are a formalism to model a wide range of *abstractions*.

▶ **Definition 3.** *An* abstraction framework *is a triple* $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$*, where* $\mathcal{C}$ *is a set of* configurations*,* $\mathcal{A}$ *is a set of* constraints*, and* $\mathcal{V} \subseteq \mathcal{A} \times \mathcal{C}$ *is an* interpretation*.* $\mathcal{F}$ *is* regular *if* $\mathcal{C}$ *and* $\mathcal{A}$ *are regular languages over alphabets* Σ *and* Γ*, respectively, and the interpretation* $\mathcal{V}$ *is a regular relation over* $\mathcal{A} \times \mathcal{C}$*.*

Intuitively, the constraints of an abstraction framework are the abstract objects of the abstraction, and $\mathcal{V}(A)$ is the set of configurations abstracted by A. The following remark formalises this.

▶ Remark 4. An abstraction framework $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$ induces an abstract interpretation as follows. The concrete and abstract domains are $(2^{\mathcal{C}}, \leq_{\mathcal{C}})$ and $(2^{\mathcal{A}}, \leq_{\mathcal{A}})$, respectively, where $\leq_{\mathcal{C}} := \subseteq$ and $\leq_{\mathcal{A}} := \supseteq$. Both are complete lattices. The concretisation function $\gamma \colon 2^{\mathcal{A}} \to 2^{\mathcal{C}}$ and the abstraction function $\alpha \colon 2^{\mathcal{C}} \to 2^{\mathcal{A}}$ are given by:

- $\gamma(\mathcal{A}') := \bigcap_{A \in \mathcal{A}'} \mathcal{V}(A)$. Intuitively, $\gamma(\mathcal{A}')$ is the set of configurations that satisfy all constraints of $\mathcal{A}'$. In particular, $\gamma(\emptyset) = \mathcal{C}$.
- $\alpha(\mathcal{C}') := \{A \in \mathcal{A} \mid \mathcal{C}' \subseteq \mathcal{V}(A)\}$. Intuitively, $\alpha(\mathcal{C}')$ is the set of constraints satisfied by all configurations in $\mathcal{C}'$. In particular, $\alpha(\emptyset) = \mathcal{A}$.

It is easy to see that the functions α and γ form a Galois connection, that is, for all $C \subseteq \mathcal{C}$ and $A \subseteq \mathcal{A}$, we have $\mathcal{B} \subseteq \alpha(C) \Leftrightarrow C \subseteq \gamma(\mathcal{B})$.

Regular abstractions can be combined to yield more precise ones. Given abstraction frameworks $\mathcal{F}_1 = (\mathcal{C}, \mathcal{A}_1, \mathcal{V}_1)$ and $\mathcal{F}_2 = (\mathcal{C}, \mathcal{A}_2, \mathcal{V}_2)$, we can define new frameworks $(\mathcal{C}, \mathcal{A}, \mathcal{V})$ by means of the following operations:

- Union: $\mathcal{A} := \mathcal{A}_1 \cup \mathcal{A}_2$, $\mathcal{V}(A) := \mathcal{V}_1(A)$ if $A \in \mathcal{A}_1$, else $\mathcal{V}_2(A)$.
  A constraint of the union framework is either a constraint of the first framework, or a constraint of the second.
- Convolution: $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$, $\mathcal{V}(A_1, A_2) := \mathcal{V}_1(A_1) \cap \mathcal{V}_2(A_2)$.
  A constraint of the convolution framework is the conjunction of two constraints, one of each framework. This operation is implicitly used in [19]: the constraint for a Boolean formula with b clauses is the convolution, applied b times, of the constraints for formulas with one clause.

The proof of the following Lemma is in Appendix A.

▶ **Lemma 5.** *Regular abstraction frameworks are closed under union and convolution. If the interpretations of the frameworks are recognised by transducers with* n_1 *and* n_2 *states, then the interpretations of the union and convolution frameworks are recognised by transducers with* $O(n_1 + n_2)$ *and* $O(n_1 n_2)$ *states, respectively.*

Many abstractions used in the literature can be modeled as regular abstraction frameworks. We give some examples.

▶ **Example 6.** Consider a transition system where $\mathcal{C} = \mathbb{N}^d$ for some d, and Δ is given by a formula of Presburger arithmetic $\delta(\mathbf{x}, \mathbf{x}')$, that is, $(\mathbf{n}, \mathbf{n}') \in \Delta$ iff $\delta(\mathbf{n}, \mathbf{n}')$ holds. It is well-known that for any Presburger formula there is a transducer recognising the set of its solutions when numbers are encoded in binary, and so with this encoding $(\mathcal{C}, \Delta)$ is an RTS. Any Presburger formula $\varphi(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ has dimension d and $\mathbf{y}$ has some arbitrary dimension e, induces a regular abstraction framework as follows. The set of constraints is the set of all tuples $\mathbf{m} \in \mathbb{N}^e$; the interpretation assigns to $\mathbf{m}$ all tuples $\mathbf{n}$ such that $\varphi(\mathbf{n}, \mathbf{m})$ holds. Intuitively, the constraints are the formulas $\varphi_{\mathbf{m}}(\mathbf{x}) := \varphi(\mathbf{x}, \mathbf{m})$, but using $\mathbf{m}$ as encoding of $\varphi_{\mathbf{m}}$.

Special cases of this setting are used in many different areas. For example, bounded difference matrices (see e.g. [27, 8]) are the abstraction framework where the constraints φ

are of the form $x_1 - x_2 \leq y$, and octagons [29] are the abstraction framework with constraints of the form $x_1 - x_2 \leq y$ or $x \leq y$.

▶ **Example 7.** The approach to regular model checking of [19] is another instance of a regular abstraction framework. The paper encodes sets of configurations as positive Boolean formulas in conjunctive normal form with a bounded number b of clauses. We explain this by means of an example. Consider an RTS with $\Sigma = \{a, b, c\}$ and $\mathcal{C} = \Sigma^*$. Consider the formula $\varphi = (a_{1:5} \vee b_{1:5} \vee a_{3:5}) \wedge b_{4:5}$. We interpret φ on configurations. The intended meaning of a literal, say $a_{1:5}$, is "if the configuration has length 5, then its first letter is an a." So the set of configurations satisfying the formula is $\Sigma^{\leq 4} + \Sigma^6\Sigma^* + (a+b)\Sigma^2 b\Sigma + \Sigma^2 ab\Sigma$. In the formulas of [19] all literals have the same length, where the length of a literal $x_{i:j}$ is j.

Formulas with at most b clauses can be encoded as words over the alphabet $\Gamma = (2^\Sigma)^b$. Each clause is encoded as a word over 2^Σ. For example, the encodings of the clauses $(a_{1:5} \vee b_{1:5} \vee a_{3:5})$ and $b_{4:5}$ are $\{a, b\}\emptyset\{a\}\emptyset\emptyset$ and $\emptyset\emptyset\emptyset\{b\}\emptyset$, and the encoding of φ is the convolution of the encodings of the clauses. It is easy to see that the interpretation of [19] that assigns to a formula the set of configurations satisfying it is a regular relation recognised by a transducer with 2^b states [19]. In particular, for the case $b = 1$ we get the two-state transducer on the left of Figure 1.

▶ **Example 8.** In [3] Abdulla et al. introduce *view abstraction* for the verification of parameterised systems. Given a number $k \geq 1$, a *view* of a word $w \in \Sigma^*$ is a scattered subword of w. Loosely speaking, Abdulla et al. abstract a word by its set of views of length up to k. In our setting, a constraint is a set $F \subseteq \Sigma^{\leq k}$ of "forbidden views", and $\mathcal{V}(V)$ is the set of all words that do not contain any view of F. Since k is fixed, this interpretation is regular.

3.1 The abstract safety problem

We apply regular abstraction frameworks to the problem of deciding whether an RTS avoids some regular set of unsafe configurations. For simplicity, we assume w.l.o.g. that the set of configurations of the RTS is Σ^*. (Any RTS can be transformed into an equivalent one with the same transitions where the set of configurations is Σ^*.) Let us first formalise the safety problem.

<u>Safety</u>

Given: a nondeterministic transducer recognising a regular relation $\Delta \subseteq \Sigma^* \times \Sigma^*$, and two NFAs recognising regular sets $C_I, C_U \subseteq \Sigma^*$ of initial and unsafe configurations, respectively.

Decide: does $Reach(C_I) \cap C_U = \emptyset$ hold?

It is a folklore result that Safety is undecidable. Let us sketch the argument. The configurations of a given Turing machine can be encoded as words of the form $w_l\, q\, w_r$, where w_l, w_r encode the contents of the tape to the left and to the right of the head, and q encodes the current state. With this encoding, the successor relation between configurations of the Turing machine is regular, and so is the set of accepting configurations. Taking the latter as set of unsafe configurations, the Turing machine accepts a given initial configuration iff the RTS started at the initial configuration is unsafe.

AbstractSafety. We show that regular abstraction framework induces an "abstract" version of the safety problem, in which we replace the reachability relation by an overapproximation derived from the abstraction framework. Fix an RTS $\mathcal{S} = (\mathcal{C}, \Delta)$ and a regular abstraction framework $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$. We introduce some definitions:

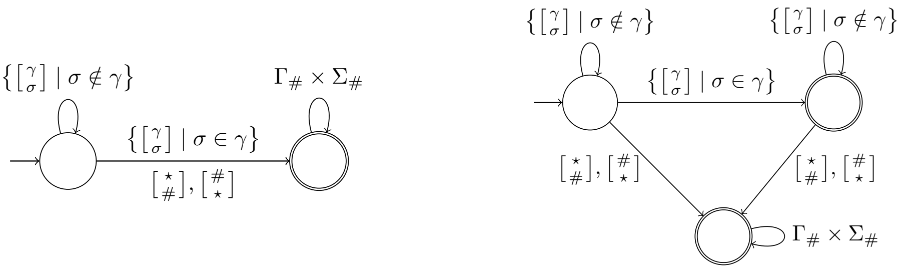

Figure 1 Transducers for the interpretations of Example 7 and 10. We have $\Gamma = 2^\Sigma$, and so the alphabet of the transducer is $(2^\Sigma)_\# \times \Sigma_\#$. The symbols $\left[\begin{smallmatrix}\star\\ \#\end{smallmatrix}\right]$ and $\left[\begin{smallmatrix}\#\\ \star\end{smallmatrix}\right]$ stand for the sets of all letters of the form $\left[\begin{smallmatrix}\gamma\\ \#\end{smallmatrix}\right]$ and $\left[\begin{smallmatrix}\#\\ \sigma\end{smallmatrix}\right]$, respectively .

▶ **Definition 9.** *A set $C \subseteq \mathcal{C}$ of configurations is* inductive *if $\Delta(C) \subseteq C$. A constraint A is* inductive *if $\mathcal{V}(A)$ is inductive. We let $Ind \subseteq \mathcal{A}$ denote the set of all inductive constraints of $\mathcal{A}$. Given two configurations c, c' and $A \in Ind$, we say that A* separates *c from c' if $c \in \mathcal{V}(A)$ and $c' \notin \mathcal{V}(A)$.*

It is a folklore result that $Reach(C)$ is the smallest inductive set containing C, and that if some $A \in Ind$ separates c and c', then $(c, c') \notin Reach$. Hence, an abstraction framework $(\mathcal{C}, \mathcal{A}, \mathcal{V})$ induces a *potential reachability* relation $PReach \subseteq \mathcal{C} \times \mathcal{C}$, defined as the set of all pairs of configurations that are not separated by any inductive constraint. Formally:

$$PReach := \{(c, c') \in \mathcal{C} \times \mathcal{C} \mid \forall A \in Ind\colon c \in \mathcal{V}(A) \to c' \in \mathcal{V}(A)\}$$

We have $Reach(C) \subseteq PReach(C)$ for every set of configurations C. In particular, given sets $C_I, C_U \subseteq \mathcal{C}$ of initial and unsafe configurations, if $PReach(C_I) \cap C_U = \emptyset$, then the RTS is safe.

▶ **Example 10.** Consider the RTS of the token passing system of Example 2, where $\Sigma = \{t, n\}$. We give two examples of abstraction frameworks. The first one is the abstraction framework of [19], already presented in Example 7, with $b = 1$. We have $\Gamma = 2^\Sigma = \{\emptyset, \{t\}, \{n\}, \Sigma\}$. A constraint like $\varphi = \bigvee_{i=3}^{5} t_{i:5}$ is encoded by the word $\emptyset\emptyset\{t\}\{t\}\{t\} \in \Gamma^*$, and interpreted as the set of all configurations of length 5 that have a token at positions 3, 4, or 5, plus the set of all configurations of length different from 5. The two-state transducer for this interpretation is on the left of Figure 1. For example, the left state has transitions leading to itself for the letters $\left[\begin{smallmatrix}\emptyset\\ n\end{smallmatrix}\right], \left[\begin{smallmatrix}\emptyset\\ t\end{smallmatrix}\right], \left[\begin{smallmatrix}\{t\}\\ n\end{smallmatrix}\right], \left[\begin{smallmatrix}\{n\}\\ t\end{smallmatrix}\right]$. The constraint φ is inductive. In fact, the language of all non-trivial inductive constraints (a constraint is trivial if it is satisfied by all configurations or by none) is $\{n\}^+\emptyset^*\{t\}^* + \{n\}^*\emptyset^*\{t\}^+$. The set of configurations potentially reachable from $C_I = tn^*$ is $PReach(C_I) = (tn + nn^*t)(t + n)^*$. In particular, $PReach(C_I) \cap n^* = \emptyset$, but $tnt \in PReach(C_I)$. So this abstraction framework is strong enough to prove that every reachable configuration has at least one token, but not to prove that it has exactly one.

Consider now the framework in which, instead of a *disjunction* of literals, a constraint is an *exclusive disjunction* of literals, that is, a configuration satisfies the constraint if it satisfies *exactly one* of its literals. So, in particular, the interpretation of $\emptyset\emptyset\{t\}\{t\}\{t\}$ is now that exactly one of the positions 3, 4, and 5 has a token. The interpretation is also regular; it is given by the three-state transducer on the right of Figure 1. Examples of inductive constraints are $\{t\}\emptyset\{t, n\}\{n\}$ and all words of $\{t\}^*$. The language of non-trivial inductive constraints is given by the DFA on the left of Figure 2. Observe that the set

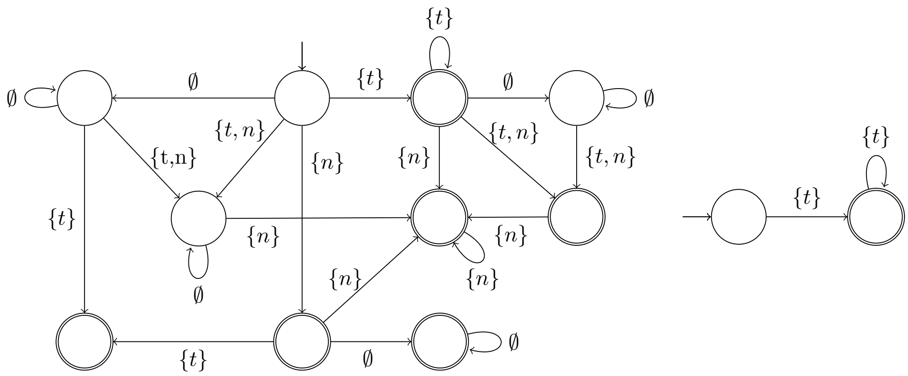

Figure 2 On the left, DFA recognising all non-trivial inductive constraints of Example 17. On the right, fragment with the same interpretation as the DFA on the left.

of words satisfying all constraints of $\{t\}^*$ is the language n^*tn^*. In particular, we have $PReach(C_I) \subseteq n^*tn^* = Reach(C_I)$, and so $PReach(C_I) = Reach(C_I)$. ◀

The abstract safety problem is defined exactly as the safety problem, just replacing the reachability set $Reach(C_I)$ by the potential reachability set $PReach(C_I)$ implicitly defined by the regular abstraction framework:

ABSTRACTSAFETY

Given: a nondeterministic transducer recognising a regular relation $\Delta \subseteq \Sigma^* \times \Sigma^*$; two NFAs recognising regular sets $C_I, C_U \subseteq \Sigma^*$ of initial and unsafe configurations, respectively; and a deterministic transducer recognising a regular interpretation $\mathcal{V}$ over $\Gamma \times \Sigma$.

Decide: does $PReach(C_I) \cap C_U = \emptyset$ hold?

Recall that SAFETY is undecidable. In the rest of this section and in the next one we show that ABSTRACTSAFETY is EXPSPACE-complete. Membership in EXPSPACE was essentially proved in [20], while EXPSPACE-hardness, which is highly non-trivial, was left open. We briefly summarise the proof of membership in EXPSPACE presented in [20], for future reference in our paper.

▶ Remark 11. The result we prove in Section 3.2 is slightly more general. In [20], membership in EXPSPACE is only proved for RTSs whose transducers are length-preserving, while we prove it in general. General transducers allow one to model parameterised systems with process creation. For example, we can model a token passing algorithm in which the size of the array can dynamically grow and shrink by adding the transitions $\left(\left[\begin{smallmatrix} n \\ n \end{smallmatrix}\right] + \left[\begin{smallmatrix} t \\ t \end{smallmatrix}\right]\right)^+ \left(\left[\begin{smallmatrix} n \\ \# \end{smallmatrix}\right] + \left[\begin{smallmatrix} \# \\ n \end{smallmatrix}\right]\right)$ to the transition relation of Example 2.

3.2 AbstractSafety is in EXPSPACE

We first show that the set of all inductive constraints of a regular abstraction framework is a regular language. Fix a regular abstraction framework $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$ over an RTS $(\mathcal{C}, \Delta)$. Let n_Δ, $n_\mathcal{V}$, n_I, n_U be the number of states of the transducers and NFAs of a given instance of ABSTRACTSAFETY.

▶ **Lemma 12.** *[20] The set $\overline{Ind}$ is regular. Further, one can compute an NFA with at most $n_\Delta \cdot n_\mathcal{V}^2$ states recognising $\overline{Ind}$, and a DFA with at most $2^{n_\Delta \cdot n_\mathcal{V}^2}$ states recognising Ind.*

Proof. By definition, we have

$$\begin{aligned}\overline{Ind} &= \{A \in \Gamma^* \mid \exists c, c' \in \mathcal{C}\colon c \in \mathcal{V}(A), c' \in \Delta(c), \text{ and } c' \notin \mathcal{V}(A)\} \\ &= \{A \in \Gamma^* \mid \exists c, c' \in \mathcal{C}\colon (A, c) \in \mathcal{V}, (c, c') \in \Delta \text{ and } (c', A) \in \overline{\mathcal{V}^{-1}}\}\end{aligned}$$

Let $Id_\Gamma = \{(A, A) \mid A \in \Gamma^*\}$. We obtain $\overline{Ind} = \left(\left(\mathcal{V} \circ \Delta \circ \overline{\mathcal{V}^{-1}}\right) \cap Id_\Gamma \right) |_1$. By the results at the end of Section 2, $\overline{Ind}$ is recognised by a NFA with $n_\mathcal{V} \cdot n_\Delta \cdot n_\mathcal{V} = n_\Delta n_\mathcal{V}^2$ states, and so Ind is recognised by a DFA with $2^{n_\Delta \cdot n_\mathcal{V}^2}$ states. ◀

▶ **Lemma 13.** *[20] The potential reachability relation PReach is regular. Further, one can compute a nondeterministic transducer with at most $K := n_\mathcal{V}^2 \cdot 2^{n_\Delta \cdot n_\mathcal{V}^2}$ states recognising $\overline{PReach}$, and a deterministic transducer with at most 2^K states recognising PReach.*

Proof. By definition, we have

$$\begin{aligned}\overline{PReach} &= \{(c, c') \in \mathcal{C} \times \mathcal{C} \mid \exists A \in Ind\colon c \in \mathcal{V}(A) \text{ and } c' \notin \mathcal{V}(A)\} \\ &= \left\{(c, c') \in \mathcal{C} \times \mathcal{C} \mid \exists A \in Ind\colon (c, A) \in \mathcal{V}^{-1} \text{ and } (A, c') \in \overline{\mathcal{V}}\right\}\end{aligned}$$

Let $Id_\Gamma = \{(A, A) \mid A \in \Gamma^*\}$. We obtain $\overline{PReach} = \left(\mathcal{V}^{-1} \circ (Id_\Gamma \cap Ind) \circ \overline{\mathcal{V}}\right)$. Apply now the results at the end of Section 2 and Lemma 12. ◀

▶ Remark 14. The fact that Ind and $PReach$ are regular is also a direct consequence of their definitions and elementary results on automatic structures [26, 10, 21]. The lemmas above also bound the size of their DFA representations.

▶ **Theorem 15.** *[20] AbstractSafety is in* EXPSPACE.

Proof. By Lemma 13, one can effectively compute an NFA for $PReach(C_I) = C_I \circ PReach$ with $2^K \cdot n_I$ states, and so an NFA for $PReach(C_I) \cap C_U$ with $2^K \cdot n_I \cdot n_U$ states. We give a nondeterministic algorithm for the abstract safety problem. The algorithm guesses step by step a run of this NFA leading to an accepting state. The memory required is the memory needed to store one state of the NFA. Since the DFA has $2^K \cdot n_I \cdot n_U$ states, the memory needed is $O(K + \log n_I + \log n_U)$ which is exponential in the size of the input. So the abstract safety problem is in NEXPSPACE = EXPSPACE. ◀

4 AbstractSafety is EXPSPACE-hard

In [19] it was shown that AbstractSafety was PSPACE-hard, and the paper left the question of closing the gap between the upper and lower bounds open. We first recall and slightly alter the PSPACE-hardness proof of [19], and then present our techniques to extend it to EXPSPACE-hardness.

The proof is by reduction from the problem of deciding whether a Turing machine $\mathcal{M}$ of size n does not accept when started on the empty tape of size n. (For technical reasons, we actually assume that the tape has $n - 2$ cells.) Given $\mathcal{M}$, we construct in polynomial time an RTS $\mathcal{S}$ and a set of initial configurations C_I that, loosely speaking, satisfy the following two properties: the execution of $\mathcal{S}$ on an initial configuration simulates the run of $\mathcal{M}$ on the empty tape, and $PReach(C_I) = Reach(C_I)$. We choose C_U as the set of configurations of $\mathcal{S}$ in which $\mathcal{M}$ ends up in the accepting state. Then $\mathcal{S}$ is safe iff $\mathcal{M}$ does not accept.

Turing machine preliminaries. We assume that $\mathcal{M}$ is a deterministic Turing machine with states Q, tape alphabet Γ', initial state q_0 and accepting state q_f.

We represent a configuration of $\mathcal{M}$ as a word $\#\,\beta\,q\,\eta$ of length n, where $\mathcal{M}$ is in state q, the content of the tape is $\beta\,\eta \in \Gamma'^*$, and the head of $\mathcal{M}$ is positioned at the first letter of η. The symbol $\#$ serves as a separator between different configurations. The initial configuration is $\alpha_0 := \#q_0B^{n-2}$, where B denotes the blank symbol of $\mathcal{M}$; so the tape is initially empty.

We assume w.l.o.g. that the successor of a configuration in state q_f is the configuration itself, so the run of $\mathcal{M}$ can be encoded as an infinite word $\alpha' := \alpha_0\alpha_1\cdots$ where α_i represents the i-th configuration of $\mathcal{M}$. For convenience, we write $\Lambda := Q \cup \Gamma' \cup \{\#\}$ for the set of symbols in α'. It is easy to see that the symbol at position $i+n$ of α' is completely determined by the symbols at positions $i-1$ to $i+2$ and the transition relation of $\mathcal{M}$. We let $\delta(x_1x_2x_3x_4)$ denote the symbol which "should" appear at position $i+n$ when the symbols at positions $i-1$ to $i+2$ are $x_1x_2x_3x_4$; in particular, $\delta(x_1\#x_2x_4) = \#$.

Configurations of $\mathcal{S}$. We choose the set of configurations as $\mathcal{C} := \alpha_0\#(\Lambda \cup \{\Box\})^*$, and the initial configurations as $C_I := \alpha_0\#\Box^*$. Intuitively, the RTS starts with the representation of the initial configuration of $\mathcal{M}$, followed by some number of blank cells $\Box$. During its execution, the RTS will "write" the run of $\mathcal{M}$ into these blanks.

A configuration is unsafe if it contains some occurrence of q_f, the accepting state of $\mathcal{M}$, so $C_U := (\Lambda \cup \{\Box\})^*q_f(\Lambda \cup \{\Box\})^*$.

Transitions. For convenience, we will denote the i-th position of a word w as $w(i)$ instead of w_i. Given a configuration c, the set $\Delta(c)$ contains one single configuration c', defined as follows. Let i be some position of c such that $c_{i+n} = \Box$. Then c' coincides with c everywhere except at position $i+n$, where instead $c'(i+n) := \delta(c(i-1)c(i)c(i+1)c(i+2))$. It is easy to see that Δ is a regular relation: The transducer nondeterministically guesses the position $i-1$, reads the next four symbols, say $x_1...x_4$, stores $\delta(x_1...x_4)$ in its state, moves to position $i+n$, checks if $c(i+n) = \Box$ and writes $c'(i+n) := \delta(x_1...x_4)$. The transducer has $O(n^2)$ states.

It follows from the definitions above that $\mathcal{M}$ accepts the empty word iff $\mathcal{S}$ can reach C_U from C_I, i.e. $\mathit{Reach}(C_I) \cap C_U \neq \emptyset$.

Regular abstraction framework. We define a regular abstraction framework $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$ of polynomial size such that $\mathit{PReach}(C_I) = \mathit{Reach}(C_I)$. Hence, for every configuration $c \notin \mathit{Reach}(C_I)$, we must find an inductive constraint $A \in \mathcal{A}$ which separates C_I and c. (Note that C_I contains exactly one configuration of length $|c|$.)

As the reachable configurations are precisely the prefixes of α' with some symbols replaced by $\Box$, there is a position i s.t. $c(i) \notin \{\Box, \alpha'(i)\}$. Let us fix the smallest such i. As we noted above, $\alpha'(i)$ is determined entirely by $\alpha'(i-n-1)...\alpha'(i-n+2)$ via the mapping δ. So the constraint "if $c(i-n-1)...c(i-n+2) = x_1...x_4$, then $c(i) \in \{\Box, \delta(x_1...x_4)\}$" is inductive and separates C_I and c.

Therefore, it is sufficient to define an abstraction framework in which every constraint of the above form can be expressed. This is relatively straightforward. We set $\mathcal{A} := \Box^*\Lambda^4\Box^*\Lambda\Box^*$. Given a constraint $A = \Box^ix_1...x_4\Box^jx\Box^k$, define $\mathcal{V}(A)$ as the set of all configurations c s.t. $c(i+1)...c(i+4) = x_1...x_4$ implies $c(i+j+5) \in \{\Box, x\}$. Clearly, $\mathcal{V}$ is a regular relation which can be recognised by a transducer with 3 states.

▶ **Theorem 16.** *[19] The abstract safety problem is* PSPACE*-hard, even for regular abstraction frameworks where the transducer for the interpretation has a constant number of states.*

4.1 From PSPACE-hardness to EXPSPACE-hardness

In order to prove EXPSPACE-hardness, we start with a machine $\mathcal{M}$ of size n and run it on a tape with 2^n cells. However, if we proceed exactly as in the PSPACE-hardness proof, we encounter two obstacles: (1) The length of α_0 is 2^n, so our definitions of $\mathcal{C}$ and C_I require automata of exponential size. (2) The transducer for the transition relation Δ needs to "count" to 2^n, as this is the distance between the corresponding symbols of α_i and α_{i+1}. Again, this requires an exponential number of states.

Obstacle (1) will be easy to overcome. Essentially, instead of starting the RTS with the entire initial configuration α_0 of $\mathcal{M}$ already in place, we set $C_I := \#\, q_0\, \Box^*$ and modify the transitions of $\mathcal{S}$ to also write out α_0.

However, obstacle (2) poses a more fundamental problem. On its face, it is easy to construct an RTS that can count to 2^n by executing multiple transitions in sequence, e.g. by implementing a binary counter. However, we need to balance this with the needs of the constraint system: if the RTS is too sophisticated, our constraints can no longer capture its behaviour using only regular languages.

We will now sketch an RTS $\mathcal{S}'$ which extends $\mathcal{S}$ from above.

A two-phase system. Our RTS will use a "mark and write" approach. In a first phase, it will execute n transitions to mark positions with distance m, where $m \geq 2^n$ is some fixed constant. Then, it nondeterministically guesses a marked position, reads and stores 4 symbols from that position, and moves to the next marker to write the symbol according to δ.

Let $p_1, \dots, p_n$ be the first n prime numbers (i.e. $p_1 = 2, p_2 = 3$, etc.). Define $m := \prod_{j=1}^{n} p_j$ and $s := \sum_{j=1}^{n} p_i$. We have $m \geq 2^n$ and, by the Prime Number Theorem, $s \in O(n^2 \log n)$.

The configurations of $\mathcal{S}'$ will be of the form $w \left[\begin{smallmatrix} m \\ c \end{smallmatrix}\right]$, where $w \in \{0,1\}^s$ will store the current state of the mark phase, $m \in \{0,1\}^*$ are the markers (0 means marked), and $c \in (\Lambda \cup \{\Box\})^*$ is as for $\mathcal{S}$, with the reachable configurations being the prefixes of α' with some symbols replaced by $\Box$. We refer to $\left[\begin{smallmatrix} m \\ c \end{smallmatrix}\right]$ as the *TM part*.

The RTS will have three kinds of transitions: $\Delta' := \Delta_{mark} \cup \Delta_{write} \cup \Delta_{init}$.

In the mark phase, the RTS will execute a transition of Δ_{mark} for each $j \in [n]$. When executing such a transition, we choose a remainder $r \in [0, p_j - 1]$ and set the corresponding bit in w. We then unmark every position in the TM part which is *not* equivalent to r modulo p_j (by replacing the 0 with a 1). Hence, after executing n transitions in Δ_{mark}, the positions of all 0's in the TM part are equivalent modulo every p_j. By the Chinese remainder theorem, these positions must also be equivalent modulo m.

Afterwards, we execute either a transition in Δ_{write} or Δ_{init}. To execute Δ_{write}, the transducer nondeterministically guesses a marked position i, reads $x := c(i-1)...c(i+2)$, moves to the next marked position i', and writes $\delta(x)$.

As mentioned in obstacle (1) above, the RTS must write out the initial configuration of $\mathcal{M}$. This is done by Δ_{init}. If the first position of the TM part is not marked, the transducer moves to the first marked position and writes B, otherwise it moves to the second marked position and writes $\#$. By executing this transition multiple times, eventually a configuration $w \left[\begin{smallmatrix} m \\ c \end{smallmatrix}\right]$ with $c = \# q_0 B^{m-2} \# \Box^i$ can be reached.

While executing either Δ_{write} or Δ_{init}, the transducer resets the mark phase state and marks all positions, i.e. the resulting configurations have $w = 0^s$ and $m \in 0^*$.

The abstraction framework. Consider a configuration $w\,[{}^{m}_{c}]$. Intuitively, we start with the constraint "$c(i-1)...c(i+2) = x_1...x_4 \Rightarrow c(i+m) = \delta(x_1...x_4)$", similar to before. However, this constraint is no longer inductive! To see this, observe that Δ_{write} only reads from one marked position and writes to the next, it does not (and cannot) check that the markers have distance m. So if we have a configuration where w indicates that the first phase is finished, but the markers have not been set appropriately, then Δ_{write} will read from an incorrect position and write the wrong symbol.

So the constraint must also ensure that positions are marked according to Δ_{mark}. In a single constraint, we are only concerned about one particular position i. Recall that the j-th execution of Δ_{mark} in each mark phase picks a remainder $r_j \in [0, p_j - 1]$. There is a unique sequence of remainders $r_1, ..., r_n$ s.t. position i remains marked.

For $j \in [n]$, let I_j denote the positions i for which there exists an $r_{j'} \in \{r_1, ..., r_j\}$ s.t. i is not equivalent to p_j modulo $r_{j'}$. Let C_j denote the set of configurations fulfilling the constraint "if bits corresponding to $r_1, ..., r_j$ are set in w, then exactly the positions in I_j are unmarked". Essentially, C_j contains the configurations where, if remainders $r_1, ..., r_j$ are picked, all positions which should be unmarked are unmarked.

Finally, let C denote the configurations of the initial constraint, i.e. "$c(i-1)...c(i+2) = x_1...x_4 \Rightarrow c(i+m) = \delta(x_1...x_4)$". We will prove that the set $C \cap C_0 \cap ... \cap C_n$ is inductive, and that we can construct a regular abstraction framework capable of expressing these constraints.

With these constraints, we can then exclude every unsafe configuration. The argument is analogous to $\mathcal{S}$: there must be some position i with a "wrong" symbol, but this would violate C and thus $C \cap C_0 \cap ... \cap C_n$.

For the full proof, see Appendix B.

5 Learning regular sets of inductive constraints

Recall the algorithm for AbstractSafety underlying Theorem 15. It computes an automaton recognising the set *Ind* of inductive constraints (Lemma 12); uses this automaton to compute a transducer recognising the potential reachability relation *PReach* (Lemma 13); uses this transducer to compute an automaton recognising $\mathit{PReach}(C_I) \cap C_U$; and finally uses this automaton to check if $\mathit{PReach}(C_I) \cap C_U$ is empty (Theorem 15). The main practical problem of this approach is that, while the automaton for $\overline{\mathit{Ind}}$ has polynomial size in the input, the automaton for *Ind* can be exponential, and, while the automaton for $\overline{\mathit{PReach}}$ has polynomial size in *Ind*, the size of the automaton for *PReach* can be exponential.

In practice one typically does not need all inductive constraints to prove safety. This can be illustrated even on the tiny RTS of Example 2.

▶ **Example 17.** Consider the RTS of the token passing system of Example 2, where $\Sigma = \{t, n\}$, and the second abstraction framework of Example 10, where $\Gamma = 2^{\Sigma} = \{\emptyset, \{t\}, \{n\}, \{t, n\}\}$. Recall that in this abstraction framework a constraint is an *exclusive disjunction* of literals, that is, a configuration satisfies the constraint if it satisfies *exactly one* of its literals. The minimal DFA recognising all non-trivial inductive constraints was shown on the left of Figure 2. The set of inductive constraints $\{t\}\{t\}^*$ is satisfied by the configurations n^*tn^*, and so the DFA on the right is already strong enough to prove any safety property.

We present a learning algorithm that computes automata recognising increasingly large sets $\mathsf{H} \subseteq \mathit{Ind}$ of inductive constraints until either H is large enough to prove safety, or it becomes clear that even the whole set *Ind* is not large enough. More precisely, recall that, by definition, we have $PReach := \{(c, c') \in \mathcal{C} \times \mathcal{C} \mid \forall A \in Ind\colon c \in \mathcal{V}(A) \to c' \in \mathcal{V}(A)\}$. Given a set $\mathsf{H} \subseteq Ind$, define the relation $PReach_{\mathsf{H}}$ exactly as $PReach$, just replacing Ind by H. Clearly, we have $PReach_{\mathsf{H}} \supseteq PReach$ and $PReach_{Ind} = PReach$. Our algorithm checks for each H whether $PReach_{\mathsf{H}}(C_I) \cap C_U = \emptyset$ holds; if so, then $PReach(C_I) \cap C_U = \emptyset$ holds as well, and the RTS is safe. Otherwise, the algorithm computes a pair $(c, c') \in PReach_{\mathsf{H}} \cap (C_I \times C_U)$, and checks if some inductive constraint separates c and c'. If so, the new constraint is used to construct the next set H. Otherwise, the algorithm returns (c, c'), which witnesses that Ind is not powerful enough to prove safety. We now describe each step of the algorithm in detail.

5.1 The learning algorithm

Let $\mathcal{S} = (\mathcal{C}, \Delta)$ and $\mathcal{F} = (\mathcal{C}, \mathcal{A}, \mathcal{V})$ be a regular transition system and a regular abstraction framework, respectively. Further, let C_I and C_U be regular sets of initial and unsafe configurations. The algorithm refines Angluin's algorithm L^* for learning a DFA for the full set Ind [7, 32]. Recall that Angluin's algorithm involves two agents, usually called Learner and Teacher. Learner sends Teacher membership and equivalence queries, which are answered by Teacher according to the following specification:

Membership Query:
- Input: a constraint $A \in \mathcal{A}$
- Output: ✓ if $A \in Ind$, and $\times$ otherwise.

Equivalence Query:
- Input: a DFA recognising a set $\mathsf{H} \subseteq \mathcal{A}$.
- Output: ✓ if $\mathsf{H} = Ind$, otherwise a constraint $A \in (\mathsf{H} \setminus Ind) \cup (Ind \setminus \mathsf{H})$.

Angluin's algorithm describes a strategy for Learner guaranteeing that Learner eventually learns the minimal DFA recognising Ind. The number of equivalence queries asked by Learner is at most the number of states of the DFA.

Answering the queries. We describe the algorithms used by Teacher to answer queries. For membership queries, Teacher constructs an NFA for $\overline{Ind} \cap \{A\}$ with $O(|A| \cdot n_\Delta \cdot n_{\mathcal{V}}^2)$ states (see Lemma 12), and checks it for emptiness.

For equivalence queries, Teacher proceeds as follows :

1. Teacher first checks whether $\mathsf{H} \setminus Ind \neq \emptyset$ holds by computing an NFA recognising $\mathsf{H} \cap \overline{Ind}$ with $O(n_{\mathsf{H}} \cdot n_\Delta \cdot n_{\mathcal{V}}^2)$ states (see Lemma 12), and checking it for emptiness. If $\mathsf{H} \setminus Ind$ is nonempty, then Teacher returns one of its elements.
2. Otherwise, Teacher constructs an automaton for $PReach_{\mathsf{H}}(C_I) \cap C_U$ of size $O(2^{n_{\mathcal{V}}^2 \cdot n_{\mathsf{H}}})$ and checks it for emptiness. There are two cases:
   a. If $PReach_{\mathsf{H}}(C_I) \cap C_U = \emptyset$, then the system is safe; Teacher reports it and terminates. In this case, the learning algorithm is aborted without having learned a DFA for Ind, because it is no longer necessary.
   b. Otherwise, Teacher chooses an element $(c, c') \in PReach_{\mathsf{H}} \cap (C_I \times C_U)$, and searches for an inductive constraint A such that $c \in \mathcal{V}(A)$ and $c' \notin \mathcal{V}(A)$. We call this problem the *separability problem*, and analyze it further in Section 5.2.

5.2 The separability problem.

The separability problem is formally defined as follows:

<u>SEPARABILITY</u>

Given: a nondeterministic transducer recognising a regular relation $\Delta \subseteq \Sigma^* \times \Sigma^*$; a deterministic transducer recognising a regular interpretation $\mathcal{V}$ over $\Gamma \times \Sigma$; and two configurations $c, c' \in \mathcal{C}$

Decide: is c' separable from c, that is, does some inductive constraint A satisfy $c \in \mathcal{V}(A)$ and $c' \notin \mathcal{V}(A)$?

Contrary to ABSTRACTSAFETY, the complexity of SEPARABILITY is different for arbitrary transducers, and for length-preserving ones.

▶ **Theorem 18.** *SEPARABILITY is* PSPACE*-complete, even if Δ is length-preserving. If $\mathcal{V}$ is length-preserving, then SEPARABILITY is* NP*-complete.*

Membership in PSPACE and in NP respectively, are easy to see. In both cases we guess a constraint A that separates c from c', and then check that $c \in \mathcal{V}(A)$, $c' \notin \mathcal{V}(A)$, and that A is inductive, which can be done by checking that $A \notin \overline{Ind}$. Since the automaton for $\mathcal{V}$ is part of the input, and a NFA recognising $\overline{Ind}$ can be constructed in polynomial time, given an input of size n the checks can be carried out in $O(q(n))$ space and $O(|A| \cdot p(n))$ time for some polynomials p and q. This proves SEPARABILITY$\in$ NPSPACE = PSPACE. However, it does not prove SEPARABILITY$\in$ NP because the length of A may not be polynomial in n. However, if Δ and $\mathcal{V}$ are length-preserving, then we necessarily have $|c| = |c'| = |A|$. So the algorithm also runs in polynomial time. The hardness proofs can be found in Appendix C.

Most applications of regular model checking to the verification of parameterised systems, and in particular all the examples studied in [19, 20], have length-preserving transition functions and length-preserving interpretations. For this reason, in our implementation we only consider this case, and leave an extension for future research. Since SEPARABILITY is NP-complete in the length-preserving case, it is natural to solve it by reduction to SAT. As shown above, it can be reduced to checking if there exists a word of polynomial length in the input that is not accepted by a given NFA, also of polynomial length in the input. So, for the reduction to SAT, it suffices to construct a Boolean formula whose satisfying assignments are the words rejected by a given NFA. A brief description is given in Appendix D.

5.3 Some experimental results

We have implemented the learning algorithm in a tool prototype, built on top of the libraries `automatalib` and `learnlib` [23] and the SAT solver `sat4j` [9]. We compare our learning approach with the one of [19], which constructs automata for *Ind* and *PReach* using the regular abstraction framework of Example 7. In the rest of this section we call these two approaches the *lazy* and the *direct* approach, respectively. We use the same case studies as [19]. We compare the sizes of the DFA for the final hypothesis H and $PReach_{\mathsf{H}}$ with the sizes of the DFA for *Ind* and *PReach*. The results are available at [35] and are shown in Table 1.

The left table in Table 1 shows results on RTSs modeling mutex and leader election algorithms, and academic examples, like various versions of the dining philosophers. The right top table shows results on models of cache-coherence protocols. The bottom right table is explained later.

In each table, the first three columns contain the name of the RTS and the sizes of the automata for C_I and Δ. The fourth column (Pr.) indicates the checked property, where D, M, and O stand for "deadlock freedom", "mutex" (at most one process in a given state), and "other" (custom properties of the particular RTS). The next two columns give the results for the lazy approach: sizes of the DFAs for H and $PReach_{\mathsf{H}}$ (abbreviated as PR_{H}), and the next two the same results for the direct approach. The last column (Re.) indicates the result of the check: the property could be proved (✓), could not, (×), or, in the case of multiple properties, how many of the properties were proved (e.g. 2/3 means two out of three). (Observe that the results are identical in Observe that *Ind* and *PReach* do not depend on the property, but H and $PReach_{\mathsf{H}}$, because the algorithm can finish early. In this case, the sizes given in columns H and $PReach_{\mathsf{H}}$ are the largest ones computed over all properties checked.

The main result is that the automata computed by our tool are significantly smaller than those for [19]. (Observe that in all cases we compute minimal DFAs, and so the differences are not due to algorithms for the computation of automata.) Observe that in five cases the deadlock-freedom and the mutex properties could not be proved. In one case (deadlock-freedom of Herman (linear)) this is because the property does not hold. In the other four cases, the problem is that [19] uses only a specific regular abstraction framework, namely the one of Example 7. We can prove the property by refining the abstraction: we take the union of the "disjunctive" and the "exclusive disjunctive" abstractions of Example 10. The bottom-right table gives the results of these four cases.

We do not give detailed results on performance, because both tools take less than three seconds in 54 out of the 59 case studies in the left and top right tables. In the other five cases, the implementation of [19] still needs less than one second, while our implementation takes minutes (more than ten minutes in two cases). In these five cases the time performance is dominated by the SAT solver. We think that future experimentation with other solvers for the separation problem is likely to reduce the time considerably.

				Lazy		Direct		
System	$\lvert C_I \rvert$	$\lvert\Delta\rvert$	Pr.	$\lvert\mathsf{H}\rvert$	$\lvert PR_{\mathsf{H}}\rvert$	$\lvert Ind \rvert$	$\lvert PR \rvert$	Re.
Bakery	3	5	D	1	1	9	8	✓
			M	4	3			✓
Burns	1	6	D	1	1	10	6	✓
			M	5	3			✓
Dijkstra	2	17	D	4	4	218	22	✓
			M	11	8			✓
Dijkstra (ring)	2	12	D	9	9	47	17	✓
			M	9	7			×
Dining crypto.	2	8	D	23	18	86	19	2/2
Herman	2	11	D	3	2	8	7	✓
			O	1	2			✓
Herman (linear)	2	3	D	1	2	7	7	×
			O	1	2			✓
Israeli-Jafon	3	10	D	1	4	21	7	✓
			O	1	4			✓
Token passing	2	3	O	4	4	9	7	✓
Lehmann-Rabin	1	7	D	5	6	29	13	✓
LR phils. 1	1	11	D	13	14	29	15	×
LR phils. 2	1	11	D	25	11	29	9	✓
Atomic phils.	1	8	D	13	9	22	20	✓
Mux array	2	4	D	1	2	7	8	✓
			M	3	5			×
Res. alloc.	1	5	D	5	5	9	8	✓
			M	3	3			×

				Lazy		Direct		
System	$\lvert C_I \rvert$	$\lvert\Delta\rvert$	Pr.	$\lvert\mathsf{H}\rvert$	$\lvert PR_{\mathsf{H}}\rvert$	$\lvert Ind \rvert$	$\lvert PR \rvert$	Re.
Berkeley	1	9	D	1	1	12	9	✓
			O	4	4			2/3
Dragon	1	23	D	1	1	37	11	✓
			O	15	7			6/7
Firefly	1	16	D	1	1	12	7	✓
			O	4	3			0/4
Illinois	1	16	D	1	1	18	14	✓
			O	4	3			0/2
MESI	1	7	D	1	1	8	7	✓
			O	4	4			2/2
MOESI	1	7	D	1	1	15	10	✓
			O	4	4			7/7
Synapse	1	5	D	1	1	8	7	✓
			O	2	3			2/2

				Lazy		Direct		
System	$\lvert C_I \rvert$	$\lvert\Delta\rvert$	Pr.	$\lvert\mathsf{H}\rvert$	$\lvert PR_{\mathsf{H}}\rvert$	$\lvert Ind \rvert$	$\lvert PR \rvert$	Re.
Dijkstra (ring)	2	12	M	9	7			✓
LR phils. 1	1	11	D	34	11			✓
Mux array	2	4	M	5	3			✓
Res. alloc.	1	5	M	5	5			✓

Table 1 Comparison of the sizes of the automata computed by the lazy and direct approaches.

6 Conclusions

We have generalised the technique of [19, 20] for checking safety properties of RTS to arbitrary regular abstraction frameworks. While the safety problem is undecidable for RTSs, the abstract safety problem induced by any abstraction framework is decidable and in EXPSPACE. We have shown that the abstract safety problem is EXPSPACE-complete, solving an open problem of [19, 20], by means of a complex reduction of independent interest. For particular abstraction frameworks the complexity can be better, and an interesting question for future research is to find classes of frameworks, as general as possible, of lower complexity.

Computing an automaton recognising *all* inductive constraints of a given abstraction framework can be an overkill. We have used automata learning to design a lazy algorithm that stops when the inductive constraints computed so far are enough to prove safety. Its combination with other learning techniques, as those proposed in [30, 33, 16, 34, 31], is a question for future research.

References

1 Parosh Aziz Abdulla. Regular model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):109–118, 2012.

2 Parosh Aziz Abdulla. Regular model checking: Evolution and perspectives. In *Model Checking, Synthesis, and Learning*, volume 13030 of *Lecture Notes in Computer Science*, pages 78–96. Springer, 2021.

3 Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. Parameterized verification through view abstraction. *Int. J. Softw. Tools Technol. Transf.*, 18(5):495–516, 2016.

4 Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d'Orso. Regular model checking made simple and efficient. In *CONCUR*, volume 2421 of *Lecture Notes in Computer Science*, pages 116–130. Springer, 2002.

5 Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. A survey of regular model checking. In *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2004.

6 Parosh Aziz Abdulla, A. Prasad Sistla, and Muralidhar Talupur. Model checking parameterized systems. In *Handbook of Model Checking*, pages 685–725. Springer, 2018.

7 Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

8 Clark W. Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

9 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *J. Satisf. Boolean Model. Comput.*, 7(2-3):59–6, 2010.

10 Achim Blumensath and Erich Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.

11 Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large (extended abstract). In *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235. Springer, 2003.

12 Ahmed Bouajjani, Peter Habermehl, Adam Rogalewicz, and Tomás Vojnar. Abstract regular (tree) model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):167–191, 2012.

13 Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.

14 Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

15 Ahmed Bouajjani and Tayssir Touili. Widening techniques for regular tree model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):145–165, 2012.

16 Yu-Fang Chen, Chih-Duo Hong, Anthony W. Lin, and Philipp Rümmer. Learning to prove safety over parameterised concurrent systems. In *FMCAD*, pages 76–83. IEEE, 2017.

17 Dennis Dams, Yassine Lakhnech, and Martin Steffen. Iterating transducers. In *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2001.

18 Javier Esparza and Michael Blondin. *Automata Theory – An Algorithmic Approach*. MIT Press, 2023.

19 Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. Regular model checking upside-down: An invariant-based approach. In *CONCUR*, volume 243 of *LIPIcs*, pages 23:1–23:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

20 Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. Regular model checking upside-down: An invariant-based approach. *CoRR*, abs/2205.03060, version 2, 2022.

21 Erich Grädel. Automatic structures: Twenty years later. In *LICS*, pages 21–34. ACM, 2020.

22 Chih-Duo Hong and Anthony W. Lin. Regular abstractions for array systems. *Proc. ACM Program. Lang.*, 8(POPL):638–666, 2024.

23 Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source learnlib - A framework for active automata learning. In *CAV (1)*, volume 9206 of *Lecture Notes in Computer Science*, pages 487–495. Springer, 2015.

24 Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS*, volume 1785 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2000.

25 Yonit Kesten, Oded Maler, Monica Marcus, Amir Pnueli, and Elad Shahar. Symbolic model checking with rich assertional languages. *Theor. Comput. Sci.*, 256(1-2):93–112, 2001.

26 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *LCC*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer, 1994.

27 Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

28 Axel Legay. Extrapolating (omega-)regular model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):119–143, 2012.

29 Antoine Miné. The octagon abstract domain. *High. Order Symb. Comput.*, 19(1):31–100, 2006.

30 Daniel Neider. *Applications of automata learning in verification and synthesis*. PhD thesis, RWTH Aachen University, 2014.

31 Daniel Neider and Nils Jansen. Regular model checking using solver technologies and automata learning. In *NASA Formal Methods*, volume 7871 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2013.

32 Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In *SFM*, volume 6659 of *Lecture Notes in Computer Science*, pages 256–296. Springer, 2011.

33 Abhay Vardhan. *Learning To Verify Systems*. PhD thesis, University of Illinois, 2006.

34 Abhay Vardhan, Koushik Sen, Mahesh Viswanathan, and Gul Agha. Learning to verify safety properties. In *ICFEM*, volume 3308 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2004.

35 Christoph Welzel-Mohr, Valentin Krasotin, Philipp Czerner, and Javier Esparza. dodo, February 2024. `doi:10.5281/zenodo.10654926`.

A Operations on regular abstraction frameworks

▶ **Lemma 5.** *Regular abstraction frameworks are closed under union and convolution. If the interpretations of the frameworks are recognised by transducers with n_1 and n_2 states, then the interpretations of the union and convolution frameworks are recognised by transducers with $O(n_1 + n_2)$ and $O(n_1 n_2)$ states, respectively.*

Proof. For $i \in \{1,2\}$, let $(Q_i, \Gamma_i \times \Sigma, \delta_i, q_{0i}, F_i)$ be a deterministic transducer for $\mathcal{V}_i$. By the product construction,

$$(Q_1 \times Q_2, (\Gamma_1 \times \Gamma_2) \times \Sigma, \delta_\cap, (q_{01}, q_{02}), F_1 \times F_2)$$

with

$$\delta((q_1, q_2), \begin{bmatrix}(\gamma_1, \gamma_2)\\ a\end{bmatrix}) = (\delta_1(q_1, a), \delta_2(q_2, a))$$

is a deterministic transducer for $\mathcal{V}_\cap$.

For union, assume that Γ_1 and Γ_2 are disjoint. In this case,

$$(Q_1 \cup Q_2 \cup \{q_0, q_\emptyset\}, (\Gamma_1 \cup \Gamma_2) \times \Sigma, \delta_\cup, q_0, F_1 \cup F_2)$$

with

$$\delta_\cup(q, a) := \begin{cases} \delta_i(q, a) & \text{if } q \in Q_i \text{ and } a \in \Gamma_i \times \Sigma \\ \delta_i(q_{0i}, a) & \text{if } q = q_0 \text{ and } a \in \Gamma_i \times \Sigma \\ q_\emptyset & \text{if } q = q_\emptyset \text{ and } a \in (\Gamma_1 \cup \Gamma_2) \times \Sigma \end{cases}$$

is a deterministic transducer for $\mathcal{V}_\cup$. ◀

B AbstractSafety is EXPSPACE-hard

We prove that AbstractSafety is EXPSPACE-hard. We briefly recall and make formally precise the definitions from Section 4.1. Furthermore, for convenience, we will denote the letter in the i-th position of a word w by $w(i)$ instead of w_i.

Turing machine. Let $\mathcal{M}$ denote a deterministic Turing machine with tape alphabet Γ', states Q, initial state q_0, final state q_f, and let $n := |Q|$. We write $p_1, ..., p_n$ for the first n primes, and set $s := \sum_i p_i$ and $m := \prod_i p_i$.

We reduce from the problem of deciding whether $\mathcal{M}$ accepts when run on an empty tape of size $m - 2 \geq 2^n$. (In the following, we shorten this to "$\mathcal{M}$ accepts".)

Configurations of $\mathcal{M}$ are represented as described in Section 4, so they are words of length m over the alphabet $\Lambda := Q \cup \Gamma' \cup \{\#\}$. We write α_i for the representation of the i-th configuration of $\mathcal{M}$.

The infinite word $\alpha' := \alpha_0\alpha_1...$ encodes the run of $\mathcal{M}$ on an empty tape of size $m-2$. From the transitions of $\mathcal{M}$ we obtain a function $\delta : \Lambda^4 \to \Lambda$ s.t. $\alpha'(i + m) = \delta(\alpha'(i - 1)...\alpha'(i + 2))$ for all i. We extend δ to a function $\delta : (\Lambda \cup \{\Box\})^4 \to \Lambda \cup \{\Box\}$ by mapping every word $x \in (\Lambda \cup \{\Box\})^4 \setminus \Lambda^4$ to $\Box$.

▶ **Observation 19.** *$\mathcal{M}$ accepts iff there exists a word $c \in \Lambda^*$ s.t. c begins with $\alpha_0\#$, $c(i+m) = \delta(c(i - 1)...c(i + 2))$ for all $i \geq 2$, and c contains q_f.*

Regular transition system. We define an RTS $\mathcal{S}' = (\mathcal{C}, \Delta)$. The configurations are $\mathcal{C} := \{0,1\}^s(\{0,1\} \times \Lambda)^*$. For a configuration $u = w \begin{bmatrix} m \\ c \end{bmatrix} \in \mathcal{C}$ we refer to w as the *prime part* and $\begin{bmatrix} m \\ c \end{bmatrix}$ as the *TM part*.

We say that prime p_j is *selected* in u if $w(p_1 + ... + p_{j-1} + i) = 1$ for some $i \in [p_j]$. We write $J(u) := \max(\{0\} \cup \{j : p_j \text{ is selected in } u\})$ for the index of the largest selected prime.

Input: $u \in \mathcal{C}$
Output: $v \in \mathcal{C}$ such that $\left[\begin{smallmatrix} u \\ v \end{smallmatrix}\right] \in \Delta_{mark}$
1 $v := u$;
2 $j := J(u) + 1$;
3 pick $r \in [0, p_j - 1]$;
4 $v(p_1 + ... + p_{j-1} + 1 + r) := 1$;
5 **for** $k = 1, 2, ...$ **do**
6   **if** $k \not\equiv r \pmod{p_j}$ **then**
7   unmark $v(s + k)$;

Input: $u \in \mathcal{C}$
Output: $v \in \mathcal{C}$ such that $\left[\begin{smallmatrix} u \\ v \end{smallmatrix}\right] \in \Delta'_{write}$
1 $v := u$;
2 $v(1)...v(s) := 0^s$;
3 pick $i \geq s + 2$ s.t. $u(i)$ is marked;
4 $\left[\begin{smallmatrix} m \\ x \end{smallmatrix}\right] := u(i-1)...u(i+2)$;
5 pick smallest $i' > i$, s.t. $u(i')$ is marked;
6 mark $v(s+1), v(s+2), ...$;
7 write $\delta(x)$ to $v(i')$;

Figure 3 Pseudo-code representation of the transducers for Δ_{mark} and Δ'_{write}.

Within the TM part, we say that a symbol $\left[\begin{smallmatrix} 0 \\ x \end{smallmatrix}\right]$ is *marked*, and $\left[\begin{smallmatrix} 1 \\ x \end{smallmatrix}\right]$ is *unmarked*. One *marks* (or *unmarks*) a symbol $\left[\begin{smallmatrix} b \\ x \end{smallmatrix}\right]$ by replacing it with $\left[\begin{smallmatrix} 0 \\ x \end{smallmatrix}\right]$ (or $\left[\begin{smallmatrix} 1 \\ x \end{smallmatrix}\right]$). To *write* y to $\left[\begin{smallmatrix} b \\ x \end{smallmatrix}\right]$, we replace it with $\left[\begin{smallmatrix} b \\ y \end{smallmatrix}\right]$ if $x = \square$, otherwise we leave it unchanged.

Let us now construct the transitions of $\mathcal{S}'$. As described in Section 4.1, we set $\Delta' := \Delta_{mark} \cup \Delta_{write} \cup \Delta_{init}$.

We start with Δ_{mark}, which is shown in Figure 3. The idea is that we pick some residual r of some prime p_j, and unmark all positions that do *not* have the same residual.

For Δ_{write}, we split the description into two parts: a regular relation Δ'_{write} which implements the modification of the configuration, and a regular language C_{good} describing invariants that should hold while executing Δ'_{write}. The final relation is $\Delta_{write} := \{(u, v) \in \Delta'_{write} : u \in C_{good}\}$. In other words, the RTS only executes a write transition if some consistency conditions are met.

The transducer for Δ'_{write} is given in Figure 3. Informally, the language C_{good} contains the configurations where the prime part has precisely one bit set for every prime. Writing $L_i \subseteq \{0,1\}^i$ for the binary words of length i with exactly one 1, we thus set $C_{good} := L_{p_1} L_{p_2} ... L_{p_n} (\{0,1\} \times \Lambda)^*$.

It remains to describe Δ_{init}. Analogously to Δ_{write}, we define a relation Δ'_{init} and only execute the transition if the input is in C_{good}. The relation is shown in Figure 4; it initialises the tape of $\mathcal{M}$ by replacing the first marked position after $s + 1$ with the appropriate symbol.

Finally, we give the sets of initial and unsafe configurations: $C_I := 0^s \left[\begin{smallmatrix} 0 \\ \# \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 \\ q_0 \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 \\ \square \end{smallmatrix}\right]^*$, $C_U := 0^s (\{0\} \times \Lambda)^* \left[\begin{smallmatrix} 0 \\ q_f \end{smallmatrix}\right] (\{0\} \times \Lambda)^*$.

▶ **Lemma 20.** *If $\mathcal{M}$ accepts, $Reach(C_I) \cap C_U \neq \emptyset$.*

Proof. As $\mathcal{M}$ accepts, there is some l with $\alpha(l) = q_f$. We show that we can move from any configuration $u_i = 0^s \left[\begin{smallmatrix} 0^l \\ c \end{smallmatrix}\right] \in \mathcal{C}$ where $c = \alpha(1)...\alpha(i)\square^{l-i}$ to the configuration u_{i+1}. In other words, we can write the run of $\mathcal{M}$ down one symbol at a time, until we reach q_f.

Clearly, $u_2 \in C_I$ and $u_l \in C_U$, so this suffices to show the lemma.

Fix some i, and let $r_1, ..., r_n$ denote the unique remainders with $r_j \in [0, p_j - 1]$ and $r_j \equiv i + 1 \pmod{p_j}$ for $j \in [n]$. Starting from u_i, we execute Δ_{mark} a total of n times, selecting remainder $r := r_j$ during the j-th execution in line 3. Let u' denote the resulting execution. In u', position $u'(s + i + 1)$ is marked.

We now have two cases. If $i + 1 - m > 1$, then $u'(s + i + 1 - m)$ is marked as well, and we execute Δ_{write} to write $\alpha(i+1) = \delta(\alpha(i-m)...\alpha(i-m+3))$ to $u'(s+i+1)$, which uses $u'(s+i-m)...u'(s+i-m+3) = \left[\begin{smallmatrix} b \\ \alpha(i-m)...\alpha(i-m+3) \end{smallmatrix}\right]$. Otherwise, executing Δ_{init} writes $\alpha(i+1) = \alpha_0(i+1) \in \{B, \#\}$ to $u'(s+i+1)$.

In either case, the prime part is reset and all positions in the TM part are marked, so the resulting configuration is precisely u_{i+1}. ◀

Regular abstraction framework. If $\mathcal{M}$ accepts, no constraint proving safety can exist due to Lemma 20. Consequently, when constructing the abstraction framework we only need to ensure that — provided $\mathcal{M}$ does not accept — for every pair $(u, v) \in C_I \times C_U$ there is an inductive constraint separating u and v.

We now describe the abstraction framework $(\mathcal{C}, \mathcal{A}, \mathcal{V})$. It is the convolution of two independent parts, i.e. $\mathcal{A} := \mathcal{A}_1 \times \mathcal{A}_2$ and $\mathcal{V}([\begin{smallmatrix} A_1 \\ A_2 \end{smallmatrix}]) := \mathcal{V}_1(A_1) \cap \mathcal{V}_2(A_2)$.

For every pair $(u, v) \in C_I \times C_U$ there will be a constraint $A_1 \in \mathcal{A}_1$ separating u and v. However, A_1 is not inductive. Intuitively, when applied to a configuration $w\,[\begin{smallmatrix} m \\ c \end{smallmatrix}] \in \mathcal{C}$, A_1 states "if $c(i-1)...c(i+2) = x$, then $c(i+m) \in \{\delta(x), \Box\}$", for some i, x. (Depending on v we instead may need A_1 stating just "$c(i) \in \{\alpha_0(i), \Box\}$".) This is enough to separate u and v, as v must contain an "error" somewhere. But it is not inductive: We can take any configuration which satisfies L_{write} and which has $c(i+m) = \Box$, but where the cells have not been marked correctly, s.t. executing Δ_{write} would write to position $i + m$ after reading symbols $c(j-1)...c(j+2)$ with $j \neq i$. So the resulting configuration may have $c(i+m) \neq \delta(x)$, which no longer fulfills A_1.

We solve this issue via $\mathcal{V}_2$. For the constraint A_1 from before there is going to be a constraint $A_2 \in \mathcal{A}_2$ s.t. the combination $\mathcal{V}_1(A_1) \cap \mathcal{V}_2(A_2)$ is inductive. Essentially, A_2 will ensure that it is impossible to write to position $i + m$ without reading from position i.

Non-inductive separation: $\mathcal{V}_1$. We set $\mathcal{A}_1 := \Box^s\Box^*(\Lambda^4\Box^*\Lambda + \Lambda)\Box^*$. Let $A \in \mathcal{A}_1$ denote a constraint, and $\mathcal{C}_l$ the configurations of length $|A|$. For $x \in \Lambda^*$ and index i, we write $C_i^x \subseteq \mathcal{C}_l$ for the set of configurations $w\,[\begin{smallmatrix} m \\ c \end{smallmatrix}]$ s.t. $c(i+1)...c(i+|x|) = [\begin{smallmatrix} b \\ x \end{smallmatrix}]$ for some $b \in \{0,1\}^{|x|}$.

If $A = \Box^{s+i}x\Box^j y\Box^k$, we set $\mathcal{V}_1(A)$ to the configurations $w\,[\begin{smallmatrix} m \\ c \end{smallmatrix}]$ of length $|A|$ where $c(i+j+5) \in \{y, \Box\}$ or $c(i+1)...c(i+4) \in \Lambda^4 \setminus \{x\}$.

If $A = \Box^{s+i}y\Box^k$, we simply set $\mathcal{V}_1(A)$ to the configurations with $c(i+1) \in \{y, \Box\}$.

▶ **Lemma 21.** *Let $(u, v) \in C_I \times C_U$. If $\mathcal{M}$ does not accept, there exists an $A \in \mathcal{A}_1$ with $u \in \mathcal{V}_1(A)$ and $v \notin \mathcal{V}_1(A)$.*

Proof. Let $w\,[\begin{smallmatrix} m \\ c \end{smallmatrix}] := v$. We have $v \in C_U$ and thus $c \in \Lambda^*$. As $\mathcal{M}$ does not accept, c must violate one of the conditions of Observation 19. We do a case distinction on which.

If c does not start with $\alpha_0\#$, there is some i with $c(i) \neq (\alpha_0\#)(i) =: x$ and we set $A := \Box^{s+i-1}x\Box^{|c|-s-i}$. As $u \in C_I$, we have $u(i) \in \{[\begin{smallmatrix} 0 \\ \Box \end{smallmatrix}], [\begin{smallmatrix} 0 \\ x \end{smallmatrix}]\}$, so A separates u and v.

If there is an i with $c(i+m) \neq \delta(c(i-1)...c(i+2)) =: y$, let $x := c(i-1)...c(i+2)$. We choose $A := \Box^{s+i-2}x\Box^{m-3}y\Box^{|c|-s-i-m}$. Using $u(i) = [\begin{smallmatrix} 0 \\ \Box \end{smallmatrix}]$, we immediately get $u \in \mathcal{V}(A)$. ◀

Adding inductiveness: $\mathcal{V}_2$. Let $\mathcal{A}_2 := \{0,1\}^s[0,n]^*$. Intuitively, a constraint $x\,y \in \mathcal{A}_2$ (where $|x| = s$) states: "if remainders for the first j primes have been chosen according to w, then all positions i with $y(i) \leq j$ are unmarked". To describe this formally, we say that prime p_j is *selected* in a configuration $w\,[\begin{smallmatrix} m \\ c \end{smallmatrix}] \in \mathcal{C}$ if $w(p_1 + ... + p_{j-1} + i) = 1$ for some $i \in [p_j]$. Figure 4 gives a pseudo-code representation of $\mathcal{V}_2$. Note that we allow $j = 0$ in the first line, for the case that no prime is selected.

Let i denote an index. We will now define a constraint $A_2^{(i,l)} \in \mathcal{A}_2$, which essentially encodes that the marking procedure works correctly for indices $i, i+m, i+2m, ...$ of the TM part, if it is of length l. Let $r_1, ..., r_n \in [0, p_j - 1]$ denote the unique remainders s.t. $r_j \equiv i \pmod{p_j}$. We will refer to $r_1, ..., r_n$ as the *remainder sequence* of i. Let $x \in$

Input: $u \in \mathcal{C}$
Output: $v \in \mathcal{C}$ such that $[\begin{smallmatrix} u \\ v \end{smallmatrix}] \in \Delta'_{init}$
$v := u$;
$v(1)...v(s) := 0^s$;
if $u(s+1)$ *is marked* **then** $x := \#$;
else $x := B$;
pick smallest $i' > s+1$ s.t. $u(i')$ is marked;
mark $v(s+1), v(s+2), ...$;
write x to $v(i')$;

Input: $u \in \mathcal{C}$, $A \in \mathcal{A}_2$
Output: whether $u \in \mathcal{V}_2(A)$
1 $j := J(u)$, $l := p_1 + ... + p_j$;
2 **if** $u(k) \neq A(k)$ *for some* $k \in [l]$ **then**
3   **for** $i > s$ *s.t.* $A(i) = n$ **do**
4   **if** $u(i)$ *marked* **then** reject;
5   accept;
6 **for** $k = s+1, ...$ **do**
7   **if** $A(k) \geq j \not\leftrightarrow u(k)$ *marked* **then** reject;
8 accept;

Figure 4 Pseudo-code representation of the transducers for Δ_{init} and $\mathcal{V}_2$.

$\{0,1\}^s$ be a prime part where exactly the bits corresponding to $r_1, ..., r_n$ have been set, i.e. $x(p_1 + ... + p_{j-1} + 1 + r_j) = 1$ for each j, and x is 0 elsewhere. Let $y \in [0,n]^l$ s.t. $y(k) := \min\{j \mid k \not\equiv i \pmod{p_j}\} - 1$ and $y(k) := n$ if $k \equiv i \pmod{p_j}$ for all $j \in [n]$. Then $A_2^{(i,l)} := x\,y$.

We remark that positions k with $A_2^{(i,l)}(k) = j$ have distance at least $p_1 \cdot ... \cdot p_j$; in particular, $..., y(i-m), y(i), y(i+m), ... = n$.

▶ **Lemma 22.** *$A_2^{(i,l)}$ is inductive.*

Proof. Let $u, v \in \mathcal{C}$ with $u \in \mathcal{V}_2(A_2^{(i,l)})$ and $(u,v) \in \Delta$. We have to show that $v \in \mathcal{V}_2(A_2^{(i,l)})$.

We start with Δ_{mark}. If $\mathcal{V}_2$ accepts u in line 5, then it will accept v in the same way, as Δ_{mark} only selects primes p_j with $j > J(u)$ and only unmarks cells. If it accepts u in line 8, then Δ_{mark} will select a remainder r_j where $j := J(u)+1$. If $r_j \not\equiv i \pmod{p_j}$, then v is accepted in line 5 (every k with $A_2^{(i,l)}(k) = n$ has $k - s \equiv i \pmod{p_j}$). Otherwise, let $k > s$. We consider three cases:

- If $A_2^{(i,l)}(k) < j-1$, then $u(k)$ is unmarked (as u was accepted in line 8), and $v(k)$ is as well (as Δ_{mark} only unmarks cells).
- If $A_2^{(i,l)}(k) = j-1$, then $k - s \not\equiv i \pmod{p_j}$ by definition of $A_2^{(i,l)}$, so Δ_{mark} unmarks $v(k)$.
- If $A_2^{(i,l)}(k) > j-1$, then $u(k)$ is marked and $k - s \equiv i \pmod{p_j}$, which implies $u(k) = v(k)$.

We find that $A_2^{(i,l)}(k) > j-1$ iff $v(k)$ is marked, so v will also be accepted in line 8.

The remaining transitions Δ_{write} and Δ_{init} are straightforward, as they reset the prime part and mark every cell. ◀

Concluding the proof. All that remains is combining $\mathcal{V}_1$ and $\mathcal{V}_2$.

▶ **Lemma 23.** *Let $(u,v) \in C_I \times C_U$. If $\mathcal{M}$ does not accept, there exists an inductive $A \in \mathcal{A}$ with $u \in \mathcal{V}(A)$ and $v \notin \mathcal{V}(A)$.*

Proof. Let A_1 denote the constraint given by Lemma 21. Based on the structure of A_1, there are two cases. We remark that we rely on the specific construction in the proof of Lemma 21.

The first case. We have $A_1 = \square^{s+k_1} x \square^{k_2} y \square^{k_3}$ for some k_1, k_2, k_3 (note $k_2 = m-2$ and $y = \delta(x)$). Let $i := k_1 + 4 + k_2 + 1$ denote the position of y within the TM part. We set $A_2 := A_2^{(i,|A_1|)}$ and $A := (A_1, A_2)$.

First, we show $u \in \mathcal{V}(A)$. Here, $u \in \mathcal{V}_1(A_1)$ follows from Lemma 21, and $u \in \mathcal{V}_2(A_2)$ holds due to u starting with 0^s and having all cells marked. Second, $v \notin \mathcal{V}(A)$ follows immediately from Lemma 21.

To complete the first case, we must show that A is inductive, so let $(u', v') \in \Delta$ with $u' \in \mathcal{V}(A)$. By Lemma 22 we have $v' \in \mathcal{V}_2(A_2)$, so it suffices to show $v' \in \mathcal{V}_1(A_1)$.

As Δ_{mark} only modifies the prime part and whether a cell is marked, $v' \in \mathcal{V}_1(A_1)$ follows from $u' \in \mathcal{V}_1(A_1)$ if $(u', v') \in \Delta_{mark}$.

For $(u', v') \in \Delta_{write}$, observe that only line 7 can cause A_1 to be violated, and only if we write to $v(i)$ (recall that writing only changes a cell if it previously contained $\square$). For this to occur, $u' \in C_{good}$ must hold, i.e. the prime part of u' has precisely one bit set for every prime. In particular, we have $J(u') = n$. If $u'(k) \neq A_2(k)$ for some $k \in [s]$, constraint A_2 implies that $u'(i)$ is not marked (note $A_2(i) = n$). Otherwise, A_2 implies that $u'(k)$ is marked iff $A_2(k) = n$. So both $u'(i-m)$ and $u'(i)$ are marked, and there are no marked cells between them. Hence, Δ'_{write} will read $u'(i-m)$ in line 4. Let $[\,{}^{b}_{x'}\,] := u'(i-m-1)...u'(i-m+2)$. There are three possibilities: If $x' = x$, then $v'(i) = \delta(x) = y$. If $x' \neq x$ and $x' \notin \Lambda^4$, then $\square$ appears in x' and $v'(i) = \delta(x) = \square$. Otherwise, $x' \neq x$ and $x' \in \Lambda^4$. In all cases, we get $v' \in \mathcal{V}_1(A_1)$.

Now we consider $(u', v') \in \Delta_{init}$. We again observe that A_2 can only be violated if Δ_{init} writes to $u'(i)$. Similarly to before, we use $u' \in C_{good}$ to conclude that $u'(i)$ is only marked if $u'(i-m)$ is marked, but we have $i - m > s + 1$ from the definition of A_1.

The second case. This case is both simpler than the first, and in large part analogous to it. We have $A_1 = \square^{s+k_1} y \square^{k_2}$ with $k_1 \leq m + 1$. Again, we write $i := s + k_1 + 1$ for the position of y and set $A_2 := A_2^{(i,|A_1|)}$ and $A := (A_1, A_2)$. Observe that $y = \#$ if $i = s + m + 1$ and $y = B$ otherwise.

Similar to before, the only difficult part is showing that A is inductive w.r.t. Δ_{write} and Δ_{init}; we define u', v' as before. For both types of transitions we again find that only writing to $u'(i)$ can violate A_1. Using $u' \in C_{good}$ we then have that i is the first marked position after $s + 1$ (using $i - m \leq s + m - m$). So Δ_{write} in fact does not write to i (line 5 picks a larger index). For Δ_{init} we have that $u'(s+1)$ is marked iff $s + 1 = i - m$ iff $y = \#$. So Δ_{init} writes y to $u'(i)$, and constraint A_2 holds. ◀

This yields the desired theorem.

▶ **Theorem 24.** *AbstractSafety is* EXPSPACE*-hard.*

Proof. We have constructed an RTS $\mathcal{S}$ which has size polynomial in n. By Lemmata 20 and 23 an inductive constraint separating C_I and C_U exists if and only if $\mathcal{M}$ does not accept. ◀

C Complexity of the separability problem

▶ **Theorem 18.** *Separability is* PSPACE*-complete, even if Δ is length-preserving. If $\mathcal{V}$ is length-preserving, then Separability is* NP*-complete.*

Proof. Membership in PSPACE and in NP has already been shown (see the paragraph in the main text after the statement of the theorem). We prove that the problems are PSPACE-hard and NP-hard, respectively.

PSPACE-hardness. For the PSPACE-hardness proof, we reduce from the following problem: Given a deterministic Turing machine $\mathcal{M}$ with n states, does it accept when run on an empty tape with $n - 2$ cells?

Let Q denote the states of $\mathcal{M}$, $q_0, q_f \in Q$ the initial and final state, respectively, and let $\Gamma := \{B, 0, 1\}$ denote its tape alphabet, with B being the blank symbol. We represent a configuration of $\mathcal{M}$ by a word $\#\,\beta\,q\,\eta$ (of length n), where $\#$ is a separator, β, η encode the tape contents to the left and right of the head, respectively, and q is the current state of $\mathcal{M}$. Let $\Lambda := Q \cup \Gamma \cup \{\#\}$ be the symbols used for this. We encode the unique (infinite) run of $\mathcal{M}$ by concatenating the encodings of each configuration, e.g. $\alpha := \alpha_0\alpha_1...$, where α_i represents the i-th configuration of $\mathcal{M}$.

Observe that $\alpha_0 = \#\,q_0\,B^{n-2}$, and that the symbol $\alpha(i)$ is determined by the four symbols $\alpha(i-n-1)...\alpha(i-n+2)$ and the transition relation of $\mathcal{M}$. We write $\delta(x_1...x_4)$ to denote this mapping (i.e. $\alpha(i) = \delta(\alpha(i-n-1)...\alpha(i-n+2))$). Finally, note that $\mathcal{M}$ accepts iff q_f occurs in α.

We will define a regular transition system $\mathcal{S} = (\mathcal{C}, \Delta)$ with configurations $c_I, c_U \in \mathcal{C}$ of initial and final configurations, as well as a regular abstraction framework $(\mathcal{C}, \mathcal{A}, \mathcal{V})$. We then need to show that there exists a constraint $A \in \mathcal{A}$ separating c_I and c_U, iff $\mathcal{M}$ accepts.

Let us now give the formal description of the $\mathcal{S}$ and $(\mathcal{C}, \mathcal{A}, \mathcal{V})$. We set $\mathcal{C} := \Lambda^*$. Let $c \in \mathcal{C}$. A transition of the RTS nondeterministically picks a position i, reads the symbols $c(i-1)...c(i+2)$ and then replaces $c(i+n)$ with $\delta(c(i-1)...c(i+2))$.

For the abstraction framework, we set $\mathcal{A} := \Lambda^*$. A constraint $A \in \mathcal{A}$ is interpreted as follows: If A does not contain q_f, $\mathcal{V}(A) := \emptyset$. Otherwise, $\mathcal{V}(A)$ contains precisely the prefixes of A. Clearly, this can be done by a transducer with a constant number of states.

Let $c_I := \alpha_0$ and $c_U := q_f$. We remark that there are many possible choices for c_U, indeed any word which is not a prefix of α works. Also note that the RTS cannot execute any transition starting in c_I — however, the existence of those transitions still restricts which constraints are inductive. We make the following claim:

Claim: Let $A \in \mathcal{A}$ denote an inductive constraint with $c_I \in \mathcal{V}(A)$. Then A is a prefix of α containing q_f.

Proof of the claim: If q_f does not appear in A, then $\mathcal{V}(A) = \emptyset$, contradicting $c_I \in \mathcal{V}(A)$. It follows that q_f occurs in A, and thus $c_I \in \mathcal{V}(A)$ implies that $c_I = \alpha_0 = \#\,q_0\,B^n$ is a prefix of A.

It remains to show that A is a prefix of α, so assume the contrary. Then there is a position i with $A(i) \neq \alpha(i)$. We fix a minimal such i. From the previous paragraph we get $i > n$, and thus $A(i) \neq \alpha(i) = \delta(\alpha(i-n-1)...\alpha(i-n+2)) = \delta(A(i-n-1)...A(i-n+2))$. This proves the claim.

We can now view A as a configuration of $\mathcal{S}$, with $A \in \mathcal{V}(A)$. By executing the transition that writes to position i, we move to a configuration $c \neq A$, using the previous inequality. But then $c \notin \mathcal{V}(A)$, so A would not be inductive. This proves the claim.

We can now prove that the reduction is correct. If an inductive constraint separating c_I and c_U exists, then $\mathcal{M}$ accepts. Conversely, if $\mathcal{M}$ accepts, let A denote any prefix of α that contains q_f. In particular, α_0 is a prefix of A, and so $c_I \in \mathcal{V}(A)$. Further, since by definition $\mathcal{V}(A)$ contains precisely the prefixes of A, and q_f is not a prefix of A, we have $c_U \notin \mathcal{V}(A)$. So A separates c_U from c_I.

NP-hardness. For NP-hardness, we reduce from the 3-colourability problem for graphs. Let $G = (V, E)$ be a graph with $V = \{1, \ldots, n\}$. We construct in polynomial time an RTS, a abstraction framework, and two configurations c, c' such that c' is separable from c iff G is 3-colourable.

The RTS. We encode an edge $\{i, j\} \in E$ as a word in $\{0,1\}^n$ by setting the i-th and j-th letter to 1 while all others are 0. Let $C_E \subseteq \{0,1\}^n$ be the set of encodings of all edges. We define $\mathcal{C} = C_E \cup \{0^n\}$. We define $\Delta = C_E \times C_E$. Intuitively, the transition system allows us to "move" from an edge of E to any other edge. Clearly, it is an RTS recognised by a transducer with four states.

The abstraction framework. Let $\mathcal{A} = \{R, G, B\}^n$ (where R, G, B stand for red, gree, and blue) and define the interpretation $\mathcal{V}$ as follows $(x_1 \ldots x_n, A_1 \ldots A_n) \in \mathcal{V}$ if and only if there are exactly two indices i, j such that $x_i = x_j = 1$ and $A_i \neq A_j$. In other words, a constraint is satisfied by all edges whose endpoints have different colours under A. It follows that a constraint A is inductive iff it defines a colouring of G.

The configurations c and c'. Let c be the encoding on an arbitrary edge of E, and let $c' = 0^n$.

Correctness. If G is colourable, then there exists at least one inductive constraint A. Further, c satisfies A, and c' does not, because c' does not satisfy any constraints at all. So c' is separable from c. If G is not colourable, then no constraint of $\mathcal{A}$ is inductive, and so c' is not separable from c. ◀

D Reducing length-preserving separability to SAT

A explained in the main text of the paper, it suffices to construct a Boolean formula whose satisfying assignments are the words rejected by a given NFA. Let $(Q, \Sigma, \delta, Q_0, F)$ be an NFA. We introduce some propositional variables:

- Variables $(\sigma, i) \in \Sigma \times \{1, \ldots, n\}$ encode the word by setting (σ, i) to true if and only if the i-th letter of the word is σ. For every i, exactly one variable in the set $\{(\sigma, i) : \sigma \in \Sigma\}$ can be made true by any satisfying assignment for the formula. It is straightforward to encode this as a formula ψ_i.
- Variables $(q, i) \in Q \times \{0, \ldots, n\}$ encode that state q can be reached in the NFA after having read the first i letters of the word.

Consider the formula

$$\varphi = \bigwedge_{i=1}^{n} \psi_i \wedge \bigwedge_{q \in Q_0} (q, 0) \wedge \bigwedge_{q \in Q \setminus Q_0} \neg(q, 0) \wedge \bigwedge_{i=1}^{n} \bigwedge_{p \in Q} \left((p, i) \leftrightarrow \bigvee_{p \in \delta(q, \sigma)} (q, i-1) \wedge (\sigma, i) \right). \quad (1)$$

For any satisfying assignment χ of this formula, we have $\chi((q, i)) = 1$ if and only if one can reach the state q in the NFA after reading a word $\sigma_1 \ldots \sigma_{i-1}$ where $\chi((\sigma_j, j)) = 1$ for all $1 \leq j < i$. To enforce that the NFA must not accept the word, one adds the clause $\bigwedge_{f \in F} \neg(f, n)$.