Regular Model Checking Upside-Down: An Invariant-Based Approach

Javier Esparza
Department of Informatics – I7, Technische Universität München, Germany

Mikhail Raskin
Department of Informatics – I7, Technische Universität München, Germany

Christoph Welzel
Department of Informatics – I7, Technische Universität München, Germany

Abstract

Regular model checking is a technique for the verification of infinite-state systems whose configurations can be represented as finite words over a suitable alphabet. It applies to systems whose set of initial configurations is regular, and whose transition relation is captured by a length-preserving transducer. To verify safety properties, regular model checking iteratively computes automata recognizing increasingly larger regular sets of reachable configurations, and checks if they contain unsafe configurations. Since this procedure often does not terminate, acceleration, abstraction, and widening techniques have been developed to compute a regular superset of the reachable configurations.

In this paper we develop a complementary procedure. Instead of approaching the set of reachable configurations from below, we start with the set of all configurations and approach it from above. We use that the set of reachable configurations is equal to the intersection of all inductive invariants of the system. Since this intersection is non-regular in general, we introduce b-bounded invariants, defined as those representable by CNF-formulas with at most b clauses. We prove that, for every $b \geq 0$, the intersection of all b-bounded inductive invariants is regular, and we construct an automaton recognizing it. We show that whether this automaton accepts some unsafe configuration is in EXPSPACE for every $b \geq 0$, and PSPACE-complete for $b = 1$. Finally, we study how large must b be to prove safety properties of a number of benchmarks.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Program analysis; Theory of computation → Invariants

Keywords and phrases parameterized verification, structural analysis, regular languages, regular model-checking, traps

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.23

Related Version *Full Version*: `https://arxiv.org/abs/2205.03060`

Supplementary Material *Software (Repository of examples)*: `https://doi.org/10.5281/zenodo.6483615`

Funding The project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS).

1 Introduction

Regular model checking (RMC) is a framework for the verification of different classes of infinite-state systems (see, e.g., the surveys [5, 1, 6, 2]). In its canonical version, RMC is applied to systems satisfying the following conditions: configurations can be encoded as words, the set of initial configurations is recognized by a finite automaton $\mathcal{A}_I$, and the transition relation is recognized by a length-preserving transducer $\mathcal{A}_T$. RMC algorithms address the problem of, given a regular set of unsafe configurations, deciding if its intersection with the set of reachable configurations is empty or not. In the present paper we do not consider generalisations to non-length-preserving transitions.

The fundamental building block of current RMC algorithms is an automata-theoretic construction that, given a non-deterministic automaton (NFA) A recognizing a regular set of configurations, produces another NFA recognizing the set of immediate successors (or predecessors) of $\mathcal{L}(A)$ with respect to the transition relation represented by $\mathcal{A}_T$. Therefore, if some unsafe configuration is reachable, one can find a witness by, starting with the set of initial configurations, repeatedly adding the set of immediate successors. However, this approach never terminates when all reachable configurations are safe. Research on RMC has produced many acceleration, abstraction, and widening techniques to make the iterative computation "jump over the fixpoint" in finite time, and produce an invariant of the system not satisfied by any unsafe configuration (see, e.g., [10, 22, 15, 4, 7, 9, 11, 8, 23, 14]).

In this paper we develop a complementary approach that, starting with the set of all configurations, computes increasingly smaller regular inductive invariants, i.e., sets of configurations closed under the reachability relation and containing all initial configurations. Our main contribution is the definition of a sequence of *regular* inductive invariants that converges (in the limit) to the set of reachable configurations, and for which automata can be directly constructed from $\mathcal{A}_I$ and $\mathcal{A}_T$.

Our starting point is the fact that the set of reachable configurations is equal to the intersection of all inductive invariants. Since this intersection is non-regular in general, we introduce *b-bounded* invariants. An invariant is b-bounded if, for every $\ell \geq 0$, the configurations of length ℓ satisfying the invariant are those satisfying a Boolean formula in conjunctive normal form with at most b clauses. For example, assume that the configurations of some system are words over the alphabet $\{a, b, c, d\}$, and that the configurations of length five where the second letter is an a or the fourth letter is a b, and the second letter is a b or the third is a c, constitute an inductive invariant. Then this set of configurations is a 2-bounded invariant, represented by the formula $(a_{2:5} \vee b_{4:5}) \wedge (b_{2:5} \vee d_{3:5})$. We prove that, for every bound $b \geq 0$, the intersection of all b-bounded inductive invariants, denoted IndInv_b, is regular, and recognized by a DFA of double exponential size in $\mathcal{A}_I$ and $\mathcal{A}_T$. As a corollary, we obtain that, for every $b \geq 0$, deciding if IndInv_b contains some unsafe configuration is in EXPSPACE.

In the second part of the paper, we study the special case $b = 1$ in more detail. We exploit that 1-bounded inductive invariants are closed under union (a special feature of the $b = 1$ case) to prove that deciding if IndInv_1 contains some unsafe configuration is PSPACE-complete. The proof also shows that IndInv_b can be recognized by a NFA of single exponential size in $\mathcal{A}_I$ and $\mathcal{A}_T$.

The index b of a bounded invariant can be seen as a measure of how difficult it is for a human to understand the invariant. So one is interested in the smallest b such that IndInv_b is strong enough to prove a given property. In the third and final part of the paper, we experimentally show that for a large number of systems IndInv_1 is strong enough to prove useful safety properties.

Related work. The work closest to ours is [3], which directly computes an overapproximation of the set of reachable configurations of a parameterized system. Contrary to our approach, the paper computes one single approximation, instead of a converging sequence of overapproximations. Further, the method is designed for a model of parameterized systems with existential or universal guarded commands, while our technique can be applied to any model analyzable by RMC. Our work is also related to [14], which computes an overapproximation using a learning approach, which terminates if the set of reachably configurations is regular; our paper shows that a natural class of invariants is regular, and that automata for them can be constructed explicitly from the syntactic description of the system. This paper generalizes the work of [12, 19, 13, 20] on trap invariants for parameterized Petri nets. Trap invariants are a special class of 1-bounded invariants, and the parameterized Petri nets studied in these papers can be modeled in the RMC framework. An alternative to regular model checking are logical based approaches. The invisible invariant method synthesizes candidate invariants from examples, which are then checked for inductiveness [26]. Our approach does not produce candidates, it generates invariants by construction. Modern tools like Ivy [25, 24] have verified more complex protocols than the ones in Section 5 using a combination of automation and human interaction. The best way of achieving this interaction is beyond the scope of this paper, which focuses on the foundations of regular model checking.

Structure of the paper. Section 2 introduces basic definitions of the RMC framework. Section 3 introduces b-bound invariants and proves regularity of IndInv_b. Section 4 proves the PSPACE-completeness result. Sections 5 and 6 contain some experimental results and conclusions.

Full version. All missing proofs can be found in the full version of this paper [21].

2 Preliminaries

Automata and transducers. Given $n, m \in \mathbb{N}$, we let $[n, m]$ denote the set $\{i \in \mathbb{N} \colon n \leq i \leq m\}$. Given a a word w of length ℓ, we let $w[i]$ denote the i-th letter of w, i.e., $w = w[1] \cdots w[\ell]$.

A nondeterministic finite automaton (NFA) is a tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, Q_0, F\rangle$, where Q is a non-empty finite set of states, Σ is an alphabet, $\Delta\colon Q \times \Sigma \to 2^Q$ is a transition function, and $Q_0, F \subseteq Q$ are sets of initial and final states, respectively. A run of $\mathcal{A}$ on a word $w \in \Sigma^\ell$ is a sequence $q_0\, q_1\, \ldots\, q_\ell$ of states such that $q_0 \in Q_0$ and $q_i \in \Delta(q_{i-1}, w[i])$ for every $i \in [1, \ell]$. A run on w is accepting if $q_\ell \in F$, and $\mathcal{A}$ accepts w if there exists an accepting run of $\mathcal{A}$ on w. The language recognized by $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$ or $\mathcal{L}_\mathcal{A}$, is the set of words accepted by $\mathcal{A}$. We let $|\mathcal{A}|$ denote the number of states of $\mathcal{A}$. A NFA $\mathcal{A}$ is deterministic (*DFA*) if $|Q_0| = 1$ and $|\Delta(q, a)| = 1$ for every $q \in Q$ and $a \in \Sigma$.

The function $\delta_\mathcal{A}\colon 2^Q \times \Sigma^* \to 2^Q$ is defined inductively as follows: $\delta_\mathcal{A}(P, \varepsilon) = P$ and $\delta_\mathcal{A}(P, aw) = \delta_\mathcal{A}(\bigcup_{p\in P} \Delta(p, a), w)$. Observe that $\mathcal{A}$ accepts w iff $\delta_\mathcal{A}(Q_0, w) \cap F \neq \emptyset$.

A (length-preserving) transducer over $\Sigma \times \Gamma$ is a NFA over an alphabet $\Sigma \times \Gamma$. We denote elements of $\Sigma \times \Gamma$ as $\langle a, b\rangle$ or $\begin{bmatrix} a \\ b \end{bmatrix}$, where $a \in \Sigma$ and $b \in \Gamma$. Given two words $w \in \Sigma^\ell, u \in \Gamma^\ell$ of the same length ℓ and a transducer $\mathcal{A}$, we say that $\mathcal{A}$ accepts $\langle w, u\rangle$, or transduces w into u, if it accepts the word $\langle w[1], u[1]\rangle \cdots \langle w[\ell], u[\ell]\rangle \in (\Sigma \times \Gamma)^\ell$.

Regular model checking. Regular model checking (RMC) is a framework for the verification of systems with infinitely many configurations. Each configuration is represented as a finite word over a fixed alphabet Σ. Systems are modeled as regular transition systems (RTS).

▶ **Definition 1** (Regular transition systems). *A RTS is a triple $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T\rangle$, where Σ is an alphabet, $\mathcal{A}_I$ is a NFA over Σ, and $\mathcal{A}_T$ is a transducer over $\Sigma \times \Sigma$.*

Words over Σ are called *configurations*. Configurations accepted by $\mathcal{A}_I$ are called *initial*, and pairs of configurations accepted by $\mathcal{A}_T$ are called *transitions*. We write $w \leadsto u$ to denote that $\langle w, u\rangle$ is a transition. Observe that $w \leadsto u$ implies $|w| = |u|$. Given two configurations w, u, we say that u is *reachable* from w if $w \leadsto^* u$, where $\leadsto^*$ denotes the reflexive and transitive closure of $\leadsto$. The set of reachable configurations of $\mathcal{R}$, denoted $\mathit{Reach}(\mathcal{R})$, or just Reach when there is no confusion, is the set of configurations reachable from the initial configurations. In the following, we use $|\mathcal{R}|$ to refer to $|\mathcal{A}_I| + |\mathcal{A}_T|$.

▶ **Example 2** (Dining philosophers)**.** We model a very simple version of the dining philosophers as a RTS, for use as running example. Philosophers sit at a round table with forks between them. Philosophers can be *thinking* (t) or *eating* (e). Forks can be *free* (f) or *busy* (b). A thinking philosopher whose left and right forks are free can *simultaneously* grab both forks – the forks become busy – and start eating. After eating, the philosopher puts both forks to the table and returns to thinking. The model includes two corner cases: a table with one philosopher and one fork, which is then both the left and the right fork (unusable as it would need to be grabbed twice in a single transition), and the empty table with no philosophers or forks.

We model the system as a RTS over the alphabet $\Sigma = \{t, e, f, b\}$. A configuration of a table with n philosophers and n forks is represented as a word over Σ of length $2n$. Letters at odd and even positions model the current states of philosophers and forks (positions start at 1). For example, $tftf$ models a table with two thinking philosophers and two free forks. The language of initial configurations is $\mathcal{L}_I = (tf)^*$, and the language of transitions is

$$\mathcal{L}_T = \begin{bmatrix} t\\ e\end{bmatrix}\begin{bmatrix} f\\ b\end{bmatrix}\begin{bmatrix} x\\ x\end{bmatrix}^*\begin{bmatrix} f\\ b\end{bmatrix} \mid \begin{bmatrix} e\\ t\end{bmatrix}\begin{bmatrix} b\\ f\end{bmatrix}\begin{bmatrix} x\\ x\end{bmatrix}^*\begin{bmatrix} b\\ f\end{bmatrix} \mid \begin{bmatrix} x\\ x\end{bmatrix}^*\left(\begin{bmatrix} f\\ b\end{bmatrix}\begin{bmatrix} t\\ e\end{bmatrix}\begin{bmatrix} f\\ b\end{bmatrix} \mid \begin{bmatrix} b\\ f\end{bmatrix}\begin{bmatrix} e\\ t\end{bmatrix}\begin{bmatrix} b\\ f\end{bmatrix}\right)\begin{bmatrix} x\\ x\end{bmatrix}^*$$

where $\begin{bmatrix} x\\ x\end{bmatrix}$ stands for the regular expression $\left(\begin{bmatrix} t\\ t\end{bmatrix} \mid \begin{bmatrix} e\\ e\end{bmatrix} \mid \begin{bmatrix} f\\ f\end{bmatrix} \mid \begin{bmatrix} b\\ b\end{bmatrix}\right)$. The first two terms of $\mathcal{L}_T$ describe the actions of the first philosopher, and the second the actions of the others. It is not difficult to show that $\mathit{Reach} = (t(f \mid beb))^* \mid ebt((f \mid beb)t)^*$. (These are the configurations where no two philosophers are using the same fork, and fork states match their adjacent philosopher states.)

Safety verification problem for RTSs. The safety verification problem for RTSs is defined as follows: Given a RTS $\mathcal{R}$ and a NFA $\mathcal{U}$ recognizing a set of *unsafe* configurations, decide whether $\mathit{Reach}(\mathcal{R}) \cap \mathcal{L}_{\mathcal{U}} = \emptyset$ holds. The problem is known to be undecidable.

3 Bounded inductive invariants of a RTS

We present an invariant-based approach to the safety verification problem for RTSs. Fix a RTS $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T\rangle$. We introduce an infinite sequence

$$\Sigma^* = \mathit{IndInv}_0 \supseteq \mathit{IndInv}_1 \supseteq \mathit{IndInv}_2 \ldots \supseteq \mathit{Reach}$$

of effectively regular inductive invariants of $\mathcal{R}$ that converges to Reach, i.e., IndInv_k is effectively regular for every $k \geq 1$, and $\mathit{Reach} = \bigcap_{k=0}^{\infty} \mathit{IndInv}_k$. Section 3.1 recalls basic notions about invariants, Section 3.2 defines the inductive invariant IndInv_b for every $b \geq 0$, and Section 3.3 shows that IndInv_b is regular.

3.1 Invariants

Let $S \subseteq \Sigma^*$ be a set of configurations. S is an *invariant for length* ℓ if $\mathit{Reach} \cap \Sigma^\ell \subseteq S \cap \Sigma^\ell$, and an *invariant* if $\mathit{Reach} \subseteq S$. Observe that, since $\mathcal{A}_T$ is a length-preserving transducer, S is an invariant iff it is an invariant for every length. A set S (invariant or not) is *inductive* if it is closed under reachability, i.e. $w \in S$ and $w \rightsquigarrow u$ implies $u \in S$. Given two invariants I_1, I_2, we say that I_1 is *stronger* than I_2 if $I_1 \subset I_2$. Observe that inductive invariants are closed under union and intersection, and so there exists a unique strongest inductive invariant of $\mathcal{R}$. Since *Reach* is an inductive set, the strongest inductive invariant is *Reach*.

▶ **Example 3.** The set $I_0 = ((t \mid e)(f \mid b))^*$ is an inductive invariant of the dining philosophers. Other inductive invariants are

$$I_1 = \overline{\Sigma^* e f \Sigma^*} \quad I_2 = \overline{\Sigma^* f e\, \Sigma^*} \quad I_3 = \overline{e\, \Sigma^* f} \quad I_4 = \overline{\Sigma^* t\, b\, t \Sigma^*} \quad I_5 = \overline{t\, \Sigma^* t\, b}$$

Taking into account that the table is round, these are the sets of configurations without any occurrence of $e f$ (I_1), $f e$ (I_2 and I_3), and $t\,b\,t$ (I_4 and I_5).

3.2 Bounded invariants

Given a length $\ell \geq 0$, we represent certain sets of configurations as Boolean formulas over a set AP_ℓ of atomic propositions. More precisely, a Boolean formula over AP_ℓ describes a set containing *some* configurations of length ℓ, and *all* configurations of other lengths.

The set AP_ℓ contains an atomic proposition $q_{j:\ell}$ for every $q \in \Sigma$ and for $j \in [1, \ell]$. A *formula* φ over AP_ℓ is a positive Boolean combination of atomic propositions of AP_ℓ and the constants *true* and *false*. Formulas are interpreted on configurations. Intuitively, an atomic proposition $q_{j:\ell}$ states that either the configuration does *not* have length ℓ, or it has length ℓ and its j-th letter is q. Formally, $w \in \Sigma^*$ *satisfies* φ, denoted $w \models \varphi$, if $\varphi = q_{j:\ell}$ and $|w| \neq \ell$ or $|w| = \ell$ and $w[j] = q$; for the other cases, i.e., for $\varphi = \mathit{true}, \neg\varphi_1, \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2$, satisfaction is defined as usual. The language $\mathcal{L}(\varphi) \subseteq \Sigma^*$ of a formula is the set of configurations that satisfy φ. We also say that φ *denotes* the set $\mathcal{L}(\varphi)$. A formula is inductive if it denotes an inductive set.

▶ **Example 4.** In the dining philosophers, let $\varphi = (e_{1:4} \wedge b_{4:4}) \vee f_{2:4}$. We have

$$\mathcal{L}(\varphi) = \epsilon \mid \Sigma \mid \Sigma^2 \mid \Sigma^3 \mid (e\, \Sigma\, \Sigma\, b \mid \Sigma\, f\, \Sigma\, \Sigma) \mid \Sigma^5 \Sigma^* \ .$$

Observe that an expression like $(q_{1:1} \wedge r_{1:2})$ is not a formula, because it combines atomic propositions of two different lengths, which is not allowed. Notice also that $\neg q_{j:\ell}$ is equivalent to $\bigvee_{r \in \Sigma \setminus \{q\}} r_{j:\ell}$. Therefore, if we allowed negative conditions, we would still have the same class of expressible predicates on words of a given length (and we would not obtain formulas for the same predicates with fewer clauses.) Abusing language, if φ is a formula over AP_ℓ and $\mathcal{L}(\varphi)$ is an (inductive) invariant, then we also say that φ an (inductive) invariant. Observe that (inductive) invariants are closed under conjunction and disjunction.

Convention: From now on, "formula" means "positive formula in CNF".

▶ **Definition 5.** *Let* $b \geq 0$. *A* b-formula *is a formula with at most* b *clauses (with the convention that true is the only formula with* 0 *clauses). A set* $S \subseteq \Sigma^*$ *of configurations is* b-bounded *if for every length* ℓ *there exists a* b*-formula* φ_ℓ *over* AP_ℓ *such that* $S \cap \Sigma^\ell = \mathcal{L}(\varphi_\ell)$.

Observe that, since one can always add tautological clauses to a formula without changing its language, a set S is b-bounded iff for every length ℓ there is a formula φ_ℓ with *exactly* b clauses.

▶ **Example 6.** In the dining philosophers, the 1-formulas $(t_{2i-1:\ell} \vee e_{2i-1:\ell})$ and $(f_{2i:\ell} \vee b_{2i:\ell})$ are inductive 1-invariants for every even $\ell \geq 1$ and every $i \in [1, \ell/2]$. It follows that the set I_0 of Example 3 is an intersection of (infinitely many) inductive 1-invariants. The same happens for $I_1, \ldots, I_5$. For example, I_1 is the intersection of all inductive 1-invariants of the form $(t_{i:\ell} \vee b_{i+1:\ell})$, for all $\ell \geq 1$ and all $i \in [1, \ell - 1]$; inductivity is shown by an easy case distinction.

We are now ready to define the sequence of inductive invariants we study in the paper:

▶ **Definition 7.** *Let $\mathcal{R}$ be a RTS. For every $b \geq 0$, we define IndInv_b as the intersection of all inductive b-invariants of $\mathcal{R}$.*

▶ **Proposition 8.** *Let $\mathcal{R}$ be a RTS. For every $b \geq 0$, $\mathit{IndInv}_b \supseteq \mathit{Reach}$ and $\mathit{IndInv}_b \supseteq \mathit{IndInv}_{b+1}$. Further, $\mathit{Reach} = \bigcap_{b=0}^{\infty} \mathit{IndInv}_b$.*

Proof. $\mathit{IndInv}_b \supseteq \mathit{Reach}$ follows from the fact that, since inductive invariants are closed under intersection, IndInv_b is an inductive invariant, and Reach is the strongest inductive invariant. $\mathit{IndInv}_b \supseteq \mathit{IndInv}_{b+1}$ follows from the fact that, by definition, every b-invariant is also a $b+1$-invariant. For the last part, observe that for every $\ell \geq 0$, the set $\mathit{Reach} \cap \Sigma^\ell$ is an inductive invariant for length ℓ. Let φ_ℓ be a formula over AP_ℓ such that $\mathcal{L}(\varphi_\ell) \cap \Sigma^\ell = \mathit{Reach} \cap \Sigma^\ell$, and let b_ℓ be its number of clauses. (Notice that φ_ℓ always exists, because every subset of Σ^ℓ can be expressed as a formula, and every formula can be put in conjunctive normal form.) Then φ_ℓ is a b_ℓ-bounded invariant, and so $\mathcal{L}(\varphi_\ell) \supseteq \mathit{IndInv}_{b_\ell}$ for every $\ell \geq 0$. So we have $\mathit{Reach} = \bigcap_{\ell=0}^{\infty} \mathcal{L}(\varphi_\ell) \supseteq \bigcap_{b=0}^{\infty} \mathit{IndInv}_b = \mathit{Reach}$, and we are done. ◀

Observe that, while IndInv_b is always an inductive invariant, it is not necessarily b-bounded. The reason is that b-invariants are not closed under intersection. Indeed, the conjunction of two formulas with b clauses is not always equivalent to a formula with b clauses, one can only guarantee equivalence to a formula with $2b$ clauses.

▶ **Example 9.** The deadlocked configurations of the dining philosophers are

$$\mathit{Dead} = \overline{\Sigma^* f\, t\, f\, \Sigma^*} \cap \overline{\Sigma^* b\, e\, b\, \Sigma^*} \ .$$

We prove $\mathit{IndInv}_1 \cap \mathit{Dead} = \emptyset$, which implies that the dining philosophers are deadlock-free. Let C be the set of configurations of $((t \mid e)(f \mid b))^*$ containing no occurrence of ef, fe, or $t\, b\, t$ as a cyclic word. In Example 6 we showed that C is an intersection of 1-invariants, which implies $\mathit{IndInv}_1 \subseteq C$. We prove $C \cap \mathit{Dead} = \emptyset$, which implies $\mathit{IndInv}_1 \cap \mathit{Dead} = \emptyset$. Let $w \in C$. If $|w| \leq 3$ the proof is an easy case distinction. Assume $|w| \geq 4$. We show that w contains an occurrence of $f\, t\, f$ or $b\, e\, b$, and so it is not a deadlock. If all philosophers are thinking at w, then, since w contains no occurrence of $t\, b\, t$, it contains an occurrence of ftf. If at least one philosopher is eating at w, then, since w contains no occurrence of $e\, b$ or $b\, e$, it contains an occurrence of $b\, e\, b$.

Further, for the dining philosophers we have $\mathit{Reach} = \mathit{IndInv}_3$. Apart from some corner cases (e.g. an unsatisfiable invariant for every odd length), the reason is that the 3-formula

$$(t_{i:\ell} \vee b_{i+1:\ell}) \wedge (b_{i+1:\ell} \vee t_{i+2:\ell}) \wedge (t_{i:\ell} \vee f_{i+1:\ell} \vee t_{i+2:\ell})$$

is an inductive 3 invariant for every $\ell \geq 3$ and every $i \in [1, \ell - 2]$. The configurations satisfying this invariant and the inductive 1-invariants $I_0, \ldots, I_5$ of Example 6 are the reachable configurations $\mathit{Reach} = (t(f \mid beb))^* \mid ebt((f \mid beb)t)^* b$.

It is not difficult to construct an (artificial) family $\mathcal{R}_b$ of RTSs such that $\mathit{IndInv}_b \subset \mathit{Reach}(\mathcal{R}_k) = \mathit{IndInv}_{b+1}$.

▶ **Example 10.** Fix some $b > 0$. Consider a RTS with $\Sigma = \{0, 1\}$ and $\mathcal{L}_T$ given by

$$\langle 0,0\rangle^{k_1}\ \langle 1,0\rangle\ \langle 0,0\rangle^{k_2}\ (\langle 0,0\rangle + \langle 0,1\rangle + \langle 1,0\rangle + \langle 1,1\rangle)^*$$

for every $k_1, k_2 \in \mathbb{N}$ such that $k_1 + k_2 = b - 1$. Then every transition of the RTS is of the form $u \cdot v \rightsquigarrow u' \cdot v'$, where $|u| = b = |u'|$, the word u contains exactly one 1, and the word u' contains only 0s. If we choose 0^* as set of initial configurations, then no transition is applicable to any initial configuration, and so $\mathit{Reach} = 0^*$. It is easy to check (see Appendix A of [21]) that $\mathit{IndInv}_b \supset \mathit{IndInv}_{b+1} = 0^* = \mathit{Reach}$. Also, one can easily check that if we set $\mathcal{L}_T$ to

$$\langle 0,0\rangle^{*}\ \langle 1,0\rangle\ \langle 0,0\rangle^{*}\ (\langle 0,0\rangle + \langle 0,1\rangle + \langle 1,0\rangle + \langle 1,1\rangle)$$

then $\mathit{IndInv}_b \supset 0^* = \mathit{Reach}$ for every $b > 0$.

Finally, we show in the next section that IndInv_b is regular for every $b \geq 0$, which implies that any RTS $\mathcal{R}$ such that Reach is not regular satisfies $\mathit{Reach} \neq \mathit{IndInv}_b$ for every $b \geq 0$.

3.3 IndInv_b is regular for every $b \geq 1$

We prove that IndInv_b is regular for every $b \geq 1$. For this, we first show how to encode b-formulas as words over the alphabet $(2^\Sigma)^b$, and then we prove the following two results:

1. The language of all b-formulas φ such that $\mathcal{L}(\varphi)$ is an inductive invariant is regular.
2. Given a regular language of b-formulas, the set of configurations that satisfy every formula in the language is regular.

Since IndInv_b contains the configurations that satisfy all the inductive b-invariants of $\mathcal{R}$, these two results imply that IndInv_b is regular.

Observe that a b-formula is an inductive invariant if it is satisfied by all initial configurations and is inductive. So we prove the second result in two steps. We first show that the set of b-formulas satisfied by all initial configurations is regular, and then that the set of all b-inductive formulas is regular.

Encoding b-formulas as b-powerwords. We introduce an encoding of b-formulas. We start with some examples. Assume $\mathcal{R}$ is a RTS with $\Sigma = \{a, b, c\}$. We consider formulas over AP_3, i.e., over the atomic propositions $\{a_{1:3}, a_{2:3}, a_{3:3}, b_{1:3}, b_{2:3}, b_{3:3}, c_{1:3}, c_{2:3}, c_{3:3}\}$.

We encode the 1-formula $(a_{1:3} \vee a_{2:3})$ as the word $\{a\}\,\{a\}\,\emptyset$ of length three over the alphabet 2^Σ. Intuitively, $\{a\}\,\{a\}\,\emptyset$ stands for the words of length 3 that have an a in their first or second position. Similarly, we encode $(a_{1:3} \vee b_{1:3} \vee b_{3:3})$ as $\{a, b\}\,\emptyset\,\{b\}$. Intuitively, $\{a, b\}\,\emptyset\,\{b\}$ stands for the set of words of length 3 that have a or b as first letter, or b as third letter. Since 2^Σ is the powerset of Σ, we call words over 2^Σ *powerwords*.

Consider now the 2-formula $(a_{1:3} \vee b_{1:3} \vee a_{2:3}) \wedge (b_{1:3} \vee b_{3:3} \vee c_{3:3})$. We put the encodings of its clauses "on top of each other". Since the encodings of $(a_{1:3} \vee b_{1:3} \vee a_{2:3})$ and $(b_{1:3} \vee b_{3:3} \vee c_{3:3})$ are $\{a, b\}\,\{a\}\,\emptyset$ and $\{b\}\,\emptyset\,\{b, c\}$, respectively, we encode the formula as the word

$$\begin{bmatrix}\{a,b\}\\ \{b\}\end{bmatrix} \begin{bmatrix}\{a\}\\ \emptyset\end{bmatrix} \begin{bmatrix}\emptyset\\ \{b,c\}\end{bmatrix}$$

of length three over the alphabet $2^\Sigma \times 2^\Sigma = (2^\Sigma)^2$. We call such a word a 2-*powerword*. Similarly, we encode a b-formula by a b-*powerword* of length three over the alphabet $(2^\Sigma)^b$.

In the following we overload φ to denote both a formula and its encoding as a b-powerword, and for example write

$$\varphi = \begin{bmatrix} X_{11} \\ \cdots \\ X_{b1} \end{bmatrix} \cdots \begin{bmatrix} X_{1\ell} \\ \cdots \\ X_{b\ell} \end{bmatrix} \quad \text{instead of} \quad \varphi = \bigwedge_{i=1}^{b} \bigvee_{j=1}^{\ell} \bigvee_{a \in X_{ij}} a_{i:\ell}$$

where $X_{ij} \subseteq \Sigma$. Intuitively, each row $X_{i1} \cdots X_{i\ell}$ encodes one clause of φ. We also write $\varphi = \varphi[1] \cdots \varphi[\ell]$, where $\varphi[i] \in (2^\Sigma)^b$ denotes the i-th letter of the b-powerword encoding φ. The satisfaction relation $w \models \varphi$ translates into a purely set-theoretical property:

▶ **Fact 11.** Let $w = w[1] \cdots w[\ell]$ be a configuration over Σ, and let $\varphi = \varphi[1] \ldots \varphi[\ell]$ be a b-formula, i.e., a b-powerword over $(2^\Sigma)^b$, where $\varphi[j] = \begin{bmatrix} X_{1j} \\ \cdots \\ X_{bj} \end{bmatrix}$. We have $w \models \varphi$ iff for every $i \in [1, b]$ there exists $j \in [1, \ell]$ such that $w[j] \in X_{ij}$.

A DFA for the b-formulas satisfied by all initial configurations. Given a RTS $\mathcal{R}$ and a bound $b \geq 0$, we let Init_b denote the set of all b-formulas satisfied by all initial configurations of $\mathcal{R}$. Recall that b-formulas are encoded as b-powerwords, and so Init_b is a language over the alphabet $(2^\Sigma)^b$. We show that Init_b is effectively regular:

▶ **Proposition 12.** *Let $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T \rangle$ be a RTS, and let n_I be the number of states of $\mathcal{A}_I$. For every $b \geq 1$, the language $\overline{\mathit{Init}_b}$ is recognized by a NFA with at most $n_I b$ states, and so Init_b is recognized by a DFA with at most 2^{bn_I} states.*

Proof. $\overline{\mathit{Init}_b}$ contains the set of all b-formulas φ such that $w \not\models \varphi$ for some $w \in \mathcal{L}_I$. Let $\mathcal{A}_I = (Q_I, \Sigma, \delta_I, Q_{0I}, F_I)$. We consider only the case $b = 1$, the general case is handled in Appendix B of [21]. Let $\mathcal{B} = (Q_I, 2^\Sigma, \delta_B, Q_0, F_I)$ be the NFA with the same states, initial and final states as $\mathcal{A}$, and transition relation δ_B defined as follows for every $q \in Q_I$ and $X \in 2^\Sigma$:

$$q' \in \delta_B(q, X) \text{ iff there exists } a \in \Sigma \setminus X \text{ such that } q' \in \delta_I(q, a).$$

We show that $\mathcal{B}$ recognizes $\overline{\mathit{Init}_1}$, i.e., that for every length $\ell \geq 0$, $\mathcal{B}$ accepts a 1-formula φ iff there exists a configuration w such that $w \in \mathcal{L}_I$ and $w \not\models \varphi$. By Fact 11, this is the case iff there exists an accepting run $q_0 \xrightarrow{w[1]} q_1 \cdots q_{\ell-1} \xrightarrow{w[\ell]} q_\ell$ of $\mathcal{A}_I$ such that $w[j] \notin \varphi[j]$ for every $j \in [1, \ell]$. By the definition of $\mathcal{B}$, this is the case iff $q_0 \xrightarrow{\varphi[1]} q_1 \cdots q_{\ell-1} \xrightarrow{\varphi[\ell]} q_\ell$ is an accepting run of $\mathcal{B}$. ◀

A DFA for the inductive b-formulas. For every $b \geq 0$, let Ind_b be the set of inductive b-formulas. We show that Ind_b is effectively regular:

▶ **Proposition 13.** *Let $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T \rangle$ be an RTS, and let n_T be the number of states of $\mathcal{A}_T$. $\overline{\mathit{Ind}_b}$ is recognized by a NFA with at most $n_T b 2^b$ states, and so Ind_b is recognized by a DFA with at most $2^{n_T b 2^b}$ states.*

Proof. Let $\mathcal{A}_T = (Q_T, \Sigma, \delta_T, Q_{0T}, F_T)$. We consider only the case $b = 1$, for the general case see Appendix B of [21].

Let $\mathcal{C} = (Q_C, 2^\Sigma \times \Sigma, \delta_C, Q_{0C}, F_C)$ be a transducer accepting the words $\langle \varphi, w \rangle$ such that $\varphi \in (2^\Sigma)^*$ is a 1-formula and $w \models \varphi$, i.e., $w[j] \in \varphi[j]$ for at least one $j \in [1, \ell]$ (Fact 11). It is trivial to construct a transducer for this language with two states.

We define the NFA $\mathcal{B} = (Q_C \times Q_T, 2^\Sigma, \delta_B, Q_{0C} \times Q_{0T}, F_C \times F_T)$ over 2^Σ with transition relation δ_B as follows:

$$(q', r') \in \delta_B((q,r), X) \text{ iff } \exists a_1 \in \Sigma, a_2 \in \Sigma \setminus X \colon q' \in \delta_C\,(q, \langle X, a_1\rangle) \wedge r' \in \delta_T\,(r, \langle a_1, a_2\rangle).$$

We show that $\mathcal{B}$ recognizes $\overline{Ind_1}$. A 1-formula φ is not inductive iff there exist configurations w, u satisfying three conditions: $w \models \varphi$, i.e., $w[j] \in \varphi[j]$ for some $j \in [1,\ell]$; $\langle w, u\rangle \in \mathcal{L}(\mathcal{A}_t)$; and $u \not\models \varphi$, i.e., $u[j] \notin \varphi[j]$ for every $j \in [1,\ell]$ (Fact 11). By the definition of $\mathcal{C}$, this is the case iff there are accepting runs

$$q_0 \xrightarrow{\langle \varphi[1], w[1]\rangle} q_1 \cdots q_{\ell-1} \xrightarrow{\langle \varphi[\ell], w[\ell]\rangle} q_\ell \quad \text{and} \quad r_0 \xrightarrow{\langle w[1], u[1]\rangle} r_1 \cdots r_{\ell-1} \xrightarrow{\langle w[\ell], u[\ell]\rangle} r_\ell$$

of $\mathcal{C}$ and $\mathcal{A}_T$, respectively. By the definition of $\mathcal{B}$, this the case iff

$$(q_0, r_0) \xrightarrow{\varphi[1]} (q_1, r_1) \cdots (q_{\ell-1}, r_{\ell-1}) \xrightarrow{\varphi[\ell]} (q_\ell, r_\ell)$$

is an accepting run of $\mathcal{B}$. ◀

A DFA for the set of configurations satisfying a regular set of b-formulas. Given a NFA $\mathcal{A}$ over the alphabet $(2^\Sigma)^b$, i.e., a NFA recognizing a language of b-formulas, let $Sat(\mathcal{A})$ be the set of configurations satisfying all b-formulas of $\mathcal{L}(\mathcal{A})$.

▶ **Proposition 14.** *Let $\mathcal{R}$ be a RTS over Σ, and let $\mathcal{A}$ be a NFA over $(2^\Sigma)^b$ with m states. $\overline{Sat(\mathcal{A})}$ is recognized by a NFA with at most mb states, and so $Sat(\mathcal{A})$ is recognized by a DFA with at most 2^{mb} states.*

Proof. Let $\mathcal{A} = (Q, 2^\Sigma, \delta_A, Q_0, F)$. We consider only the case $b = 1$, for the general case see Appendix B of [21]. Let φ be a 1-formula of length ℓ. By Fact 11 we have $w \models \varphi$ iff $\bigvee_{j=1}^{\ell} w[j] \in \varphi[j]$

Consider the NFA $\mathcal{B} = (Q, \Sigma, \delta_B, Q_0, F)$ over Σ with the same states, initial and final states as $\mathcal{A}$, and transition relation defined as follows:

$$q' \in \delta_B(q, a) \text{ iff there exists } X \in 2^\Sigma \text{ such that } q' \in \delta_A(q, X) \text{ and } a \notin X.$$

We show that $\mathcal{B}$ recognizes $\overline{Sat(\mathcal{A})}$. More precisely, we show that $w \notin Sat(\mathcal{A})$ iff $w \in \mathcal{L}(\mathcal{B})$ holds for every configuration w. Let $\ell \geq 0$, and let w be an arbitrary configuration of length ℓ. We have $w \notin Sat(\mathcal{A})$ iff there is an accepting run $q_0 \xrightarrow{\varphi[1]} q_1 \cdots q_{\ell-1} \xrightarrow{\varphi[\ell]} q_\ell$ of $\mathcal{A}$ such that $w[j] \notin \varphi[j]$ for every $j \in [1,\ell]$. By the definition of $\mathcal{B}$, this is the case iff $q_0 \xrightarrow{\varphi[1]} q_1 \cdots q_{\ell-1} \xrightarrow{\varphi[\ell]} q_\ell$ is an accepting run of $\mathcal{B}$. ◀

Putting everything together. We combine the previous results to show that, given a RTS $\mathcal{R}$, the complement of $IndInv_b$ is recognized by a NFA whose number of states is exponential in $\mathcal{R}$ and double exponential in b.

▶ **Theorem 15.** *Let $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T\rangle$ be a RTS. Let n_I and n_T be the number of states of $\mathcal{A}_I$ and $\mathcal{A}_T$, respectively, and let $K = n_I b + n_T 2^b$. For every $b \geq 0$, the set $\overline{IndInv_b}$ is recognized by a NFA with at most $2^K b$ states, and so $IndInv_b$ is recognized by a DFA with at most 2^{b2^K} states.*

Proof. Recall that $IndInv_b$ contains the configurations satisfying all b-formulas that are inductive invariants of $\mathcal{R}$. A b-formula is an inductive invariant iff it is inductive and it is satisfied by all initial configurations of $\mathcal{R}$. So $Init_b \cap Ind_b$ is the language of all b-formulas (equivalently, all b-powerwords) that are inductive invariants. By Propositions 12 and 13, this language is recognized by a DFA $\mathcal{A}$ with at most $2^{n_I b} \cdot 2^{n_T 2^b} = 2^K$ states. A configuration w belongs to $IndInv_b$ iff it satisfies every formula of $Init_b \cap Ind_b$, i.e., $IndInv_b = Sat(\mathcal{A})$. By Proposition 14, $\overline{IndInv_b}$ is recognized by a NFA with $2^K b$ states. ◀

4 Deciding $\mathit{IndInv}_1 \cap \mathcal{U} = \emptyset$ is PSPACE-complete

Given an instance $\mathcal{R}, \mathcal{U}$ of the safety verification problem and $b \geq 0$, if the set IndInv_b satisfies $\mathit{IndInv}_b \cap \mathcal{L}(\mathcal{U}) = \emptyset$, then $\mathcal{R}$ is safe. By Theorem 15, deciding whether $\mathit{IndInv}_b \cap \mathcal{L}(\mathcal{U}) = \emptyset$ holds is in EXSPACE for every fixed b. Indeed, the theorem and its proof show that there is a NFA recognizing $\mathit{IndInv}_b \cap \mathcal{L}(\mathcal{U})$ such that one can guess an accepting path of it, state by state, using exponential space. We do not know if there is a b such that the problem is EXSPACE-complete for every $b' \geq b$. In this section we show that for $b = 1$ the problem is actually PSPACE-complete.

4.1 Deciding $\mathit{IndInv}_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$ is in PSPACE

We give a non-deterministic polynomial space algorithm that decides $\mathit{IndInv}_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$. As a byproduct, we show that IndInv_1 is recognized by a NFA with a single exponential number of states. (Notice that we proved this for $\overline{\mathit{IndInv}_1}$ in Proposition 13, but not for IndInv_1.) All missing proofs and full versions of proof sketches can be found in the appendices of [21].

1-formulas have a special property: since the disjunction of two clauses is again a clause, the disjunction of two 1-formulas is again a 1-formula. This allows us to define the *separator* of a configuration w.

▶ **Definition 16.** *The* separator *of a configuration w, denoted Sep_w, is the union of all inductive* 1*-sets* not *containing w.*

We characterize membership of w in IndInv_1 in terms of its separator:

▶ **Lemma 17.** *For every configuration w, its separator Sep_w is an inductive 1-set. Further $w \in \mathit{IndInv}_1$ iff Sep_w is not an invariant.*

Proof. Since inductive sets are closed under union, Sep_w is inductive. Since the disjunction of two clauses is again a clause, the union of two 1-sets of configurations is also a 1-set, and so Sep_w is an inductive 1-set. For the last part, we prove that $w \notin \mathit{IndInv}_1$ iff Sep_w is an invariant. Assume first $w \notin \mathit{IndInv}_1$. Then some inductive 1-invariant does not contain w. Since, by definition, Sep_w contains this invariant, Sep_w is also an invariant. Assume now that Sep_w is an invariant. Then Sep_w is an inductive 1-invariant, and so $\mathit{Sep}_w \supseteq \mathit{IndInv}_1$. Since $w \notin \mathit{Sep}_w$, we get $w \notin \mathit{IndInv}_1$. ◀

Our plan for the rest of the section is as follows:

- We introduce the notion of a separation table for a configuration. (Definition 18)
- We show that, given a configuration w and a separation table τ for w, we can construct a 1-formula $\varphi_{\mathit{Sep}_w^\tau}$ such that $\mathcal{L}(\varphi_{\mathit{Sep}_w^\tau}) = \mathit{Sep}_w$. (Lemma 19)
- We use this result to define a transducer T_{sep} over $\Sigma \times 2^\Sigma$ that accepts a word $\langle w, \varphi \rangle$ iff $\varphi = \varphi_{\mathit{Sep}_w^\tau}$. (Proposition 21)
- We use T_{sep} and Proposition 12 to define a NFA over Σ that accepts a configuration w iff Sep_w is not an invariant, and so, by Lemma 17, iff $w \in \mathit{IndInv}_1$. (Theorem 22)

We present a characterization of Sep_w in terms of *tables*. Given a transition $s \rightsquigarrow t$, we call s and t the *source* and *target* of the transition, respectively. A *table* of length ℓ is a sequence $\tau = s_1 \rightsquigarrow t_1, \ldots, s_n \rightsquigarrow t_n$ of transitions of $\mathcal{R}$ (not necessarily distinct), all of length ℓ.[1] We define the *separation tables* of a configuration w.

[1] We call it table because we visualize $s_1, t_1, \ldots, s_n, t_n$ as a matrix with $2n$ rows and ℓ columns.

▶ **Definition 18.** *Let w be a configuration and let $\tau = s_1 \rightsquigarrow t_1, \dots, s_n \rightsquigarrow t_n$ be a table, both of length ℓ. For every $j \in [1, \ell]$, let $In(w, \tau)[j] = \{w[j], s_1[j], \dots, s_n[j]\}$ be the set of letters at position j of w and of the source configurations of the table.*

- *τ is* consistent with *w if for every $i \in [1, n], j \in [1, \ell]$, either $t_i[j] = w[j]$ or $t_i[j] = s_{i'}[j]$ for some $i' < i$.*
  (Intuitively: τ is consistent with w if for every position, the letter of the target is either the letter of w, or the letter of some earlier source.)
- *τ is* complete for *w if every table τ $(s \rightsquigarrow t)$ consistent with w satisfies $s[j] \in In(w, \tau)$ for every $j \in [1, \ell]$.*
  (Intuitively: τ is complete for w if it cannot be extended by a transition that maintains consistency and introduces a new letter.)

A table is a separation table of *w if it is consistent with and complete for w.*

Observe that every configuration w has at least one separation table. If there are no transitions with target w, then the empty table with no transitions is a separation table. Otherwise, starting with any transition $s \rightsquigarrow w$, we repeatedly add a transition, maintaining consistency and introducing at least one new letter, until no such transition exists. This procedure terminates – there are only finitely many transitions between configurations of a fixed length – and yields a separation table.

The next lemma shows how to compute a 1-formula $\varphi_{Sep^\tau_w}$ such that $\mathcal{L}(\varphi_{Sep^\tau_w}) = Sep_w$ from *any* separation table τ of w.

▶ **Lemma 19.** *Let τ be any separation table for a configuration w of length ℓ. Then Sep_w is the set of all configurations $z \in \Sigma^\ell$ such that $z[j] \notin In(w, \tau)[j]$ for some $j \in [1, \ell]$. In particular, Sep_w is the language of the 1-formula*

$$\varphi_{Sep^\tau_w} := \bigvee_{j=1}^{\ell} \left(\bigvee_{a \notin In(w,\tau)[j]} a_{j:\ell} \right)$$

or, in the powerword encoding, of the formula

$$\varphi_{Sep^\tau_w} = \overline{In(w, \tau)[1]} \cdots \overline{In(w, \tau)[\ell]} \ .$$

Proof. It follows easily from the definitions that $\varphi_{Sep^\tau_w}$ denotes an inductive 1-set not containing w. To prove that it is the largest such set it suffices to show that for every $j \in [1, \ell]$ and every letter $x \in In(w, \tau)[j]$, the configuration w belongs to every inductive 1-set specified by a powerword containing x at position j. For this, we consider the tables $\tau_0, \tau_1, \dots, \tau_n = \tau$, where τ_i is the prefix of τ of length i, and prove by induction on k that the property holds for every τ_k. ◀

We construct a transducer over the alphabet $\Sigma \times 2^\Sigma$ that transduces a configuration w into the formula $\varphi_{Sep^\tau_w}$ of a table τ consistent with and complete for w. For this we need the consistency and completeness summaries of a table.

▶ **Definition 20.** *Let $\tau = s_1 \rightsquigarrow t_1, \dots, s_n \rightsquigarrow t_n$ be a separation table for a configuration w. The* consistency summary *is the result of applying the following procedure to τ:*

- *Replace each row $s_i \rightsquigarrow v_i$ by the sequence of states of an accepting run of $\mathcal{A}_T$ on it.*
  (This produces a table with i rows and $\ell + 1$ columns, whose entries are states of $\mathcal{A}_T$.)
- *In each column, keep the first occurrence of each state, removing the rest.*
  (The result is a sequence of columns of possibly different lengths.)

The completeness summary *is the sequence* $(Q_0, Q'_0), (Q_1, Q'_1) \dots (Q_\ell, Q'_\ell)$ *of pairs of sets of states of* $\mathcal{A}_T$*, defined inductively as follows for every* $j \in [0, \ell]$*:*

- Q_0 *is the set of initial states and* Q'_0 *is empty.*
- Q_{j+1} *is the set of states reachable from* Q_j *by means of letters* $[a, b]$ *such that* $b \in \mathit{In}(w, \tau)$*.*
- Q'_{j+1} *is the set of states reachable from* Q'_j *by means of letters* $[a, b]$ *such that* $b \in \mathit{In}(w, \tau)$*, or reachable from* Q_j *by means of letters* $[a, b]$ *such that* $a \notin \mathit{In}(w, \tau)$ *and* $b \in \mathit{In}(w, \tau)$*.*

Observe that the consistency summary is a sequence $\alpha = \alpha[1] \dots \alpha[\ell]$, where $\alpha[i]$ is a sequence of *distinct* states of $\mathcal{A}_T$, i.e., an element of $Q_T^{n_T}$, and the completeness summary is a sequence $\beta = \beta[1] \dots \beta[\ell]$, where $\beta[i]$ is a pair of sets of states of $\mathcal{A}_T$, i.e, an element of $2^{Q_T} \times 2^{Q_T}$. We prove in Appendix C of [21]:

▶ **Proposition 21.** *There exists a transducer* T_{sep} *over the alphabet* $\Sigma \times 2^\Sigma$ *satisfying the following properties:*

- *The states of* T_{sep} *are elements of* $(Q_T \cup \{\square\})^{n_T} \times (2^{Q_T} \times 2^{Q_T})$*, where* n_T *is the number of states of* $\mathcal{A}_T$*.*
- *There is a polynomial time algorithm that, given two states* q, q' *of* T_{sep} *and a letter* $\langle a, X \rangle \in \Sigma \times 2^\Sigma$ *decides whether the triple* $(q, \langle a, X \rangle, q')$ *is a transition of* T_{sep}*.*
- T_{sep} *recognizes a word* $\langle w, \varphi \rangle$ *over* $\Sigma \times 2^\Sigma$ *iff* $\varphi = \varphi_{\mathit{Sep}_w^\tau}$*.*

We can now use Theorem 12 to obtain our main result:

▶ **Theorem 22.** *Let* $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T \rangle$ *be a RTS. There exists a NFA* $\mathcal{A}_1$ *over* Σ *satisfying the following properties:*

- *The states of* $\mathcal{A}_1$ *are elements of* $Q_T^{n_T} \times (2^{Q_T} \times 2^{Q_T}) \times Q_I$*.*
- *There is a polynomial time algorithm that, given two states* q, q' *of* $\mathcal{A}_1$ *and a letter* $a \in \Sigma$ *decides whether the triple* (q, a, q') *is a transition of* T_{sep}*.*
- $\mathcal{L}(\mathcal{A}_1) = \mathit{IndInv}_1$

Proof. Let T_{sep} be the transducer over the alphabet $\Sigma \times 2^\Sigma$ of Proposition 21. Let $\mathcal{A}_{\overline{init}}$ be a NFA recognizing $\overline{\mathit{Init}_1}$, i.e, the language of all 1-formulas satisfied by all initial configurations, or, in other words, all 1-formulas that are invariants. By Lemma 17, $w \in \mathit{IndInv}_1$ iff there exists a 1-formula φ such that $\langle w, \varphi \rangle \in \mathcal{L}(T_{sep})$ and $\varphi \in \mathcal{L}(\mathcal{A}_{\overline{init}})$. So there exists a NFA $\mathcal{A}_1$ for IndInv_1 whose states are the pairs $\langle q, r \rangle$ such that q is a state of T_{sep} and r a state of $\mathcal{A}_{\overline{init}}$. Since, by Proposition 12, $\mathcal{A}_{\overline{init}}$ has the same states as $\mathcal{A}_I$, the result follows. ◀

Observe that a state of $\mathcal{A}_1$ can be stored using space linear in $\mathcal{A}_I$ and $\mathcal{A}_T$.

▶ **Corollary 23.** *Deciding* $\mathit{IndInv}_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$ *is in PSPACE.*

Proof. Guess a configuration w and an accepting run of $\mathcal{A}_1$ and $\mathcal{U}$ on w, step by step. By Proposition 21, this can be done in polynomial space. Apply then NPSPACE = PSPACE. ◀

4.2 Deciding $\mathit{IndInv}_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$ is PSPACE-hard

Given a linearly bounded Turing machine, we construct a sequence $\mathcal{R}_n$ of RTSs such that the instance of $\mathcal{R}_n$ for some length $\Theta(t \times n)$ simulates the Turing machine on inputs of length n up to t steps. Moreover, we show that, for this RTS, $\mathit{IndInv}_1 = \mathit{Reach}(\mathcal{R}_n)$ holds. Therefore, we can reduce acceptance of the Turing machine to our problem (see Appendix C.1 of [21]).

▶ **Lemma 24.** *Deciding* $\mathit{IndInv}_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$ *is PSPACE-hard.*

System	$\lvert\mathcal{L}_I\rvert$	$\lvert\mathcal{L}_T\rvert$	$\lvert IndInv_1\rvert$	Properties		time (ms)
Bakery	5	9	8	deadlock	✓	< 1
				mutual exclusion	✓	
Burns	5	9	6	deadlock	✓	< 1
				mutual exclusion	✓	
Dijkstra	6	24	22	deadlock	✓	1920
				mutual exclusion	✓	
Dijkstra (ring)	6	17	17	deadlock	✓	2
				mutual exclusion	×	
D. cryptographers	6	69	11	one cryptographer paid	✓	5
				no cryptographer paid	✓	
Herman, linear	6	7	6	deadlock	×	< 1
				at least one token	✓	
Herman, ring	6	7	7	deadlock	✓	< 1
				at least one token	✓	
Israeli-Jafon	6	21	7	deadlock	✓	< 1
				at least one token	✓	
Token passing	6	7	7	at most one token	✓	< 1
Lehmann-Rabin	5	15	13	deadlock	✓	1
LR phils.	6	14	15	deadlock	×	2
LR phils.(with b_ℓ and b_r)	5	14	9	deadlock	✓	1
Atomic D. phil.	5	12	20	deadlock	✓	5
Mux array	6	7	8	deadlock	✓	< 1
				mutual exclusion	×	
Res. allocator	5	9	8	deadlock	✓	< 1
				mutual exclusion	×	
Berkeley	5	19	9	deadlock	✓	1
				custom properties	2/3	
Dragon	5	26	11	deadlock	✓	3
				custom properties	6/7	
Firefly	5	18	7	deadlock	✓	1
				custom properties	0/4	
Illinois	5	25	14	deadlock	✓	1
				custom properties	0/2	
MESI	5	13	7	deadlock	✓	< 1
				custom properties	2/2	
MOESI	5	13	10	deadlock	✓	1
				custom properties	7/7	
Synapse	5	16	7	deadlock	✓	1
				custom properties	2/2	

Figure 1 Experimental results of using $IndInv_1$ as abstraction of the set of reachable configurations.

5 How large must the bound b be?

The index b needed to prove a property (i.e., the least b such that $IndInv_b$ implies the property) can be seen as a measure of how difficult it is for a human to understand the proof. We use the experimental setup of [12, 19, 13, 20], where systems are encoded as WS1S formulas and MONA [17] is used as computation engine, to show that $b = 1$ is enough for a substantial number of benchmarks used in the RMC literature. Note that our goal is to evaluate the complexity of invariants needed for systems from diverse domains, not to present a tool ready to verify complex systems.

Our set of benchmarks consists of problems studied in [14, 3, 12, 19, 13, 20]. In a first step, we use MONA to construct a minimal DFA for $IndInv_1$. For this, we write a WS1S formula $\Psi_1(w)$ expressing that, for every 1-formula φ, if φ is an inductive invariant, then w satisfies φ. MONA yields a minimal DFA for the configurations w satisfying Ψ_1, which is precisely $IndInv_1$. We then construct the formula $\Psi_1(w) \wedge Unsafe(w)$, and use MONA to check if it is satisfiable[2]. All files containing the MONA formulas and the results are provided in [18]. The results are shown in Figure 1. The first column gives the name of the example.

[2] The second formula $\Psi_1(w) \wedge Unsafe(w)$ being unsatisfiable suffices for verification purposes, but we use $\Psi_1(w)$ to obtain information on the size of the minimal DFA.

In the second and third column we give the number of states of the minimal DFA for $\mathcal{L}_I$ and $\mathcal{L}_T$, which we also compute via `MONA`. In the next column we give the size of the minimal DFA for IndInv_1. The fifth column reports whether a property is implies by IndInv_1 (indicated by ✓) or not (indicated by $\times$). For the cache coherence protocols we replace ✓ with $^k/_m$ to state that k of m custom safety properties can be established. The last column gives the total time[3]. As we can see, IndInv_1 is strong enough to satisfy 46 out of 57 properties.

In a second step, we have studied some of the cases in which IndInv_1 is not strong enough. Direct computation of the automaton for IndInv_2 from a formula $\Psi_2(w)$ using `MONA` fails. (A direct computation based on the automata construction of Section 3 might yield better results, and will be part of our future work.) Using a combination of the automatic invariant computation method of [12, 19, 13, 20] and manual inspection of the returned invariants, we can report some results for some examples.

Examples for IndInv_b with $b > 1$. Table 1 contains two versions of the dining philosophers in which philosophers take one fork at a time. All philosophers but one are right-handed, i.e., take their right fork first, and the remaining philosopher is left handed. If the forks "know" which philosopher has taken them (i.e., if they have states b_ℓ and b_r indicating that the left or the right philosopher has the fork), then deadlock-freedom can be proved using IndInv_1. If the states of the forks are just "free" and "busy", then proving deadlock-freedom requires IndInv_3, and in fact $\mathit{Reach} = \mathit{IndInv}_3$ holds. We show how to establish this using the technique of [20] and some additional reasoning in Appendix A of [21].

The Berkeley and Dragon cache coherence protocols are considered as parameterized system in [16]. For both examples IndInv_1 is too coarse to establish all desired consistency assertions. In Appendix B we describe the formalization of both examples and show that IndInv_2 suffices to obtain the missing assertions.

6 Conclusion

We have introduced a regular model checking paradigm that approaches the set of reachable configurations from above. As already observed in [3, 14], such an approach does not require widening or acceleration techniques, as is the case when approaching from below. The main novelty with respect to [3, 14] is the discovery of a natural sequence of regular invariants converging to the set of reachable configurations.

Our new paradigm raises several questions. The first one is the exact computational complexity of checking emptiness of the intersection IndInv_b and the unsafe configurations. We have shown PSPACE-completeness for $b = 1$, and we conjecture that the problem is already EXPSPACE-complete for all $b \geq 2$. We also think that the CEGAR techniques used in [20, 19] can be extended to the RMC setting, allowing one to compute intermediate regular invariants between IndInv_b and IndInv_{b+1}. Another interesting research venue is the combination with acceleration or widening techniques, and the application of learning algorithms, like the one of [14]. Currently these techniques try to compute some inductive regular invariant, or perhaps one described by small automata, which may lead to invariants difficult to interpret by humans. A better approach might be to stratify the search, looking first for invariants for small values of b.

[3] As reported by `MONA`.

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, Giorgio Delzanno, Noomene Ben Henda, and Ahmed Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 721–736. Springer, 2007.

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 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.

8 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.

9 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.

10 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.

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

12 Marius Bozga, Javier Esparza, Radu Iosif, Joseph Sifakis, and Christoph Welzel. Structural invariants for the verification of systems with parameterized architectures. In *TACAS (1)*, volume 12078 of *Lecture Notes in Computer Science*, pages 228–246. Springer, 2020.

13 Marius Bozga, Radu Iosif, and Joseph Sifakis. Checking deadlock-freedom of parametric component-based systems. *J. Log. Algebraic Methods Program.*, 119:100621, 2021.

14 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.

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

16 Giorgio Delzanno. Constraint-based verification of parameterized cache coherence protocols. *Formal Methods Syst. Des.*, 23(3):257–301, 2003.

17 Jacob Elgaard, Nils Klarlund, and Anders Møller. MONA 1.x: new techniques for WS1S and WS2S. In *Proc. 10th International Conference on Computer-Aided Verification, CAV '98*, volume 1427 of *LNCS*, pages 516–520. Springer-Verlag, June/July 1998.

18 Javier Esparza, Mikhail Raskin, and Christoph Welzel. Repository of examples. `https://doi.org/10.5281/zenodo.6483615`, April 2022. `doi:10.5281/zenodo.6483615`.

19 Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. Abduction of trap invariants in parameterized systems. In *GandALF*, volume 346 of *EPTCS*, pages 1–17, 2021.

20 Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. Computing parameterized invariants of parameterized petri nets. In *Petri Nets*, volume 12734 of *Lecture Notes in Computer Science*, pages 141–163. Springer, 2021.

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

22 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.

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

24 Kenneth L. McMillan and Oded Padon. Ivy: A multi-modal verification tool for distributed algorithms. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 190–202. Springer, 2020.

25 Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. Ivy: safety verification by interactive generalization. In *PLDI*, pages 614–630. ACM, 2016.

26 Amir Pnueli, Sitvanit Ruah, and Lenore D. Zuck. Automatic deductive verification with invisible invariants. In *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2001.

A Dining philosophers with one left-handed philosopher

We sketch the formalization of the case in which the states of the forks are only "free" and "busy". Consider a RTS with $\Sigma = \{f, b, t, h, e\}$. The state h represents a philosophers who already grabbed the first fork and waits for the second one. All other states are used as before. The philosopher at index 1 takes first the fork at index 2 and then the fork at index n, while any other philosopher $i > 1$ first takes the fork at index $i - 1$ and then the fork at index $i + 1$.

In [20] the absence of deadlocks in this example is shown via only a few inductive assertions. These assertions can be equivalently expressed as 3-invariants. Moreover, these assertions are actually enough to completely characterize *Reach* in this example. To this end, observe that, analogously to Example 2, *Reach* is completely characterized by the absence of a few *invalid* patterns. These patterns separate into three cases: First, a philosopher should use some fork, but this fork is still considered free. Second, two philosophers are in states that require the same fork. Third, no adjacent philosopher currently uses some fork, yet this fork is busy. More formally, we get

- $\Sigma\ (\Sigma\ \Sigma)^*\ f\ (h \mid e)\ \Sigma\ (\Sigma\ \Sigma)^*$, $(\Sigma\ \Sigma)^+\ e\ f\ (\Sigma\ \Sigma)^*$, $(e \mid h)\ f\ (\Sigma\ \Sigma)^*$, $e\,(\Sigma\ \Sigma)^*\ f$,
- $(\Sigma\ \Sigma)^+\ e\,\Sigma\ (h \mid e)\ \Sigma\ (\Sigma\ \Sigma)^*$, $(e \mid h)\ \Sigma\ (e \mid h)\ \Sigma\ (\Sigma\ \Sigma)^*$, $e\ \Sigma\ (\Sigma\ \Sigma)^*\ e\ \Sigma$,
- $t\ b\ t\ \Sigma\ (\Sigma\ \Sigma)^*$, $(t \mid h)\ \Sigma\ (\Sigma\ \Sigma)^*\ (t \mid h)\ b$, $(\Sigma\ \Sigma)^+\ (t \mid h)\ b\ t\ \Sigma\ (\Sigma\ \Sigma)^*$,

The absence of these patterns can be established with the following languages of inductive 1-invariants and inductive 3-invariants:

$$[\{e\}]\ [\emptyset]\ ([\emptyset]\ [\emptyset])^*\ [\{e\}]\ [\{f\}] \qquad \begin{bmatrix}\{t,h\}\\\{t,h\}\\\emptyset\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix} \left(\begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix}\right)^* \begin{bmatrix}\emptyset\\\{t,h\}\\\{t,h\}\end{bmatrix} \begin{bmatrix}\{b\}\\\emptyset\\\{b\}\end{bmatrix}$$

$$[\{e,h\}]\ [\{f\}]\ [\{e,h\}]\ [\emptyset]\ ([\emptyset]\ [\emptyset])^* \qquad \begin{bmatrix}\{t\}\\\{t\}\\\emptyset\end{bmatrix} \begin{bmatrix}\{b\}\\\emptyset\\\{b\}\end{bmatrix} \begin{bmatrix}\emptyset\\\{t\}\\\{t\}\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix} \left(\begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix}\right)^*$$

$$[\emptyset]\ [\emptyset]\ ([\emptyset]\ [\emptyset])^*\ [\{e\}]\ [\{f\}]\ [\{e,h\}]\ [\emptyset]\ ([\emptyset]\ [\emptyset])^* \quad \begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix} \left(\begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix}\right)^* \begin{bmatrix}\{t,h\}\\\{t,h\}\\\emptyset\end{bmatrix} \begin{bmatrix}\emptyset\\\{b\}\\\{b\}\end{bmatrix} \begin{bmatrix}\{t\}\\\emptyset\\\{t\}\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix} \left(\begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\\\emptyset\end{bmatrix}\right)^*$$

Consequently, *IndInv*$_3$ and *Reach* coincide for this example. This immediately implies that *IndInv*$_3$ proves deadlock-freedom since the system actually is deadlock-free.

However, *IndInv*$_2$ is insufficient to prove deadlock-freedom: assume there exists some inductive 2-invariant I that invalidates that $D = h\ b\ t\ f\ e\ b$ can be reached. Then, I must separate all elements from *Reach* and all configurations D' with $D' \rightsquigarrow^* D$ because it is inductive. In particular, $D' = t\ b\ e\ b\ e\ b$ and the reachable configuration $t\ b\ h\ b\ e\ b$. Hence, one clause of I contains $h_{3:6}$. Consider the following pair of configurations: $D'' = t\ f\ h\ f\ t\ f$

and $C = t\ b\ h\ f\ t\ f$. I must separate D'' from C since $D'' \leadsto^* D$ while $C \in \mathit{Reach}$. Since $D'' \models h_{3:6}$ this separation is based on the second clause of I which must contain $b_{2:6}$. This means $t\ b\ h\ f\ e\ b \models I$. Since I is inductive and $t\ b\ h\ f\ e\ b \leadsto t\ b\ e\ b\ e\ b \leadsto t\ f\ t\ f\ e\ b \leadsto h\ b\ t\ f\ e\ b = D$ the assumption that I exists is folly. Consequently, D cannot be excluded via inductive 2-invariants.

B *IndInv*$_2$ for cache coherence protocols Berkeley and Dragon

For both protocols we follow the specification of [16].

Berkeley

In the Berkeley cache coherence protocol, each cell is in one of four different states: invalid (i), unowned (u), exclusive (e), and shared (s). Initially, all cells are invalid. Consequently, the language of initial configurations is i^*. For the transitions we consider a few different events. The first one is that the memory is read and the corresponding cell does provide some value of it; i.e., the cell is *not* in the state i. In this case nothing changes:

$$\left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\u\end{bmatrix} \mid \begin{bmatrix}e\\e\end{bmatrix} \mid \begin{bmatrix}s\\s\end{bmatrix}\right)^* \left(\begin{bmatrix}u\\u\end{bmatrix} \mid \begin{bmatrix}e\\e\end{bmatrix} \mid \begin{bmatrix}s\\s\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\u\end{bmatrix} \mid \begin{bmatrix}e\\e\end{bmatrix} \mid \begin{bmatrix}s\\s\end{bmatrix}\right)^*$$

If, on the other hand, a value is read from some cell that is in state i, then this memory cell fetches the information without claiming ownership; i.e., moves into the state u. Every other memory cell observes this process. Thus, cells that previously were in e move to s to account for the fact that another memory cell holds the same information.

$$\left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\u\end{bmatrix} \mid \begin{bmatrix}e\\s\end{bmatrix} \mid \begin{bmatrix}s\\s\end{bmatrix}\right)^* \left(\begin{bmatrix}i\\u\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\u\end{bmatrix} \mid \begin{bmatrix}e\\s\end{bmatrix} \mid \begin{bmatrix}s\\s\end{bmatrix}\right)^*$$

If a value is written to a cell that was invalid before, then this cell claims exclusive ownership; that is, all other cells are invalidated.

$$\left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\i\end{bmatrix} \mid \begin{bmatrix}e\\i\end{bmatrix} \mid \begin{bmatrix}s\\i\end{bmatrix}\right)^* \left(\begin{bmatrix}i\\e\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\i\end{bmatrix} \mid \begin{bmatrix}e\\i\end{bmatrix} \mid \begin{bmatrix}s\\i\end{bmatrix}\right)^*$$

If a cell already has exclusive ownership of this information there is nothing to be done. If the cell has only shared ownership of the value, all other cells *that claim shared ownership* are invalidated.

$$\left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\i\end{bmatrix} \mid \begin{bmatrix}e\\e\end{bmatrix} \mid \begin{bmatrix}s\\i\end{bmatrix}\right)^* \left(\begin{bmatrix}u\\e\end{bmatrix} \mid \begin{bmatrix}s\\e\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\i\end{bmatrix} \mid \begin{bmatrix}e\\e\end{bmatrix} \mid \begin{bmatrix}s\\i\end{bmatrix}\right)^*$$

Finally, the cache can decide to drop data at any moment in time. Thus, any cell might move into the state i.

$$\left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\u\end{bmatrix} \mid \begin{bmatrix}e\\e\end{bmatrix} \mid \begin{bmatrix}s\\s\end{bmatrix}\right)^* \left(\begin{bmatrix}u\\i\end{bmatrix} \mid \begin{bmatrix}e\\i\end{bmatrix} \mid \begin{bmatrix}s\\i\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix} \mid \begin{bmatrix}u\\u\end{bmatrix} \mid \begin{bmatrix}e\\e\end{bmatrix} \mid \begin{bmatrix}s\\s\end{bmatrix}\right)^*$$

We pose now the question whether a configuration can be reached where two different cells claiming exclusive access to some data. The corresponding set $\mathcal{U}$ corresponds to $\Sigma^*\ e\ \Sigma^*\ e\ \Sigma^*$. As shown in Table 1, *IndInv*$_1$ does not prove this property. Let us see why. Assume there is an inductive 1-invariant I which invalidates the bad word $b = e\ e$; that is, $b \not\models I$. Observe now that we can reach b in one step from $b' = u\ e$, and $b'' = e\ u$. Consequently, I cannot be satisfied by b' or b'' either. Otherwise, since I is inductive, we already get $b \models I$. This means, I must not contain $e_{1:2}$, $e_{2:2}$, $u_{1:2}$, or $u_{2:2}$. This, however, makes I unsatisfiable for the actually reachable configuration $u\ u$.

Using an adapted version of the semi-automatic approach of [20] and some additional reasoning led us to the following language of inductive 2-invariants which exclude all configurations from $\mathcal{U}$:

$$\begin{bmatrix}\emptyset\\\emptyset\end{bmatrix}^* \begin{bmatrix}\{i,s,u\}\\\{i\}\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\end{bmatrix}^* \begin{bmatrix}\{i\}\\\{i,s,u\}\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\end{bmatrix}^* .$$

Since IndInv_2 is the strongest inductive 2-invariant, this shows that IndInv_2 is strong enough to prove the property.

Dragon

The Dragon protocol distinguishes five states. As before, we have states for invalid cells (i), cells that maintain an exclusive copy of the data (e) and cells that have a (potentially) shared copy of the data (s). In contrast to before, the Dragon protocol does not invalidate other copies of some data when it is updated. Instead we introduce two new states which mirror e and s but, additionally, indicate that the data might have changed. We refer to these states as $\hat{e}$ and $\hat{s}$ respectively. Regardless, we initialize all cells as invalid; i.e., we have the initial language i^*.

Assume a read from a "valid" cell; that is, some cell that is not in state i. In that case, nothing changes:

$$\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^* \left(\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^* .$$

If a read occurs from an invalid cell – while all cells are invalid – the accessed cell becomes an exclusive reference:

$$\begin{bmatrix}i\\i\end{bmatrix}^* \begin{bmatrix}i\\e\end{bmatrix} \begin{bmatrix}i\\i\end{bmatrix}^* .$$

If not all cells are invalid but a read occurs for an invalid cell then this cell obtains a shared copy to the data. Moreover, all exclusive references; i.e., cells in states e or $\hat{e}$, move to their shared counterparts (s and $\hat{s}$ respectively).

$$\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{s}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^* \begin{bmatrix}i\\s\end{bmatrix} \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{s}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^* .$$

Writing a cell in state $\hat{e}$ does not change anything. On the other hand, writing a cell in state e moves that cell into the state $\hat{e}$:

$$\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^* \begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix} \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^*$$

and

$$\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^* \begin{bmatrix}e\\\hat{e}\end{bmatrix} \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^* .$$

A write-operation on a cell that is the only one in state s or $\hat{s}$ results in a change to $\hat{e}$. If there are other cells in either state, one moves two $\hat{s}$ while all others move to s.

$$\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\right)^* \begin{bmatrix}\hat{s}\\\hat{e}\end{bmatrix} \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\right)^*,$$

$$\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\right)^* \begin{bmatrix}s\\\hat{e}\end{bmatrix} \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\right)^*$$

and

$$\begin{aligned}&\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^* \left(\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^* \left(\begin{bmatrix}s\\\hat{s}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^*\\
|&\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^* \left(\begin{bmatrix}s\\\hat{s}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^* \left(\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^*.\end{aligned}$$

If a value is written to a previously invalid cell, then either this cell moves to $\hat{e}$ (assuming all other cells are i as well), while the occurrence of another cell with this value causes the written cell to become $\hat{s}$ and all other cells to move to state s.

$$\begin{bmatrix}i\\i\end{bmatrix}^* \begin{bmatrix}i\\\hat{e}\end{bmatrix} \begin{bmatrix}i\\i\end{bmatrix}^*$$

and

$$\begin{aligned}&\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^* \begin{bmatrix}i\\\hat{s}\end{bmatrix} \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^* \left(\begin{bmatrix}\hat{e}\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^*\\
|&\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^* \left(\begin{bmatrix}\hat{e}\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^* \begin{bmatrix}i\\\hat{s}\end{bmatrix} \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\s\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\s\end{bmatrix}\right)^*.\end{aligned}$$

Finally, any cell might drop its content at any point.

$$\left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^* \left(\begin{bmatrix}e\\i\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\i\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\i\end{bmatrix}\middle|\begin{bmatrix}s\\i\end{bmatrix}\right) \left(\begin{bmatrix}i\\i\end{bmatrix}\middle|\begin{bmatrix}e\\e\end{bmatrix}\middle|\begin{bmatrix}s\\s\end{bmatrix}\middle|\begin{bmatrix}\hat{e}\\\hat{e}\end{bmatrix}\middle|\begin{bmatrix}\hat{s}\\\hat{s}\end{bmatrix}\right)^*$$

We are interested now to establish that the language $\Sigma^*\ \hat{e}\ \Sigma^*\ \hat{e}\ \Sigma^*$ cannot be reached. The proof that *IndInv*$_1$ is insufficient to exclude all configurations of $\Sigma^*\ \hat{e}\ \Sigma^*\ \hat{e}\ \Sigma^*$ is straightforward: Observe that both $s\ \hat{e}$ and $\hat{e}\ s$ can reach $\hat{e}\ \hat{e}$ in one step. In consequence, analogously to the argument used for the Berkeley protocol, any inductive 1-invariant cannot distinguish between the reachable $s\ s$ and the unreachable $\hat{e}\ \hat{e}$.

On the other hand, the language

$$\begin{bmatrix}\emptyset\\\emptyset\end{bmatrix}^* \begin{bmatrix}\{\hat{s},i,s\}\\\{i\}\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\end{bmatrix}^* \begin{bmatrix}\{i\}\\\{\hat{s},i,s\}\end{bmatrix} \begin{bmatrix}\emptyset\\\emptyset\end{bmatrix}^*$$

of inductive 2-invariants induces an abstraction disjoint from $\Sigma^*\ \hat{e}\ \Sigma^*\ \hat{e}\ \Sigma^*$. Consequently, *IndInv*$_2$ does as well.