TUM School of Computation, Information and Technology
Technische Universität München

INDUCTIVE STATEMENTS FOR REGULAR TRANSITION SYSTEMS

Christoph Welzel-Mohr

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Matthias Althoff

Prüfende der Dissertation:

1. Prof. Dr. Francisco Javier Esparza Estaun
2. Prof. Dr. Anthony Widjaja Lin

Die Dissertation wurde am 26.09.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 04.03.2024 angenommen.

Für meinen Tölpel

* * *

In Erinnerung an

Prof. Dr. Bertram Wild,

dessen Liebe zum Zählen meine Frau zu einer akademischen Laufbahn inspirierte

– wo wir uns trafen.

Abstract

Regular model checking is a well-established formalism for reasoning about parameterized systems which are modeled as regular transition systems. In this thesis, we propose to analyze regular transition systems using inductive statements. A statement φ is inductive if the transition relation only relates a state v satisfying φ with states that also satisfy φ. Thus, the set of all states that satisfy φ over-approximates the set of all states reachable from v. We present a way to encode and reason about inductive statements using finite state automata – called *interpretations*.

Based on interpretations, we introduce an approach for regular model checking by using inductive statements to over-approximate all reachable states. (Because regular model checking is undecidable this approach is, necessarily, incomplete.) We provide an algorithm for this which runs, for any given interpretation, in space exponential in its input. Thus, we prove that checking safety conditions using the over-approximation induced by inductive statements is, for any interpretation, in ExpSpace.

For three specific interpretations, we prove that checking safety conditions using inductive statements is PSpace-hard. Additionally, we provide, for two of these interpretations, an algorithm that solves the problem using space polynomial in its input – rendering the problem for them PSpace-complete.

In a second step, we show how, based on automata learning, one can learn a set of inductive statements that are powerful enough to establish a given safety property. We do so to improve the performance of the approach and to provide certificates of a reasonable size for established properties. Additionally, we consider how to speed up this learning process for parameterized systems with specific communication topologies.

All these approaches are implemented in our tool `dodo`, which we evaluate on a set of common examples for parameterized verification.

Übersicht

Reguläre Modell-Verifikation analysiert parametrisierte Systeme, welche als reguläre Transitionssysteme beschrieben werden.

In dieser Arbeit beschreiben wir wie man die erreichbaren Zustände eines reguläre Transitionssystems mit Hilfe induktiver logischer Aussagen abschätzen kann. Hierbei ist eine Aussage φ induktiv, wenn man von einem Zustand v, welcher die Aussage erfüllt, in einem Schritt lediglich Zustände erreichen kann, die die Aussage φ auch erfüllen. Demnach kam man vom Zustand v aus mit beliebig vielen Schritten stets nur solche Zustände erreichen, die auch φ erfüllen. Damit ist die Menge der Zustände, die φ erfüllen, eine sichere Abschätzung all jener Zustände, die von v überhaupt erreicht werden können. Im Folgenden führen wir einen Formalismus ein, der mittels endlicher Automaten, die wir *Interpretationen* nennen, die Erfüllbarkeitsrelation für diese logischen Aussagen beschreibt.

Dies erlaubt uns einen Ansatz für reguläre Modell-Verifikation zu präsentieren, der aufgrund der Unentscheidbarkeit des Problems notwendigerweise unvollständig ist. Unser Algorithmus benötigt exponentiell viel Speicher relativ zur Eingabe und zeigt damit, dass die Analyse regulärer Transitionssysteme mit Hilfe von induktiven Aussagen in der Komplexitätsklasse ExpSpace liegt.

Wir stellen uns die Frage, ob induktive Aussagen ausreichen, um eine Eigenschaft des Systems zu beweisen, für drei konkrete Interpretationen. Für alle drei von diesen zeigt sich, dass das Problem nun PSpace-schwer ist. Für zwei Interpretationen finden wir einen Algorithmus, welcher das Problem mit polynomiell viel Speicher relativ zur Eingabe löst. Demnach ist das Problem für diese Interpretationen PSpace-vollständig.

Mithilfe eines Verfahrens zum Lernen von endlichen Automaten ist es möglich lediglich für das Verifikationsproblem ausreichend viele induktive logische Aussagen zu finden. Auf diese Weise kann die Effizienz des Ansatzes verbessert und gleichzeitig eine Zertifikat für die Lösung präsentiert werden. Da viele parametrisierte Systeme sich in ihrer

Kommunikationsstruktur ähneln, zeigen wir ferner wie man diese Strukturen ausnutzen kann, um den Lernprozess weiter zu verbessern.

Schließlich diskutieren wir eine experimentelle Auswertung dieser Ansätze anhand unserer protoypischen Implementierung `dodo`.

Acknowledgements

> Look, Javier, no hands…

First, I want to thank my supervisor Javier Esparza. Our discussions and collaborations were vital to shape the ideas that ultimately led to the results of this thesis. He also provided helpful comments on drafts of this thesis. Also, Javier gave me time when I desperately needed it.

Mikhail Raskin is equally influential to the content of this thesis. Without his brilliant ideas, far fewer questions would be answered.

The seed of what would grow into this work was a collaboration with Radu Iosif, Marius Bozga, and Joseph Sifakis. For this initial spark, I am very grateful.

During the last five years, my friends and colleagues from Chair I7 were a huge support. In particular, my roommate Chana. Because Chana was, academically, only a few weeks my senior, we could support each other through the initial struggles as doctoral candidates. Due to our companionship and, especially, all the amazing memes, many problems were easier to bear.

Although promised, Bala never cooked me French toast. However, his friendship makes more than up for it. A shared love for movies was the foundation on which we built this friendship and, begrudgingly, I have to admit that discussions with Bala were extremely helpful in improving the presentation of the results in the thesis[1].

Also, I am indebted to Michael Luttenberger for enduring my tutoring for his various lectures. Although it is my impression that I still do not understand much of cryptography, stochastic games, or algebra, Michael complained only a little. Regardless, I very much enjoyed all my teaching duties and those with Michael in particular.

---

[1] So much so that I refer any reader to him that feels that the presentation is lacking.

Philipp Czerner and Christopher Hugenroth read and commented on an early draft of this thesis. Their input is greatly appreciated. Philip Offtermatt and Qais Hamarneh trusted me to advise them with their master's theses. Since both excelled in their work, this was a very rewarding experience for which I am grateful.

More personally, I want to thank my family. First and foremost my wife, Steffi. Without her unfailing support and kindness, this thesis would not have been possible. How she refrained from strangling me during times of constant complaints will remain a mystery to me, forever.

Secondly, my parents, Christiane and Thomas, and siblings, Philipp and Anne, for providing constant encouragement when it was desperately needed. Also, I want to thank my inlaws, Bettina and Wolfgang, and Sebastian, Tobias, and Torben, for welcoming me into their family.

The annual Christmas dinner with my friends, Johannes, Simon, Kerstin, Nick, Lukas, and Andreas, is a highly valued tradition and, thus, an event I look forward to the whole year. Similarly, I am very glad for my friendship with Jannik, Marjo, and Theo.

Finally, this thesis would not have been possible without the help of the accessibility software `talon` and its community. Due to some problems with repetitive strain, most of the text in this thesis was generated by speaking rather than typing. I am extremely grateful that this software allowed me to finish my work – something that appeared very doubtful at times. Since I had to learn how to use this software, the preparation of this thesis took longer than anticipated. Therefore, I gratefully acknowledge the financial support from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under grant agreement No 787367 (PaVeS) which allowed me to conduct my research and finish this thesis.

Contents

Definitions

Examples

Figures

1 Introduction

Because software systems are omnipresent, it is an ongoing endeavor to make them as reliable as possible. Although excessive testing of a software system increases confidence in it, it cannot replace a formal correctness proof. For this reason, we are interested in *formally verifying* software systems *automatically*.

For *finite* state systems *model checking* is the most established automatic verification procedure [CES09; BK08]. However, not all software systems can be modeled with finitely many states. For instance, one can consider protocols for *mutual exclusion* that should grant mutually exclusive access to some resource regardless of the number of participants in the protocol. We call these systems *parameterized*, where the parameter is the number of participants.

Such parameterized systems are the focus of this thesis. In particular, we consider systems in which each of the participants can be identified – for example, with numbers from 1 to n. This separates this model from models in which the agents are *anonymous*: Petri nets, population protocols, VASS, et cetera. Also, we do *not* consider randomized or probabilistic systems [Hon+19; Len+17; LR16].

One “important framework for infinite state model-checking” [Abd12] which is considered “elegant, simple, but powerful” [LR21] is *regular model checking*. Initially advocated in [Abd+04; Abd12; WB98; Kes+01], regular model checking was the focus of a considerable amount of research; e. g. [Abd+12; Boi12; Bou+12; BT12; DR12; Leg12; Abd+02]. At its core, regular model checking defines a (potentially infinite) transition system using a regular language and a transition relation that can be captured by a finite state automaton, and asks whether one can reach in this transition system any undesired configuration. These undesired configurations are defined by another regular language. In this way, regular model checking is used to establish safety properties for parameterized systems. This thesis contributes a novel approach to this framework.

Related research

Because there is such a large body of research on regular model checking, we do not provide an exhaustive picture of all approaches to it. Instead, we present, first, three approaches that are the main subjects of two surveys, [Abd+04] and [Abd12], for regular model checking. To do so, we begin by giving an informal definition of regular model checking. Regular model checking defines a parameterized system with the help of *regular languages*[1]. In particular, every configuration of the parameterized system is represented as a word in Σ^*. Additionally, there is one regular language $\mathcal{I} \subseteq \Sigma^*$ that describes the initial configurations of the system and a second language $\mathcal{T} \subseteq (\Sigma \times \Sigma)^*$ that describes the transition relation. More specifically, there is a transition from v to u if and only if there is a word $t \in \mathcal{T}$ such that v is the projection of every letter in t to its first component and u is the projection of every letter in t to its second component. For now, we identify the transition relation and the regular language $\mathcal{T}$. Additionally, we also introduce a composition operation $\circ$ for the transition relation. For instance, $\mathcal{T} \circ \mathcal{T}$ relates v and u if and only if there is w such that there is a transition from v to w and there is a transition from w to u. In other words, $\mathcal{T} \circ \mathcal{T}$ is the relation of doing two steps at once (which we also denote with $\mathcal{T}^2$), $\mathcal{T} \circ \mathcal{T} \circ \mathcal{T}$ is the relation of doing three steps at once (which we also denote with $\mathcal{T}^3$), and so on. Note here that the relations $\mathcal{T}^i$ are "regular" for every i; that is, there exists a finite state automaton for each of these relations. The task of regular model checking is to establish that there is no initial configuration $v \in \mathcal{I}$ that can be related to some undesired configuration u in $\mathcal{T}^+ = \bigcup_{i>0} \mathcal{T}^i$ where the set of undesired configurations is another regular language $\mathcal{B}$. Unfortunately, the relation $\mathcal{T}^+ = \bigcup_{i>0} \mathcal{T}^i$ is not necessarily regular anymore.

Based on this introduction, let us present three approaches for regular model checking:

Quotening This approach tries to construct a finite state automaton for the relation $\mathcal{T}^+$. Conceptionally, the idea is to start with the automaton for $\mathcal{T}$. This automaton is then modified to accept the relation $\mathcal{T} \cup \mathcal{T}^2$, $\mathcal{T} \cup \mathcal{T}^2 \cup \mathcal{T}^3$, and so on. To do so, the states of the automaton are columns of the states of the automaton for $\mathcal{T}$. A run $c_0 \ \ldots c_m$ in this automaton encodes multiple runs, say n, of the automaton for the relation $\mathcal{T}$. This means, there are n transitions (one for each run) $t_1, \ldots, t_n$ that are executed successively. Specifically, for any $0 \leq j \leq m$, the state c_j is a

---

[1] We assume a basic familiarity with the concept of regular languages. For more formal definitions, we refer the reader to Section 2.1.

column of n states from the automaton for the relation $\mathcal{T}$, and projecting to the i-th element of these columns yields an accepting run for t_i.

This construction never terminates. The idea is to identify repetitions in columns that can be repeated arbitrarily. For instance, consider a state q_L in the automaton for the relation $\mathcal{T}$ which can only be reached by using letters from $\{\langle v, v\rangle : v \in \Sigma\}$. In other words, the state q_L can only be reached in a run on a transition t after a prefix t' such that t' does *not* change any letter. Such a state is called *left-copying*. One can consider a second (symmetric) notion for states: if every accepting run from some state q_R only uses letters from $\{\langle v, v\rangle : v \in \Sigma\}$, then it is considered *right-copying*.

Intuitively speaking, applying multiple transitions that all lead to a left-copying state q_L does not have any effect on the prefix of the configuration because those transitions can only copy letters in this prefix. Based on this observation, one can introduce an equivalence relation that considers columns the same if they are the same after every uninterrupted sequence of the same left-copying or right-copying state is replaced by a single occurrence of this state. Using this equivalence relation, one considers, roughly speaking, the quotient automaton of the current automaton and checks whether adding another application of $\mathcal{T}$ to this quotient automaton changes what it relates. By construction, every quotient automaton in step m recognizes a relation $\mathcal{R}$ such that $\bigcup_{0<i\leq m} \mathcal{T}^i \subseteq \mathcal{R} \subseteq \mathcal{T}^+$ – hoping to eventually reach $\mathcal{R} = \mathcal{T}^+$.

Abstraction The approach before tries to compute an automaton for the relation $\mathcal{T}^+$. For regular model checking computing this relation is sufficient but not necessary. Instead, it already suffices to compute a relation $\mathcal{R}$ which contains at least $\mathcal{T}^+$ but which does not relate any $v \in \mathcal{I}$ to any $u \in \mathcal{B}$.

Thus, one can try a similar construction as before. Now, however, one considers equivalence relations (to construct the quotient automaton) which are, potentially, more reductive. More specifically, it suffices to guarantee that the quotient automaton in every step m recognizes a relation $\mathcal{R}$ such that $\bigcup_{0<i\leq m} \mathcal{T}^i \subseteq \mathcal{R}$ but there is no guarantee anymore that $\mathcal{R} \subseteq \mathcal{T}^+$.

A core strength of this approach is that one can use $\mathcal{B}$ to inform the choice of the used equivalence relation because we only want to exclude undesired configurations but are less interested in all possible behaviors of the system.

Extrapolation This last approach is, at its core, a rule-based generalization technique for the set of all reachable words. Similar to before, one considers the relations $\mathcal{T}$, $\mathcal{T} \cup \mathcal{T}^2$, $\mathcal{T} \cup \mathcal{T}^2 \cup \mathcal{T}^3$, and so on. Here, however, we apply all these relations to the language of initial configurations $\mathcal{I}$. In this way, we get regular languages $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, and so on, which are all configurations that can be reached from some initial configuration in at most one step, at most two steps, at most three steps, and so on.

Roughly speaking, we hope to identify some regular expression Λ which corresponds to the change of executing one step. The idea is to "guess" the change after arbitrarily many steps as Λ^*. That is, if $\mathcal{A}_i$ is the language of a regular expression $\rho_1 \cdot \rho_2$ and $\mathcal{A}_{i+1}$ is the language of a regular expression $\rho_1 \cdot (\Lambda|\varepsilon) \cdot \rho_2$, then one can try to over-approximate all reachable configurations as the language of the regular expression $\rho_1 \cdot \Lambda^* \cdot \rho_2$.

Another strain of research has applied learning techniques to regular model checking [Nei14; Var06; Che+17; NJ13; Var+04]. Since we are considering a related approach later, we discuss this in more detail at the beginning of Chapter 3.

[LR21] is a recent article that is closest to our approach in spirit. Therefore, we want to discuss the content of [LR21] in more detail in the following.

You can stand under my umbrella

In [LR21], the authors propose *existential second-order logic for automatic structures* as "umbrella covering a large number of regular model checking tasks". Since we believe that we fit well under this umbrella, we want to discuss this framework.

Roughly speaking, an automatic structure is a relational[2] logical structure where all elements of the structure are words of a finite alphabet Σ and all its relations can be captured by finite state automata. For instance, consider an automatic structure $\mathfrak{A}$ over a vocabulary that contains a ternary relation symbol τ. Then, there exists a finite state automaton $\mathcal{S}$ such that

$$\langle a_1 \ \dots \ a_n, b_1 \ \dots \ b_n, c_1 \ \dots \ c_n \rangle \in \tau^{\mathfrak{A}}$$

if and only if

---

[2] That is, the vocabulary of the structure does not contain any function symbols.

$$\mathcal{S} \text{ accepts the word } \langle a_1, b_1, c_1\rangle \ \ldots \ \langle a_n, b_n, c_n\rangle$$

where $\tau^{\mathfrak{A}}$ is the interpretation of the relation symbol τ in the structure $\mathfrak{A}$[3].

This means, that any regular transition system can be understood as an automatic structure over one unary relation symbol $\mathcal{I}$ which are the initial configurations of the system, and one binary relation symbol $\mathcal{T}$ which corresponds to the transitions of the system. In any automatic structure, one can compute a finite state automaton that captures any relation that can be defined in first-order logic [Grä20]. For instance, one can define the relation $\mathcal{T} \cup \mathcal{T}^2$ in first-order logic:

$$\mathit{AtMostTwoSteps}(x, y) = \mathcal{T}(x, y) \vee (\exists z \ . \ \mathcal{T}(x, z) \wedge \mathcal{T}(z, y))\,.$$

Consequently, this shows, as previously already used, that there exists a finite state automaton for the relation $\mathcal{T} \cup \mathcal{T}^2$.

The authors of [LR21] argue that many verification problems for parameterized systems can be formulated as automatic structures. Additionally, one can describe approaches to solve these verification problems in the *existential second-order logic* for these structures. Formulas of the existential second-order logic of an automatic structure over the vocabulary $\langle \tau_1, \ldots, \tau_k\rangle$ are of the form $\exists R_1, \ldots, R_n \ . \ \varphi$ where φ is a first-order formula over the vocabulary $\langle \tau_1, \ldots, \tau_k, R_1, \ldots, R_n\rangle$. Intuitively, the existentially quantified relations $R_1, \ldots, R_n$ encode a "solution" to the verification problem. Coming back to regular model checking, recall that we want to prove that one cannot reach any undesired configuration. Thus, we can encode an instance of regular model checking as an automatic structure $\mathfrak{S} = \langle \Sigma^*, \mathcal{I}, \mathcal{T}, \mathcal{B}\rangle$ where

Σ^* is the universe of the structure,

$\mathcal{I}$ is a unary relation symbol which represents the initial configurations of the system,

$\mathcal{T}$ is a binary relation symbol which represents the transitions of the system, and

$\mathcal{B}$ is a unary relation symbol which represents the undesired configurations.

In order to prove that no undesired configurations can be reached, it suffices to find a set of configurations that contains at least all reachable configurations but no undesired

---

[3] To ease presentation, we only consider those structures where all relation symbols only relate words of the same length.

ones. This can be expressed in existential second-order logic in the following formula:

$$\begin{aligned}\psi = \exists \mathit{Safe}\ .\ \forall x\ .\ \mathcal{I}(x) \rightarrow \mathit{Safe}(x) && (1.1)\\ \wedge \forall x, y\ .\ (\mathcal{T}(x,y) \wedge \mathit{Safe}(x)) \rightarrow \mathit{Safe}(y) && (1.2)\\ \wedge \forall x\ .\ \mathcal{B}(x) \rightarrow \neg \mathit{Safe}(x). && (1.3)\end{aligned}$$

In other words, ψ states the question whether some set of configurations *Safe* exists such that

(1.1) all initial configurations are part of *Safe*,

(1.2) *Safe* is closed under the transition relation $\mathcal{T}$; that is, if we take any configuration from *Safe* and execute one step, then we can only reach a configuration from *Safe* again, and

(1.3) no undesired configuration is part of *Safe*.

Specifically, the structure $\mathfrak{S}$ satisfies the formula ψ if and only if one cannot reach any undesired configuration from an initial configuration in this regular transition system. This is because, if the structure $\mathfrak{S}$ satisfies the formula ψ, then there is a set *Safe* which satisfies the conditions (1.1), (1.2), and (1.3). This set contains at least all reachable configurations (because of (1.1) and (1.2)) but no undesired configuration (because of (1.3)). Therefore, no undesired configuration can be reached.

On the other hand, if the set of all reachable configurations does not contain any undesired configuration, then this set of all reachable configurations satisfies the conditions (1.1), (1.2), and (1.3). Therefore, the structure satisfies the formula ψ because a suitable set *Safe* exists.

Thus, regular model checking can be formulated as a satisfiability question in existential second-order logic for an automatic structure. Unfortunately, one needs to consider all possible sets of configurations for *Safe* to judge whether the formula can be satisfied or not. In fact, deciding whether such a set exists at all is undecidable [Blo+16].

To avoid checking all possible sets, the authors of [LR21] propose to consider only regular languages for *Safe*. Or, more generally, for all the existentially quantified second-order variables $R_1, \dots, R_n$ that encode a solution to the verification problem in a formula $\exists R_1, \dots, R_n\ .\ \varphi$, one only tries to find relations that can be captured by finite state automata. The benefits of this restriction are twofold: on the one hand, this is complete for many important properties, and, on the other hand, has proven practically viable because there are sophisticated enumeration techniques for finite state automata.

In contrast to the three approaches that we have discussed before, this framework considers regular model checking from a logical perspective. In this way, it is closer to the approach we want to present in this thesis. Moreover, we present, in the following, the rough structure of the thesis by formulating its content in this framework.

Structure of the thesis

Let us consider an instance of regular model checking as an automatic structure $\mathfrak{S} = \langle \Sigma^*, \mathcal{I}, \mathcal{T}, \mathcal{B} \rangle$. In this thesis, we present an approach for the analysis of regular model checking that is based on *inductive logical statements* for the regular transition system. Moreover, we automate reasoning about these logical statements in a very similar fashion as automatic structures automate reasoning about first-order logic. Specifically, we introduce a second alphabet Γ which is used to encode logical statements. That is, a logical statement is a word from Γ^*. Secondly, we introduce a binary relation $\models$ which relates configurations from Σ^* and statements from Γ^*. This relation is used to state whether a configuration u *satisfies a statement* I. Crucially, we only consider satisfiability relations $\models$ that can be captured by finite state automata; that is, there is a finite state automaton $\mathcal{V}$ over the alphabet $(\Sigma \times \Gamma)$ such that $u_1 \ \ldots \ u_n \models I_1 \ \ldots \ I_n$ if and only if $\mathcal{V}$ accepts $\langle u_1, I_1 \rangle \ \ldots \ \langle u_n, I_n \rangle$.

In this way, we capture our approach to regular model checking as an automatic structure $\mathfrak{Z} = \langle (\Sigma \cup \Gamma)^*, \sigma, \gamma, \mathcal{I}, \mathcal{T}, \mathcal{B}, \models \rangle$. Here, $(\Sigma \cup \Gamma)^*$ is the universe of this structure. However, we want to strictly separate words from Σ^* and Γ^*. Thus, we introduce two unary relation symbols σ and γ such that σ is interpreted with Σ^* and γ is interpreted with Γ^*. As before, $\mathcal{I}, \mathcal{T}, \mathcal{B}$ describe the initial configurations, the transitions, and the undesired configurations of the regular model checking instance.

In the first part of our thesis, we consider the set of all *inductive* statements. These are statements that, if $\mathcal{T}$ relates the configurations u and v and u satisfies the statement, then v satisfies the statement as well. First, we observe that the set of all inductive statements can be defined in first-order logic in the structure $\mathfrak{Z}$ as

$$\text{Inductive}(I) = \gamma(I) \land \forall x, y \ . \ (\mathcal{T}(x, y) \land x \models I) \rightarrow y \models I.$$

Observe that all inductive statements that are satisfied by some configuration u are also satisfied by all configurations that can be reached from u. In other words, if there is an inductive statement I that is satisfied by u but not by v, then v cannot be reached from u. In this way, we can define, based on inductive statements, a relation that over-approximates reachability in the regular transition system $\langle \mathcal{I}, \mathcal{T} \rangle$:

$$\mathit{PotentiallyReachable}(x, y) = \sigma(x) \wedge \sigma(y) \wedge \forall I \ . \ (\text{Inductive}(I) \wedge x \models I) \rightarrow y \models I.$$

Equipped with this over-approximation, we can check whether all inductive statements suffice to establish that no undesired configuration can be reached from an initial configuration. For this, we compute whether the structure $\mathfrak{Z}$ satisfies the formula

$$\varphi = \neg \exists x, y \ . \ \mathcal{I}(x) \wedge \mathcal{B}(y) \wedge \mathit{PotentiallyReachable}(x, y).$$

If this is the case, then we can give a positive answer for this instance of regular model checking.

In Chapter 2, we examine this approach in more detail. In particular, we show that it is possible to check whether $\mathfrak{Z}$ satisfies φ in exponential space (w. r. t. to the size of the automata that capture the relations of $\mathfrak{Z}$). Additionally, we consider three specific examples for the relation $\models$. For two of these, we show that the question of whether $\mathfrak{Z}$ satisfies φ is PSpace-complete. For the last instance of the relation $\models$, we prove that the question is PSpace-hard but we do not provide a matching upper bound. Finally, we consider common communication topologies for parameterized systems and provide, for these topologies, alternative characterizations of the sets of all inductive statements.

In the second part of the thesis, we propose that it is not necessary to consider all inductive statements but we can look for sufficiently many. For this, we check if there exists a *regular* set R which only contains inductive statements such that, for every combination of initial and undesired configuration, there is at least one inductive statement in R that is satisfied by the initial but not the undesired configuration. This question can be formulated in existential second-order logic:

$$\begin{aligned} &\exists R \ . \ \forall x \ . \ R(x) \rightarrow \text{Inductive}(x) \\ &\quad \wedge \forall x, y \ . \ (\mathcal{I}(x) \wedge \mathcal{B}(y)) \rightarrow \exists I \ . \ (R(I) \wedge x \models I \wedge \neg y \models I) \end{aligned}$$

such that we only look for regular witnesses for R.

Since the set of all inductive statements is itself a regular set, only considering regular subsets of it does not restrict the power of the approach. This is because if there is any such subset there is a regular one – at least, the set of all inductive statements itself. We formulate the search for a sufficient set of inductive statements as an instance of automata learning in Chapter 3. There, we encounter the question whether, for two given configurations $u, v \in \Sigma^*$, the structure $\mathfrak{Z}$ satisfies the formula

$$\psi = \exists x \ . \ \text{Inductive}(x) \wedge u \models x \wedge v \not\models x.$$

We analyze the complexity of this question in more detail then. Also, we discuss how to speed up this learning process by exploiting the topologies of parameterized systems.

In Chapter 4, we provide an experimental evaluation of all of these approaches based on a prototype called `dodo`.

1.1 Contribution

Previous publications

As a PhD student, the author published some relevant results for this thesis. In the following, we present a list of the relevant publications with a synopsis of their content. This thesis is preceded by four conference publications:

[Boz+20] In this publication we consider a different model. This model, however, can be embedded into regular model checking. Here, we already introduce the concept of over-approximating reachability based on inductive statements. It is, therefore, an inspiration for Chapter 2.

[ERW21b] This publication introduces the concept of learning sufficiently many inductive statements to prove safety properties for parameterized systems. Thus, this already contains ideas that lead to the results of Chapter 3. The learning procedure is not formulated in the context of automata learning but solely relies on generalizing one single inductive statement to a family of inductive statements[4]. This contribution was awarded the Best Paper Award of the conference.

[ERW21a] Here we apply the methodology of the previous publication to more complex parameterized systems. In particular, these systems cannot be embedded into

---

[4] We formulate these generalizations in Lemma 3.5.

regular model checking anymore because the state space of every single agent grows with the size of the considered instance. Consequently, its relevance for this thesis is only tangential.

[ERW22b] This publication already contains most of the ideas of this thesis. In particular, the considered model is regular model checking and we consider an over-approximated reachability relation which is induced by a particular family of interpretations. One basic member of this family is also considered in this thesis. With the introduction of interpretations in this thesis, we subsume most of its results. Additionally, we present some of its results in this thesis in an expanded form.

Three of these publications were expanded to form two additional articles:

[ERW22a] This article contains the ideas of [ERW21b] and [ERW21a]. Fundamentally, both operate with the same methodology; that is, generalizing a single inductive statement to a language of inductive statements for the whole parameterized system by exploiting the topologies of the system.

[ERW22c] This is an extended version of [ERW22b] that is currently under review for the journal "Logical Methods in Computer Science". Although interpretations were developed for this thesis, they are presented in this contribution as well because the manuscripts were written in parallel. Thus, there is some overlap of the content in this article and Chapter 2.

Contributions of this thesis

The central, to the best of our knowledge, genuinely new contribution of this thesis is the introduction of interpretations as a tool for the analysis of regular transition systems. With their introduction we subsume and streamline the results presented in [Boz+20; ERW21b; ERW22b; ERW22c]. Proving the problem whether $\mathfrak{I}$ satisfies the formula φ PSpace-hard for the interpretation $\mathcal{V}_{\mathit{flow}}$ (Theorem 2.5) is here one of the central contributions. Minor contributions are Theorem 2.3 and Theorem 2.4 because the first one essentially is folklore knowledge in the Petri net community. Also, Theorem 2.3 implies Theorem 2.4 on the basis of results from [ERW22b]. Section 2.6 does not contain new results but presents the results in a more explicit way than [ERW22b]. We believe that this presentation renders the complex construction more accessible. In Section 2.7, the results of [ERW21b] are expanded and, for the first time, formulated for regular transition systems.

With the idea to learn inductive statements to prove safety properties in the framework of automata learning of Chapter 3, we generalize the results of [ERW21b] significantly. Although the problem whether $\mathfrak{I}$ satisfies the formula ψ for two given configurations u and v already arose in [ERW21b], its complexity has not been studied there. Mikhail Raskin was the first person to show that this problem is in PTIME for $\mathcal{V}_{trap}$ and that there are interpretations for which it is NP-complete [Ras22]. Valentin Krasotin has refined and expanded on these results in his master's thesis [Kra23]. Since this master's thesis uses [ERW22c] as a basis, it already contains the concept of interpretations. Considering the interpretation $\mathcal{V}_{flow}$ for the problem whether $\mathfrak{I}$ satisfies the formula ψ for two given configurations u and v and proving it to give an NP-hard instance of this problem (Lemma 3.2) is a contribution of this thesis.

The implementation and evaluation of all approaches presented in this thesis; that is, Chapter 4, is one of the major contributions of it.

2 Inductive statements for regular transition systems

2.1 Preliminaries

In this section, we introduce some basic notions that we use throughout this thesis: *regular languages* and *regular transition systems*. Regular languages are languages that can be recognized by a finite state automaton. On the other hand, regular transition systems are a model for parametrized systems that define a language of reachable configurations with the help of two regular languages.

Finite automata

We use standard notions of finite automata. We distinguish between deterministic and non-deterministic automata to recognize *regular* languages of finite words.

Definition 2.1: *Deterministic finite automaton (DFA).*

A DFA is a quintuple $\mathcal{A} = \langle Q, q_0, \Sigma, \delta, F\rangle$ where Q is a set of *states* with $q_0 \in Q$ which we call *initial state*. Σ is a finite set of letters. We call this set the *alphabet* of $\mathcal{A}$. $\delta\colon Q \times \Sigma \to Q$ is $\mathcal{A}$s *step function* and $F \subseteq Q$ is the set of *accepting states*. For any word $u_1 \ \dots \ u_n \in \Sigma^*$ the DFA $\mathcal{A}$ provides a unique run $q_0 \ q_1 \ \dots \ q_n$ on $u_1 \ \dots \ u_n$ by setting $q_i = \delta(q_{i-1}, u_i)$ for all $1 \leq i \leq n$. This run is called *accepting* if $q_n \in F$. We say $\mathcal{A}$ *accepts* a word w if the run of $\mathcal{A}$ on w is accepting. The set of all words that $\mathcal{A}$ accepts is denoted by $\mathcal{L}(\mathcal{A})$.

For any DFA its step function provides a deterministic way of computing its run on a word. Employing non-determinism to define a set of runs on a word renders finite automata for regular languages more compact (and, sometimes, more intuitive).

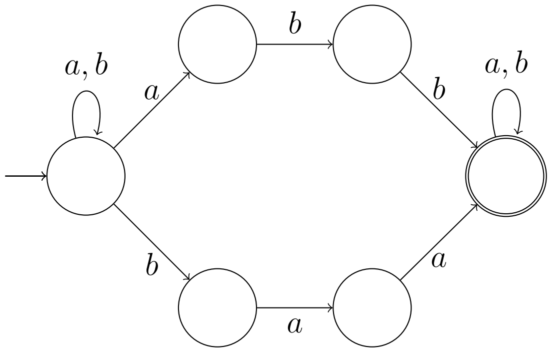
Definition 2.2: *Non-deterministic finite automaton (NFA).*

An NFA is a quintuple $\mathcal{A} = \langle Q, Q_0, \Sigma, \Delta, F\rangle$ where Q, Σ and F are as for a DFA. We replace, however, the unique initial state of a DFA with a set $Q_0 \subseteq Q$ of *initial states* and the step function with a step *relation* $\Delta \subseteq Q \times \Sigma \times Q$. We adapt the notion of *accepting run* accordingly: for any word $u_1 \;\ldots\; u_n \in \Sigma^*$ we consider a sequence of states $q_0\; q_1 \;\ldots\; q_n$ a run of $\mathcal{A}$ on $u_1 \;\ldots\; u_n$ if $q_0 \in Q_0$ and $\langle q_{i-1}, u_i, q_i\rangle \in \Delta$ for all $1 \leq i \leq n$. As before, we call this run *accepting* if $q_n \in F$. $\mathcal{A}$ *accepts* a word w if *there exists an accepting* run of $\mathcal{A}$ on w. $\mathcal{L}(\mathcal{A})$ still denotes the set of all accepted words of $\mathcal{A}$.

Throughout the thesis, we also use regular expressions to compactly denote some regular languages. Moreover, we mix regular expressions and set notations as we see fit to describe regular languages as conveniently as possible. We do not introduce regular expressions here but refer the interested reader to a standard textbook on regular languages; e. g. [HMU07].

Figure 2.1: *NFA for* $\Sigma^* \; (a\; b\; b | b\; a\; a) \; \Sigma^*$.

This automaton is constructed by simply guessing at which point of the word one of the possible patterns is read.

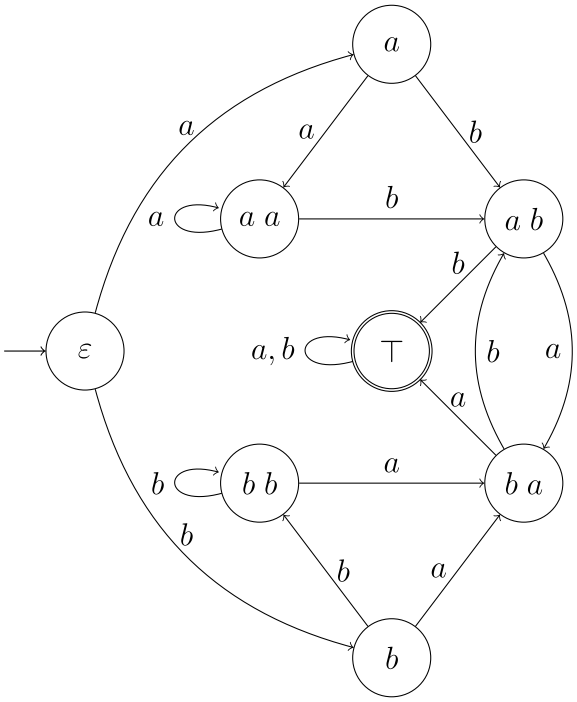


Example 2.1: *Simple regular language.*

Consider the alphabet $\Sigma = \{a, b\}$ and the language of all words in which either the pattern $a\ b\ b$ or the pattern $b\ a\ a$ occurs. This language can be described via the regular expression $\Sigma^*\ (a\ b\ b|b\ a\ a)\ \Sigma^*$. Alternatively, one can recognize this language with a DFA (as depicted in Figure 2.2) or a NFA (as depicted in Figure 2.1).

Figure 2.2: *DFA for* $\Sigma^*\ (a\ b\ b|b\ a\ a)\ \Sigma^*$.

We construct this automaton by storing the last two letters of any word in the states of the automaton and, upon encountering one of the desired patterns, we move into an accepting sink state.



Regular transition systems

This thesis deals with regular transition systems as a model of parameterized systems. The surveys [Abd12] and [Abd+04] on regular model checking (RMC) both credit [WB98] and [Kes+01] for the introduction of RMC (or at least for the observation that regular languages are a powerful tool to reason about parameterized systems). Since then the notation of the model has been streamlined significantly. We follow standard notations. In particular, the following definitions are akin to the ones in the aforementioned surveys [Abd12; Abd+04].

The underlying concept for RMC is simple. Consider some systems $\mathcal{S}$ that is parameterized by some value n. For example, $\mathcal{S}$ may describe a protocol for mutual exclusion in which n is the number of agents that participate in the execution of this protocol. If we additionally assume that every agent can be only in a finite number of states, say Σ, one can describe the state of some execution of $\mathcal{S}$ for n agents as some word of Σ^n. The first letter of this word is then the state of the first agent, the second letter the state of the second agent, and so on. Considering the current state of some *instance* of $\mathcal{S}$ as a word over the finite alphabet Σ allows us to define sets of configurations as languages in Σ^*. Assume that $c \in \Sigma$ is the critical state for which $\mathcal{S}$ guarantees mutual exclusion. Then, all elements of the set $\Sigma^*\, c\, \Sigma^*\, c\, \Sigma^*$ violate that property and must not be reached in $\mathcal{S}$.

The second crucial observation is the following: the system $\mathcal{S}$ specifies some operational semantics; i.e., how instances of the system might change their state. In RMC we assume that the behavior of $\mathcal{S}$ can be captured by a regular language over the alphabet $\Sigma \times \Sigma$ such that the configuration $v_1 \ \ldots \ v_n$ can change into configuration $u_1 \ \ldots \ u_n$ if (and only if) the word $\langle v_1, u_1\rangle \ \ldots \ \langle v_n, u_n\rangle$ is part of that language. To formalize this, we introduce the concept of *transducers*[1].

Definition 2.3: *Transducer.*

A Σ-Γ-*transducer* is an NFA $\langle Q, Q_0, \Sigma \times \Gamma, \Delta, F\rangle$. We identify with any Σ-Γ-*transducer* $\mathcal{T}$ a relation

$$\left\{ \langle v_1 \ \ldots \ v_n, u_1 \ \ldots \ u_n\rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Gamma^n \mid \langle v_1, u_1\rangle \ \ldots \ \langle v_n, u_n\rangle \in \mathcal{L}(\mathcal{T}) \right\}$$

which we denote with $[\![\mathcal{T}]\!]$. Note that this relation can only relate words of the

---

[1] More particularly, the transducers we introduce are all *length-preserving* which means that one can only relate words of the same length via a transducer.

same length. Let us introduce some related notation.

$$\text{For } v \in \Sigma^*\colon\ \mathrm{target}_{\mathcal{T}}(v) = \{u \in \Gamma^* \mid \langle v, u\rangle \in [\![\mathcal{T}]\!]\}$$
$$\text{For } u \in \Gamma^*\colon\ \mathrm{source}_{\mathcal{T}}(u) = \{v \in \Sigma^* \mid \langle v, u\rangle \in [\![\mathcal{T}]\!]\}$$

Additionally, we want to extend this notation to sets in the expected way: $\mathrm{target}_{\mathcal{T}}(V) = \bigcup_{v\in V} \mathrm{target}_{\mathcal{T}}(v)$ and $\mathrm{source}_{\mathcal{T}}(U) = \bigcup_{u\in U} \mathrm{source}_{\mathcal{T}}(u)$.

Throughout the thesis, we use the folklore knowledge that one can use standard product constructions to chain the relations of transducers together. More specifically, one can obtain from a Σ-Γ-transducer $\mathcal{F}$ and a Γ-Υ-transducer $\mathcal{S}$ a Σ-Υ-transducer $\mathcal{C}$ such that

$$[\![\mathcal{C}]\!] = [\![\mathcal{F}]\!] \circ [\![\mathcal{S}]\!] = \left\{ \langle u, w\rangle \in \bigcup_{n\geq 0} \Sigma^n \times \Upsilon^n \mid \exists v \ .\ \langle u, v\rangle \in [\![\mathcal{F}]\!] \text{ and } \langle v, w\rangle \in [\![\mathcal{S}]\!] \right\}.$$

Additionally, we use that one can obtain from a given NFA $\mathcal{A}$ over the alphabet Σ in a straightforward way a Σ-Σ-transducer $\mathcal{B}$ such that $[\![\mathcal{B}]\!] = \{\langle u, u\rangle : u \in \mathcal{L}(\mathcal{A})\}$. We refer to this relation in the future as $Id(\mathcal{A})$. Finally, one can also project, for any transducer, to either the origin or target of all letters and obtain a NFA for the language of all origins or the language of all targets.

Lemma 2.1. *Let*

- $\mathcal{F}$ *be a* Σ-Γ-*transducer with* $n_{\mathcal{F}}$ *states,*
- $\mathcal{S}$ *be a* Γ-Υ-*transducer with* $n_{\mathcal{S}}$ *states, and*
- $\mathcal{A}$ *be a NFA over the alphabet* Σ *with* $n_{\mathcal{A}}$ *states.*

One can effectively construct

- *a* Σ-Υ-*transducer* $\mathcal{C}$ *with* $n_{\mathcal{F}} \cdot n_{\mathcal{S}}$ *states such that* $[\![\mathcal{C}]\!] = [\![\mathcal{F}]\!] \circ [\![\mathcal{S}]\!]$,
- *a* Σ-Σ-*transducer* $\mathcal{B}$ *with* $n_{\mathcal{A}}$ *states such that* $[\![\mathcal{B}]\!] = \{\langle u, u\rangle : u \in \mathcal{L}(\mathcal{A})\} = Id(\mathcal{A})$, *and*
- *NFAs* $\mathcal{D}$ *and* $\mathcal{E}$ *over the alphabets* Σ, Γ *respectively with* $n_{\mathcal{F}}$ *states each such that* $\mathcal{L}(\mathcal{D}) = \mathrm{source}_{\mathcal{F}}(\Gamma^*)$ *and* $\mathcal{L}(\mathcal{E}) = \mathrm{target}_{\mathcal{F}}(\Sigma^*)$.

Based on this definition, we introduce regular transition systems; the central model of this thesis.

Definition 2.4: *Regular transition system (RTS).*

An RTS is a triple $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ where Σ is a finite alphabet while $\mathcal{I}$ is an NFA with alphabet Σ and $\mathcal{T}$ is a Σ-Σ-transducer.
We denote with $\rightsquigarrow_{\mathcal{T}}$ the relation $[\![\mathcal{T}]\!]$ and call a pair $\langle v, u \rangle \in \rightsquigarrow_{\mathcal{T}}$ (which we also write $v \rightsquigarrow_{\mathcal{T}} u$) a transition of $\mathcal{R}$. Moreover, let $\rightsquigarrow_{\mathcal{T}}^{*}$ denote the reflexive transitive closure of $\rightsquigarrow_{\mathcal{T}}$.
We consider $w \in \Sigma^*$ *reachable* in $\mathcal{R}$ if there exists $u \in \mathcal{L}(\mathcal{I})$ with $u \rightsquigarrow_{\mathcal{T}}^{*} w$. Let $\mathit{reach}(\mathcal{R}) \subseteq \Sigma^*$ denote all reachable configurations.

In the following, we represent pairs $\langle \sigma_1, \sigma_2 \rangle \in \Sigma \times \Sigma$ in transducers as $\begin{bmatrix}\sigma_1\\ \sigma_2\end{bmatrix}$ for readability. Also, we denote with struck-out versions of relation symbols; e. g. $\not\rightsquigarrow_{\mathcal{T}}$, the complement of the relation (which still only relates words of the same length). In this case, $\not\rightsquigarrow_{\mathcal{T}} = \left\{ \langle u, v \rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Sigma^n \mid \langle u, v \rangle \notin [\![\mathcal{T}]\!] \right\}$.

We want to conclude this chapter with the presentation of two examples. The first is a token passing algorithm[2] [Abd+04; Abd12]. The second example is the dining philosophers – a well-known parametrized system.

Example 2.2: *Token passing as RTS.*

This system consists of a linear array of agents. Initially, the first agent holds a token. In every step, the agent that currently holds the token can pass it down the line one step. To represent the system as an RTS we choose to represent the agent that holds the token as the letter t and the agents that do not hold the token as the letter n. Consequently, we choose the language of initial configurations to be $t\,n^*$. We capture the transitions of the system via the language $\begin{bmatrix}n\\ n\end{bmatrix}^* \begin{bmatrix}t\\ n\end{bmatrix} \begin{bmatrix}n\\ t\end{bmatrix} \begin{bmatrix}n\\ n\end{bmatrix}^*$. The corresponding NFAs $\mathcal{I}$ and $\mathcal{T}$ are depicted in Figure 2.3 and Figure 2.4, respectively.
Let us consider $t\,n\,n \in \mathcal{L}(\mathcal{I})$. We have $\mathrm{target}_{\mathcal{T}}(t\,n\,n) = \{n\,t\,n\}$ because $t\,n\,n \rightsquigarrow_{\mathcal{T}} n\,t\,n$ is the only transition[a] that originates in $t\,n\,n$. Similarly, there is exactly

---

[2] This example is the, so to speak, canonical example for RMC.

one transition that originates in $n\ t\ n$ which is $n\ t\ n \rightsquigarrow_{\mathcal{T}} n\ n\ t$. Since the token has reached the end of the agents, there are no more transitions to apply to the configuration $n\ n\ t$. Moreover, $t\ n\ n$ is the only configuration in $\mathcal{L}(\mathcal{I})$ of length 3. Hence, $\mathit{reach}(\mathcal{R}) \cap \Sigma^3$ coincides with $\{t\ n\ n, n\ t\ n, n\ n\ t\}$.
On close inspection of the language of $\mathcal{T}$ one observes that, from every initial configuration, there is a deterministic sequence of configurations that are reachable by handing down the token to the end of the line. Therefore, the language of all reachable configurations is $\mathit{reach}(\mathcal{R}) = n^*\ t\ n^*$.

---

[a] $t\ n\ n \rightsquigarrow_{\mathcal{T}} n\ t\ n$ is a transition because $\begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix}$ is accepted by $\mathcal{T}$.

Figure 2.3: *$\mathcal{I}$ for Example 2.2.*

Illustration of the automaton that recognizes the language of initial words for Example 2.2.

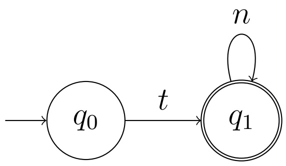
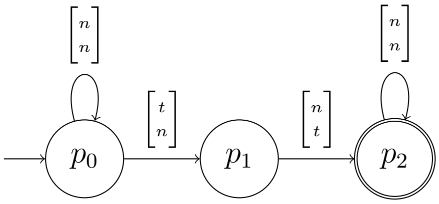


Figure 2.4: *$\mathcal{T}$ for Example 2.2.*

Illustration of the automaton that recognizes the language of transitions for Example 2.2.



Remark 2.1. *Note that the definition of regular transition systems only allows configurations to reach configurations of the same length. Hence, once one has selected any initial configuration, there are only finitely many configurations reachable from that*

initial configuration[3].

Example 2.3: *Dining philosophers as RTS.*

Let us introduce the example of the dining philosophers. In this parameterized system we consider a group of philosophers of some arbitrary but fixed size n. These philosophers sit around a round table and there is a fork placed between every two adjacent philosophers (and many cakes in the middle of the table). Every philosopher alternates between a state *thinking* and *eating*. In the state *eating* the philosopher picked up both forks adjacent to them and uses these to eat cakes from the middle of the table. The state *thinking*, on the other hand, models that the philosopher does not hold any fork but is only concerned with the thoughts in their head. The forks alternate between states *free* and *busy* to indicate whether they are lying on the table or are taken by a philosopher, respectively.

We model two different ways for the philosophers to move from state *thinking* to *eating* as RTSs. In the first version, all philosophers grab the forks adjacent to them in an atomic step. For this, we represent the state *free* and *busy* of the forks as f and b respectively. The states *thinking* and *eating* of the philosophers are represented as t and e, respectively.

In the second version, all philosophers grab the forks adjacent to them one by one. Here, we introduce a third state for all the philosophers (h) to indicate that this philosopher already grabbed one of the forks adjacent to them but not the second one. Moreover, we fix, for all philosophers but one, that they grab first the fork on their right and, afterward, the fork on their left. For the one special philosopher, this order is reversed. We refer to this philosopher as a *left-hander*. That is, this philosopher first grabs the fork on their left and, then, the fork on their right.

The atomic version

First, we consider the *atomic* variant where the philosophers grab both adjacent forks simultaneously. Initially, all philosophers are thinking and all forks are free. Hence, we choose $\mathcal{I}$ such that $\mathcal{L}(\mathcal{I})$ coincides with the regular expression $(t\ f)^*$ (as illustrated in Figure 2.5). Grabbing both forks simultaneously can be modeled via

---

[3] For this reason, one sometimes refers to such systems as *weakly finite* [EGK12].

the union of three regular languages. We describe these languages as regular expressions. To ease the presentation, we introduce two placeholders P and F which describe that we skip over a philosopher or fork respectively without changing it. Formally, we set

$$P = \left(\begin{bmatrix} t \\ t \end{bmatrix} \middle| \begin{bmatrix} e \\ e \end{bmatrix} \right) \text{ and } F = \left(\begin{bmatrix} f \\ f \end{bmatrix} \middle| \begin{bmatrix} b \\ b \end{bmatrix} \right).$$

The first language encodes that some philosopher but the first either grabs or releases their adjacent forks:

$$P \ (F\ P)^* \left(\begin{bmatrix} f \\ b \end{bmatrix} \begin{bmatrix} t \\ e \end{bmatrix} \begin{bmatrix} f \\ b \end{bmatrix} \middle| \begin{bmatrix} b \\ f \end{bmatrix} \begin{bmatrix} e \\ t \end{bmatrix} \begin{bmatrix} b \\ f \end{bmatrix} \right) (P\ F)^*.$$

The other two model the behavior of the first philosopher:

$$\begin{bmatrix} t \\ e \end{bmatrix} \begin{bmatrix} f \\ b \end{bmatrix} (P\ F)^* \ F \begin{bmatrix} f \\ b \end{bmatrix} \text{ and } \begin{bmatrix} e \\ t \end{bmatrix} \begin{bmatrix} b \\ f \end{bmatrix} (P\ F)^* \ F \begin{bmatrix} b \\ f \end{bmatrix}.$$

The language that corresponds to these regular expressions is recognized by the NFA depicted in Figure 2.6.

The version with one left-hander

Alternatively, we consider a non-atomic variant of the dining philosophers. Here the philosophers take the forks one by one in two individual steps. Since we introduce a third state (h) for the philosophers to indicate that they already grabbed one of the forks, we modify P from before slightly:

$$P = \left(\begin{bmatrix} t \\ t \end{bmatrix} \middle| \begin{bmatrix} h \\ h \end{bmatrix} \middle| \begin{bmatrix} e \\ e \end{bmatrix} \right).$$

Recall that we introduced one left-hander to the groups of philosophers. We do so because otherwise the philosophers are in danger: Assume, for the moment, that all philosophers grab the forks adjacent to them in two individual steps but all do so in the same order (first, the one to their right and second, the one to their left). Now imagine all philosophers doing the first step once. Thus, every philosopher holds exactly one fork and awaits the second fork to become free again. Unfortunately, this configuration cannot change anymore. Consequently, the system deadlocks and all philosophers starve. That is undesirable[a].

As already indicated, we fix this problem by introducing one philosopher who takes their forks in the opposite order than everyone else – the left-hander. By arbitrary choice, the first philosopher is the left-handed one. For the formalization, we keep $\mathcal{I}$ as before and we introduce $\mathcal{T}'$. We describe the language of $\mathcal{T}'$ in terms of regular expressions:

- $P\ (F\ P)^* \begin{bmatrix} f \\ b \end{bmatrix} \begin{bmatrix} t \\ h \end{bmatrix} (F\ P)^*\ F$ and $(P\ F)^+ \begin{bmatrix} h \\ e \end{bmatrix} \begin{bmatrix} f \\ b \end{bmatrix} (P\ F)^*$ model that a right-handed philosopher grabs their first or second fork.
- $\begin{bmatrix} t \\ h \end{bmatrix} \begin{bmatrix} f \\ b \end{bmatrix} (P\ F)^*$ and $\begin{bmatrix} h \\ e \end{bmatrix} F\ (P\ F)^*\ P \begin{bmatrix} f \\ b \end{bmatrix}$ model that the left-handed philosopher takes their first or second fork.
- $P\ (F\ P)^* \begin{bmatrix} b \\ f \end{bmatrix} \begin{bmatrix} e \\ t \end{bmatrix} \begin{bmatrix} b \\ f \end{bmatrix} P\ (F\ P)^*$ and $\begin{bmatrix} e \\ t \end{bmatrix} \begin{bmatrix} b \\ f \end{bmatrix} (P\ F)^*\ P \begin{bmatrix} b \\ f \end{bmatrix}$ model that any philosopher returns their forks.

---

[a] So I have been told.

Figure 2.5: *$\mathcal{I}$ of the dining philosophers.*

Illustration of the NFA recognizing the initial states of the dining philosophers of Exampe 2.3.

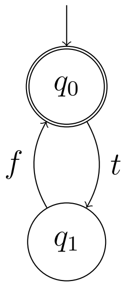
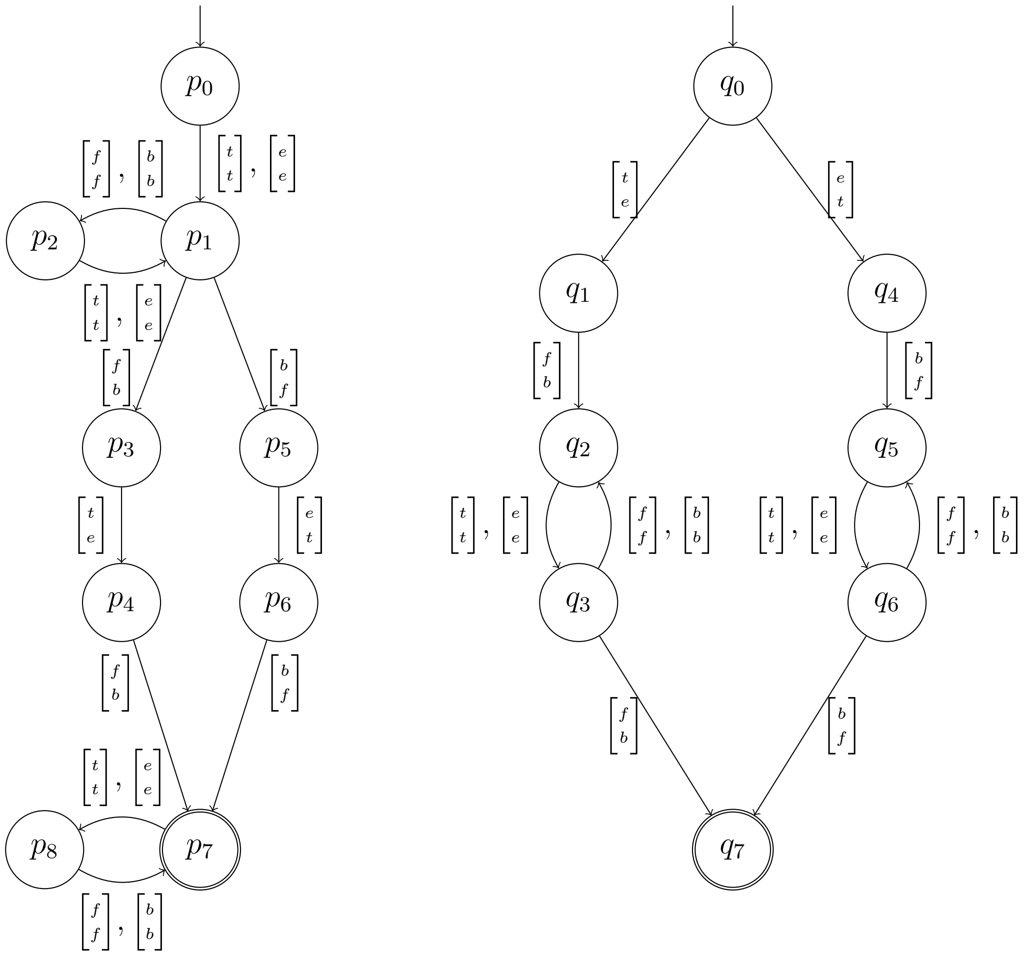


Figure 2.6: $\mathcal{T}$ *of the atomic dining philosophers.*

Illustration of the NFA recognizing the transitions of the atomic philosophers of Exampe 2.3.



2.2 Inductive statements for regular transition system

In this chapter, we explore a framework to reason over the reachable set of any RTS with logical statements. More precisely, we present an approach that is modular with respect to the form of the logical statements. At its core, this approach separates a logical statement into an *encoding* and the *interpretation* of an encoded statement. This allows us to consider a very broad class of ways to reason about RTSs. Central to this chapter are *inductive statements*; that is, statements that are necessarily true after the execution of one transition if they were true before this transition. More specifically, a statement I is *inductive* if, for any transition $v \rightsquigarrow u$ where v satisfies I, u also satisfies the statement. The main result of this chapter is that, for any given *encoding* and *interpretation*, the set of all inductive statements form a regular language and are, therefore, algorithmically accessible.

The general safety problem

In general, we ask whether a given RTS can reach any word that we consider undesirable. In particular, we define a regular set of undesirable words and try to prove (automatically) that no word of this set is reachable. Essentially, the question we are trying to solve algorithmically is the following:

Problem 2.1 (The reachability problem)**.**

Given: *RTS* $\mathcal{R}$ *and NFA* B

Compute: $\mathit{reach}(\mathcal{R}) \cap \mathcal{L}(\mathcal{B}) = \emptyset$*?*

Unfortunately, this problem is, in general, undecidable [Blo+16]. Therefore, we consider a semi-decision procedure for it that answers the following question:

> Do an initial configuration v and an undesired configuration u exist, such that u satisfies all inductive statements that v satisfies?

It is clear that, if u is reachable from v, then u satisfies the inductive statements that v satisfies since they are inductive. Depending on the inductive statements, it is also possible that we can guarantee that no undesired configuration can be reached from any initial configuration. In other words, this means that, for every pair of initial and undesired configurations, there exists (at least) one inductive statement that is satisfied by the initial but not the undesired configuration. In this case, one can naturally deduce that no undesired configuration can be reached.

We proceed by developing the notion of logical statements and how to compute inductive ones. Let us introduce an example first and develop the formalism afterward: We consider a system where each agent is either in state p, q or t. Consider the statement "in all configurations of length 4 at least one of the first three agents is in state q". The configurations $q\ p\ p\ p$, $p\ q\ p\ p$, and $p\ p\ q\ p$ satisfy this statement while $p\ p\ p\ q$ does not. A similar statement could be "in all configurations of length 3 the first agent is in state

p *or* the first agent is in state q". Here $p\ t\ p$ or $q\ t\ q$ satisfy the statement but $t\ q\ p$ does not. Both statements share a common logical structure; that is, they follow the pattern "in all configurations of a certain length m either agent i_1 is in state σ_1 *or* agent i_2 is in state σ_2 *or* … *or* agent i_k is in state σ_k." If we stipulate that the statement "agent i_j is in state σ_j" is captured by the *atomic proposition* $\sigma_j(i_j)$ one can express all these statements in the form $\mathit{size} = m \to \bigvee_{1 \leq j \leq k} \sigma_j(i_j)$. Then, the statements above translate to

$$\mathit{size} = 4 \to (q(1) \vee q(2) \vee q(3)) \text{ and } \mathit{size} = 3 \to (q(1) \vee p(1))\,.$$

We specify for every statement the size for which it applies. Recall that, as soon as one picks the initial word, one can only reach configurations of the same size. Thus, we believe it makes sense to consider these *instances*; that is, the finite reachability graph of words of the same length, individually.

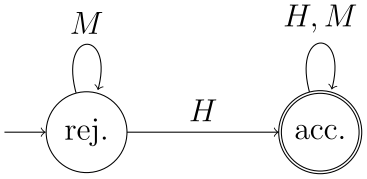
Consider a different view on statements: for example, the statement $\mathit{size} = 4 \to (q(1) \vee q(2) \vee q(3))$ contains two separate pieces of information. On the one hand, the size of the instance for which it applies, and, on the other hand, which states the first agent (in this case the state q), the second agent (q), the third agent (q), or the fourth agent (which is in this case irrelevant) can be in to satisfy the statement. In general, the necessary information of any statement can be encoded as a function $f\colon \{1, \dots, m\} \to 2^{\Sigma}$. For any such function, the domain encodes for which instance the statement is applicable, while the set of letters $f(i)$ corresponds to the states the i-th agent can be in to satisfy the statement. Thus, our examples would be

- the function $\{1 \mapsto \{q\}, 2 \mapsto \{q\}, 3 \mapsto \{q\}, 4 \mapsto \emptyset\}$, and
- the function $\{1 \mapsto \{p, q\}, 2 \mapsto \emptyset, 3 \mapsto \emptyset\}$, respectively.

Note that a function $f\colon \{1, \dots, m\} \to 2^{\Sigma}$ can be equivalently described as a word of length m over the alphabet 2^{Σ}. Therefore, we can represent our examples by the words $\{q\}\ \{q\}\ \{q\}\ \emptyset$ and $\{p, q\}\ \emptyset\ \emptyset$. These words are non-descriptive on their own. Their interpretation is crucial to understanding the statement they encode. Thus, one might ask the question: "Does $q\ p\ p\ p$ satisfy the statement encoded as $\{q\}\ \{q\}\ \{q\}\ \emptyset$?" This can be easily decided because one only has to check whether any of the following statements is true: $q \in \{q\}$, $p \in \{q\}$, $p \in \{q\}$, or $p \in \emptyset$. In fact, one can decide this with the help of a Σ-2^{Σ}-transducer which we depict in Figure 2.7.

Figure 2.7: *Disjunctive statement interpretation automaton.*

Illustration of an automaton that validates disjunctive statements. More precisely, we depict a Σ-2^Σ-transducer $\mathcal{V}$ such that $\langle v, I\rangle \in [\![\mathcal{V}]\!]$ if (and only if) the configuration v satisfies the encoded statement of a disjunction of atomic propositions I. For the transitions, we write H which denotes all pairs in $\langle v, I\rangle \in \Sigma \times 2^\Sigma$ such that $v \in I$ and M for all pairs in $\langle v, I\rangle \in \Sigma \times 2^\Sigma$ such that $v \notin I$.



Consider now a slightly different form of statements; i.e., statements of the form

$$\mathit{size} = m \to (\sigma_1(i_1)\ \mathit{xor}\ \ldots\ \mathit{xor}\ \sigma_k(i_k))\,.$$

That means the statement is satisfied by configurations of length m where an odd number of atomic propositions is satisfied by the configuration. Using the same considerations as before, one can express a statement of this form which ensures an odd amount of letters p in all configurations of length 6 as $\{p\}\ \{p\}\ \{p\}\ \{p\}\ \{p\}\ \{p\}$. Although the encoding is similar to before, the interpretation differs. However, any procedure that reads some configuration and this encoding in parallel only requires finite memory to decide whether the configuration satisfies the statement. In fact, the procedure only needs to update one bit to keep track of the fact whether an even or an odd amount of atomic propositions are satisfied. We present a Σ-2^Σ-transducer that realizes this in Figure 2.8.

Finally, we want to consider a third type of statement. This time we allow ourselves a conjunction of two disjunctions. Hence, the statements encode formulas of the form

$$\mathit{size} = m \to \bigvee_{1\le j\le k} \sigma_j(i_j) \wedge \bigvee_{1\le j\le \ell} \rho_j(n_j).$$

Essentially, this can be understood as reading two clauses simultaneously. As an example, consider the statement

$$size = 4 \to \left(\begin{array}{c} p(1) \vee q(1) \\ \wedge \\ t(1) \vee t(2) \vee t(3) \vee t(4) \end{array} \right). \tag{2.1}$$

Thus, we consider an encoding as a word over the alphabet $2^{\Sigma} \times 2^{\Sigma}$. Projecting every letter to its first component yields the same encoding for the first clause as before while projection to the second element of each letter gives us an encoding for the second clause. The example statement in (2.1) would be encoded as

$$\langle \{p, q\}, \{t\} \rangle \ \langle \emptyset, \{t\} \rangle \ \langle \emptyset, \{t\} \rangle \ \langle \emptyset, \{t\} \rangle .$$

Consequently, it is expected that we can construct a similar transducer as before by considering both clauses individually. We present this Σ-$2^{\Sigma} \times 2^{\Sigma}$-transducer in Figure 2.9.

Figure 2.8: *Parity statement interpretation automaton.*

Illustration of a Σ-2^{Σ}-transducer that validates *xor* statements. As for Figure 2.7 we write H, and M to denote all pairs in $\langle v, I \rangle \in \Sigma \times 2^{\Sigma}$ such that $v \in I$ and $v \notin I$, respectively.

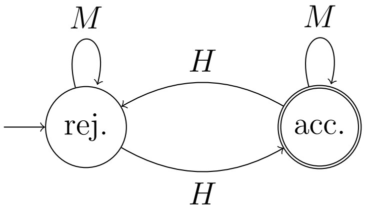


Figure 2.9: 2-*clause statement interpretation automaton.*

Illustration of a Σ-$2^\Sigma \times 2^\Sigma$-transducer that validates statements with two clauses. For the transitions, we introduce four different short forms which all represent pairs $\langle u, \langle A, B\rangle\rangle \in \Sigma \times \left(2^\Sigma \times 2^\Sigma\right)$:

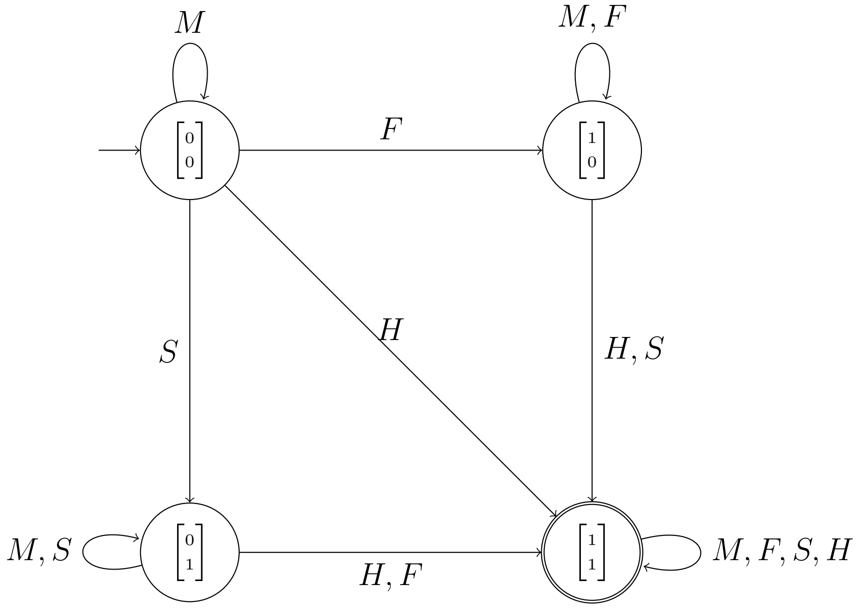
M**:** represents the case $u \notin A$ and $u \notin B$,

F**:** represents the case $u \in A$ and $u \notin B$,

S**:** represents the case $u \notin A$ and $u \in B$, and

H**:** represents the case $u \in A$ and $u \in B$.

The states are vectors of two bits. The first bit indicates whether the first clause is already satisfied. Similarly, the second bit tracks the satisfaction of the second clause.



2.3 A generic approach to statements

As we have mentioned before, we are particularly interested in inductive statements. The reason for this is that one can use inductive statements to over-approximate the reachable configurations from any given initial configuration. As we have seen before, one could consider various types of statements (each encoded and evaluated in some particular way). More formally, we relied on transducers to formalize encoding and evaluating statements. In this section, we show that, for any such transducer, the set of all inductive statements is a regular one. Moreover, we can use the set of all inductive statements to obtain an over-approximation of all reachable configurations.

Definition 2.5: *Interpretation.*

For any RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$, we call a pair $\langle \Gamma, \mathcal{V} \rangle$ an Γ-*interpretation* where Γ is a finite alphabet and $\mathcal{V}$ is a deterministic Σ-Γ-transducer. In the following, we sometimes write $u \models_{\mathcal{V}} I$ to indicate $\langle u, I \rangle \in [\![\mathcal{V}]\!]$.

Example 2.4: *A satisfied statement.*

Given our intuition, one would expect that $q\ p\ p\ q$ satisfies the statement $\mathit{size} = 4 \to (q(1) \vee q(2) \vee q(3))$. Therefore, the interpretation automaton depicted in Figure 2.7 should accept $\langle q, \{q\} \rangle\ \langle p, \{q\} \rangle\ \langle p, \{q\} \rangle\ \langle q, \emptyset \rangle$ because $\mathit{size} = 4 \to (q(1) \vee q(2) \vee q(3))$ is encoded as $\{q\}\ \{q\}\ \{q\}\ \emptyset$. And, indeed, we can verify that $\langle q\ p\ p\ q, \{q\}\ \{q\}\ \{q\}\ \emptyset \rangle$ is part of the relation of this interpretation.

If Γ is clear from the context we might refer to a Γ-interpretation simply as an interpretation. We choose to make the interpretation deterministic to ease some of the proofs later. Naturally, this requirement can be lifted for all the following results. However, some of the bounds on the number of states of some of the automata have to be adapted accordingly.

Any Γ-interpretation describes a class of statements for any RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$. Now, we identify statements that are inductive with respect to the transitions of $\mathcal{R}$; that is, $[\![\mathcal{T}]\!]$.

Definition 2.6: *Inductive statements.*

For any given Γ-interpretation $\mathcal{V}$ for $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$, we define

$$\begin{aligned}\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R}) &= \{ I \in \Gamma^* \mid \forall u \rightsquigarrow_{\mathcal{T}} v \text{ . if } \langle u, I \rangle \in [\![\mathcal{V}]\!] \text{ then } \langle v, I \rangle \in [\![\mathcal{V}]\!] \} \\ &= \{ I \in \Gamma^* \mid \forall u \rightsquigarrow_{\mathcal{T}} v \text{ . if } u \models_{\mathcal{V}} I \text{ then } v \models_{\mathcal{V}} I \} \,.\end{aligned}$$

Example 2.5: *An inductive statement of Example 2.2.*

Recall the token passing model $\mathcal{R}$ of Example 2.2. Additionally, we focus on the interpretation that we illustrated in Figure 2.7. That means, that we specifically consider an 2^{Σ}-interpretation $\langle 2^{\Sigma}, \mathcal{V} \rangle$. We argue that $\emptyset^* \ \{n\} \ \emptyset^* \ \{n\} \ \emptyset^* \subseteq \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$. To this end, observe that each word of this language corresponds to the statement "for two given indices i, j either the i-th letter is n or the j-th letter is n" where the indices correspond to the two positions where the letters are $\{n\}$; e.g., $\{n\} \ \emptyset \ \emptyset \ \{n\} \ \emptyset$ applies to words of the length 5 and ensures that either the first or fourth letter is n. Note that all transitions of this example originate in a configuration where exactly one letter is t and end up in a configuration where exactly one letter is t. Consequently, the origin and the target of every transition satisfy all of these statements. Therefore, we can conclude that $\emptyset^* \ \{n\} \ \emptyset^* \ \{n\} \ \emptyset^* \subseteq \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$.

Note that, for any RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and interpretation $\mathcal{V}$, any inductive statement $I \in \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ that is satisfied in one configuration w ($w \models_{\mathcal{V}} I$) is also satisfied in all configurations that can be reached from w ($u \models_{\mathcal{V}} I$ for all $w \rightsquigarrow_{\mathcal{T}}^* u$). This is because I stays satisfied along every possible transition of the RTS. In this way, we can try to approximate the relation $\rightsquigarrow^*$ by considering some configuration u reachable from configuration w if u satisfies the same inductive statements as w. Let us illustrate this with another example.

Example 2.6: *Approximating reachability via inductive statements.*

As we have seen in the previous example $\emptyset^* \ \{n\} \ \emptyset^* \ \{n\} \ \emptyset^* \subseteq \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ for the interpretation that is depicted in Figure 2.7 and the RTS from Example 2.2.

With respect to these statements one can, for example, conclude that $t\ n\ n\ n$ can be reached from $n\ t\ t\ t$. The reason for this is that the words of $\emptyset^*\ \ \{n\}\ \ \emptyset^*\ \ \{n\}\ \ \emptyset^*$ that are satisfied by $n\ t\ t\ t$ are

$$\begin{array}{cccc} \{n\} & \{n\} & \emptyset & \emptyset, \\ \{n\} & \emptyset & \{n\} & \emptyset, \\ \{n\} & \emptyset & \emptyset & \{n\}. \end{array}$$

All these statements are also satisfied by $t\ n\ n\ n$.
On the other hand, one can also see that $\emptyset^*\ \ \{t\}^* \subseteq \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$. This is true because every transition only moves the letter t one step to the right. Using these statements one can immediately see that $t\ n\ n\ n$ cannot be reached from $n\ t\ t\ t$ because the latter satisfies

$$\begin{array}{cccc} \emptyset & \{t\} & \{t\} & \{t\}, \\ \emptyset & \emptyset & \{t\} & \{t\}, \\ \emptyset & \emptyset & \emptyset & \{t\} \end{array}$$

while the former does not.

We proceed now by proving that $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ is a regular language for any RTS $\mathcal{R}$ and interpretation $\mathcal{V}$. We do so by considering the complement of the language since it is more intuitive.

Lemma 2.2. *Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ be an RTS. One can construct effectively an NFA with $\mathcal{O}(n_{\mathcal{T}} \cdot n_{\mathcal{V}}^2)$ states for $\overline{\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})}$ where $n_{\mathcal{T}}$ and $n_{\mathcal{V}}$ are the number of states of $\mathcal{T}$ and $\mathcal{V}$, respectively.*

Proof. By definition

$$\overline{\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})} = \{I \in \Gamma^* \mid \exists u \rightsquigarrow_{\mathcal{T}} w \ .\ u \models_{\mathcal{V}} I \text{ and } w \not\models_{\mathcal{V}} I\}\,.$$

In the remainder of the proof, we construct an automaton that recognizes this language. This automaton guesses for its input $I_1\ \ldots\ I_n$ an accepting run of $\mathcal{T}$ on some transition $u_1\ \ldots\ u_n \rightsquigarrow_{\mathcal{T}} v_1\ \ldots\ v_n$ and tracks the state q_u of $\mathcal{V}$ for the word $\langle u_1, I_1\rangle\ \ldots\ \langle u_n, I_n\rangle$ and q_v for $\langle v_1, I_1\rangle\ \ldots\ \langle v_n, I_n\rangle$, respectively. Since the transition $u_1\ \ldots\ u_n \rightsquigarrow_{\mathcal{T}} v_1\ \ldots\ v_n$ needs to witness that $I_1\ \ldots\ I_n$ is *not* inductive the automaton accepts if q_u is an accepting state while q_v is not. To this end, let $\mathcal{T} =$

$\langle P, \Sigma \times \Sigma, \Delta, p_0, E \rangle$ and $\mathcal{V} = \langle Q, \Sigma \times \Gamma, \delta, q_0, F \rangle$. The automaton for $\overline{\text{Inductive}_{\mathcal{V}}(\mathcal{R})}$ is $\langle P \times Q \times Q, \Gamma, \nabla, \langle p_0, q_0, q_0 \rangle, E \times F \times (Q \setminus F) \rangle$ where

$$\nabla = \left\{ \langle \langle p, q_1, q_2 \rangle, I, \langle p', q_1', q_2' \rangle \rangle \;\middle|\; \begin{array}{r} \exists \langle \sigma_1, \sigma_2 \rangle \in \Sigma \times \Sigma \ . \ \langle p, \langle \sigma_1, \sigma_2 \rangle, p' \rangle \in \Delta \\ \text{and } \delta(q_1, \langle \sigma_1, I \rangle) = q_1' \\ \text{and } \delta(q_2, \langle \sigma_2, I \rangle) = q_2' \end{array} \right\}.$$

Observe now that any accepting run in this automaton can be separated into three parts:

- The projection to the first element of the run yields an accepting run of $\mathcal{T}$ on some $\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \ \dots \ \begin{bmatrix} u_n \\ , v_n \end{bmatrix}$.
- The projection to the second element of the run yields the accepting run of $\mathcal{V}$ on $\langle u_1, I_1 \rangle \ \dots \ \langle u_n, I_n \rangle$.
- The projection to the third element of the run yields the rejecting run of $\mathcal{V}$ on $\langle v_1, I_1 \rangle \ \dots \ \langle v_n, I_n \rangle$.

□

As we said before (and have illustrated in Example 2.6), we want to use the inductive statements to obtain a relationship of *potential* reachability between two configurations. Before we state the definition, recall that $\text{target}_{\mathcal{V}}(u)$ is, for any interpretation $\mathcal{V}$, the set of all statements I such that $u \models_{\mathcal{V}} I$.

Definition 2.7: *Potential reachability.*

Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ be any RTS and $\langle \Gamma, \mathcal{V} \rangle$ any interpretation. We write $u \Rightarrow_{\mathcal{V}} v$ if and only if $v \models_{\mathcal{V}} I$ for all $I \in \text{target}_{\mathcal{V}}(u) \cap \text{Inductive}_{\mathcal{V}}(\mathcal{R})$.

From this definition and the definition of $\text{Inductive}_{\mathcal{V}}(\mathcal{R})$, the following observation is immediate:

Lemma 2.3. *Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ be an RTS and $\langle \Gamma, \mathcal{V} \rangle$ an interpretation. If $u \rightsquigarrow_{\mathcal{T}}^{*} v$, then $u \Rightarrow_{\mathcal{V}} v$.*

We dedicate the remainder of this section to proving that this relation of potential reachability can be captured by a Σ-Σ-transducer.

In fact, we prove the following result.

Theorem 2.1. *For any RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and interpretation $\langle \Gamma, \mathcal{V} \rangle$, there exists a Σ-Σ-transducer $\mathcal{C}$ such that $\Rightarrow_{\mathcal{V}}$ coincides with $[\![\mathcal{C}]\!]$.*

To do so, we first look at the complement of this relation. More specifically, we show that we can construct a Σ-Σ-transducer $\overline{\mathcal{C}}$ such that

$$[\![\overline{\mathcal{C}}]\!] = \left\{ \langle u, v \rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Sigma^n \mid u \not\Rightarrow_{\mathcal{V}} v \right\}.$$

The intuition behind our construction is as follows: $u \not\Rightarrow_{\mathcal{V}} v$ means that there is some statement $I \in \text{Inductive}_{\mathcal{V}}(\mathcal{R})$ such that $u \models_{\mathcal{V}} I$ and $v \not\models_{\mathcal{V}} I$. Hence, it suffices to non-deterministically guess this statement I and verify that u satisfies it while v does not. Therefore, we set out to prove:

Lemma 2.4. *For any RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and interpretation $\langle \Gamma, \mathcal{V} \rangle$ there exists a Σ-Σ-transducer $\overline{\mathcal{C}}$ such that*

$$[\![\overline{\mathcal{C}}]\!] = \left\{ \langle u, v \rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Sigma^n \mid u \not\Rightarrow_{\mathcal{V}} v \right\}.$$

We prove a slightly stronger result that we can reuse later. The idea is that Lemma 2.4 may use any statement I from $\text{Inductive}_{\mathcal{V}}(\mathcal{R})$ to witness that one cannot reach from some configuration u another configuration v. We provide a construction that is agnostic concerning the set of statements it may use to disprove reachability as long as this set is regular and provided as an NFA. Lemma 2.4 can be obtained as a corollary from that by choosing $\text{Inductive}_{\mathcal{V}}(\mathcal{R})$ as the set of statements available in the construction.

Lemma 2.5. *Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ be an RTS, $\langle \Gamma, \mathcal{V} \rangle$ an interpretation, and $\mathcal{S}$ an NFA over the alphabet Γ. Then there exists a Σ-Σ-transducer $\overline{\mathcal{C}}$ such that*

$$[\![\overline{\mathcal{C}}]\!] = \left\{ \langle u, v \rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Sigma^n \mid \exists I \in \mathcal{L}(\mathcal{S}) \ . \ u \models_{\mathcal{V}} I \text{ and } v \not\models_{\mathcal{V}} I \right\}.$$

Moreover, $\overline{\mathcal{C}}$ can be effectively computed and has $\mathcal{O}(n_{\mathcal{S}} \cdot n_{\mathcal{V}}^2)$ many states, where $n_{\mathcal{S}}$ and $n_{\mathcal{V}}$ are the numbers of states of $\mathcal{S}$ and $\mathcal{V}$, respectively.

Proof. Fix $\mathcal{S} = \langle P, \Gamma, \Delta, p_0, E \rangle$ and $\mathcal{V} = \langle Q, \Sigma \times \Gamma, \delta, q_0, F \rangle$ and construct

$$\overline{\mathcal{C}} = \langle Q \times P \times Q, \Sigma \times \Sigma, \nabla, \langle q_0, p_0, q_0 \rangle, F \times E \times (Q \setminus F) \rangle$$

with $\langle\langle q_1, p, q_2\rangle, \langle\sigma_1, \sigma_2\rangle, \langle q_1', p', q_2'\rangle\rangle \in \nabla$ if and only if there exists $I \in \Gamma$ such that $\langle p, I, p'\rangle \in \Delta$ and $\delta(q_i, \langle\sigma_i, I\rangle) = q_i'$ for $i \in \{1, 2\}$. Any run of $\overline{\mathcal{C}}$ on its input $\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \dots \begin{bmatrix} u_n \\ v_n \end{bmatrix}$ corresponds to a guess for $I_1 \dots I_n$ that can be separated in three parts:

- The projection to the first element of the states of the run yields the run of $\mathcal{V}$ on $\langle u_1, I_1\rangle \dots \langle u_n, I_n\rangle$.
- The projection to the second element of the states of the run yields a run of $\mathcal{S}$ on $I_1 \dots I_n$.
- The projection to the third element of the states of the run yields the run of $\mathcal{V}$ on $\langle v_1, I_1\rangle \dots \langle v_n, I_n\rangle$.

By the choice of the accepting states, the correctness of the construction follows. □

We see now that Theorem 2.1 follows from Lemma 2.4 because regular languages are closed under complement. In the same manner, one can also prove the following observation:

Lemma 2.6. *Let $\mathcal{R} = \langle\Sigma, \mathcal{I}, \mathcal{T}\rangle$ be an RTS, $\langle\Gamma, \mathcal{V}\rangle$ an interpretation, and $\mathcal{S}$ an NFA over the alphabet Γ. Then there exists a Σ-Σ-transducer $\mathcal{C}$ such that*

$$[\![\mathcal{C}]\!] = \left\{ \langle u, v\rangle \in \bigcup_{n\geq 0} \Sigma^n \times \Sigma^n \mid \forall I \in \mathcal{L}(\mathcal{S}) \;.\; \textit{if } u \models_{\mathcal{V}} I \textit{ then } v \models_{\mathcal{V}} I \right\}.$$

Moreover, $\overline{\mathcal{C}}$ can be effectively computed and has $\mathcal{O}(2^{n_{\mathcal{S}} \cdot n_{\mathcal{V}}^2})$ many states, where $n_{\mathcal{S}}$ and $n_{\mathcal{V}}$ are the numbers of states of $\mathcal{S}$ and $\mathcal{V}$, respectively.

Since this relation becomes relevant later, we introduce a notation for it here. Thus, in the following, we denote for any language $\mathcal{L} \subseteq \Gamma^*$ with $\xRightarrow{\mathcal{L}}_{\mathcal{V}}$ the relation

$$\left\{ \langle u, v\rangle \in \bigcup_{n\geq 0} \Sigma^n \times \Sigma^n \mid \forall I \in \mathcal{L} \;.\; \text{if } u \models_{\mathcal{V}} I \text{ then } v \models_{\mathcal{V}} I \right\}.$$

For example, $\Rightarrow_{\mathcal{V}}$ coincides with $\xRightarrow{\text{Inductive}_{\mathcal{V}}(\mathcal{R})}_{\mathcal{V}}$.

Recall the initial question of this chapter. There we asked how we can establish certain safety conditions for given RTSs by using inductive invariants as a basis for an over-approximation of the reachable states. With the help of $\Rightarrow_{\mathcal{V}}$ we can formalize this idea.

For this, consider a RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and any Γ-interpretation $\mathcal{V}$. We can construct a Σ-Σ-transducer $\mathcal{P}$ such that $[\![\mathcal{P}]\!] = Id(\mathcal{I}) \circ \Rightarrow_{\mathcal{V}}$[4] by using Lemma 2.1. Observe now that $\mathrm{target}_{\mathcal{P}}(\Sigma^*)$ is indeed a regular[5] over-approximation of $reach(\mathcal{R})$. Let us illustrate this in our running example.

Example 2.7: *Computing an over-approximation.*

Consider the formalization of the token passing algorithm $\mathcal{R}$ from Example 2.2 and the interpretation $\mathcal{V}$ depicted in Figure 2.7, again. Additionally, recall that we observed that $\{t\}^* \subseteq \emptyset^* \ \{t\}^* \subseteq \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ and $\emptyset^* \ \{n\} \ \emptyset^* \ \{n\} \ \emptyset^* \subseteq \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$. We argue now that the over-approximation that arises from $\mathrm{target}_{\mathcal{P}}(\Sigma^*)$ where $\mathcal{P}$ is the transducer for $Id(\mathcal{I}) \circ \Rightarrow_{\mathcal{V}}$ coincides with the set of actually reachable configurations $n^* \ t \ n^*$. The reason for this is that the language $\{t\}^*$ of inductive statements enforces that in every configuration at least one of the letters is t while the language $\emptyset^* \ \{n\} \ \emptyset^* \ \{n\} \ \emptyset^*$ ensures that there are no t. For the latter observation, consider a configuration in the instance of length m with two t, say at positions $i < j$. This configuration fails to satisfy the statement $\emptyset^{i-1} \ \{n\} \ \emptyset^{j-1-i} \{n\} \ \emptyset^{m-j}$ although the unique initial configuration $t \ n^{m-1}$ of this instance does satisfy this statement. Consequently, the over-approximation coincides, in this instance, with $reach(\mathcal{R})$.

We have shown how to obtain, for any interpretation $\mathcal{V}$, a regular over-approximation of $reach(\mathcal{R})$. This observation inspires a semi-decision procedure for the initially stated question of whether $reach(\mathcal{R}) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ for some given RTS $\mathcal{R}$ and an automaton $\mathcal{B}$ which recognizes undesired words. To state the computational problem, we observe that $\mathrm{target}_{\mathcal{P}}(\Sigma^*) \cap \mathcal{L}(\mathcal{B}) = \emptyset$ where $\mathcal{P}$ is the transducer for $Id(\mathcal{I}) \circ \Rightarrow_{\mathcal{V}}$ if and only if $Id(\mathcal{I}) \circ \Rightarrow_{\mathcal{V}} \circ Id(\mathcal{B}) = \emptyset$.

---

[4] Remember that $Id(\mathcal{I}) = \{\langle u, u \rangle : u \in \mathcal{L}(\mathcal{I})\}$.

[5] We use, again, Lemma 2.1.

Problem 2.2 (The approximated reachability problem).
Let $\mathcal{V}$ be any Γ-interpretation.

Given:	*RTS* $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ *and NFA* $\mathcal{B}$
Compute:	$Id(\mathcal{I}) \circ \Rightarrow_{\mathcal{V}} \circ Id(\mathcal{B}) = \emptyset$*?*

Remark 2.2. *Note here that we treat the interpretation $\mathcal{V}$ as a constant of the problem. This is not necessary. The interpretation could be part of the input as well. We argue, however, that it is implausible that one tailors an interpretation to a specific problem since this requires a lot of human effort. Rather we expect that one relies on some standard interpretations that come with any program for this problem.*

Also, note that all regular languages of this problem are defined via NFAs. It is important to note that the hardness results we present later do not rely on this fact. Specifically, they still hold if one restricts oneself to only using deterministic automata for $\mathcal{I}$, $\mathcal{T}$, and $\mathcal{B}$.

Relying on the previous analysis, we can immediately give an upper bound on the complexity of this problem:

Theorem 2.2. *Problem 2.2 is in* ExpSpace.

Proof. Fix $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$. Also, we denote with

- $n_{\mathcal{I}}$ the number of states of $\mathcal{I}$,
- $n_{\mathcal{T}}$ the number of states of $\mathcal{T}$, and
- $n_{\mathcal{B}}$ the number of states of $\mathcal{B}$.

In Lemma 2.2, an NFA for $\overline{\text{Inductive}_{\mathcal{V}}(\mathcal{R})}$ with $\mathcal{O}(n_{\mathcal{T}})$ states is constructed. Hence, one can obtain a DFA for $\text{Inductive}_{\mathcal{V}}(\mathcal{R})$ with $\mathcal{O}(2^{n_{\mathcal{T}}})$ states by complementing the automaton for $\overline{\text{Inductive}_{\mathcal{V}}(\mathcal{R})}$. Equipped with this automaton, one can construct $\overline{\mathcal{C}}$ as described in Lemma 2.4 with roughly the same amount of states; that is, again $\mathcal{O}(2^{n_{\mathcal{T}}})$. This yields a DFA $\mathcal{C}$ which captures $\Rightarrow_{\mathcal{V}}$ with $\mathcal{O}(2^{2^{n_{\mathcal{T}}}})$ states. More precisely, the states of this automaton $\mathcal{C}$ are subsets of states of $\overline{\mathcal{C}}$. Therefore, one of the states can be stored in space $\mathcal{O}(2^{n_{\mathcal{T}}})$. Using Lemma 2.1, one can see that checking whether $Id(\mathcal{I}) \circ \Rightarrow_{\mathcal{V}} \circ Id(\mathcal{B}) = \emptyset$ can be realized by checking for emptiness in an automaton where every state can be stored in space $\mathcal{O}(\log(n_{\mathcal{I}}) \cdot 2^{n_{\mathcal{T}}} \cdot \log(n_{\mathcal{B}}))$. Since the transition relation of this automaton can be computed via consulting the given $\mathcal{R}$, the space requirement of the algorithm is primarily restricted by the size to store a constant amount of these states. $\square$

2.4 Concrete interpretations

As we have noted in Remark 2.2, we believe that one should fix some interpretations to provide a starting point for the analysis of RTS. In this section, we will introduce three interpretations and explore them in more detail. The way we decide which interpretations to consider is mostly motivated by previous work. In particular, we rely for our choice on the promising results in [ERW21b; Boz+20; ERW22b] which, in turn, are inspired by [Esp+14; EM00]. Another benefit of the chosen interpretations is that they are relatively simple; that is, they have 2 or 3 states, and they all share the same alphabet 2^{Σ} for their statements.

Traps

We already introduced one of the interpretations we are interested in; that is, the interpretation for one disjunctive clause as depicted in Figure 2.7. We call this interpretation the *trap* interpretation. Let us convey the intuition behind the name. For this, we introduce (informally) an alternative view on instances of some RTS. Initially, we fix the size of the instance as n. Here, we refer to a "value" as a tuple $\langle i, \sigma \rangle \in \{1, \dots, n\} \times \Sigma$. One can think of a value as the state of one single agent in the configuration. Therefore, we can identify any configuration $u_1 \ \dots \ u_n$ with a set of values $\lbag u_1 \ \dots \ u_n \rbag = \bigcup_{1 \leq i \leq n} \{\langle i, u_i \rangle\}$. Any statement $I_1 \ \dots \ I_n$ can be understood, similarly, as a collection of values $\lbag I_1 \ \dots \ I_n \rbag = \bigcup_{1 \leq i \leq n} \{i\} \times I_i$. For example, $\lbag t \ n \ t \rbag = \{\langle 1, t \rangle, \langle 2, n \rangle, \langle 3, t \rangle\}$ and $\lbag \emptyset \ \{t\} \ \{t, n\} \rbag = \{\langle 2, t \rangle, \langle 3, t \rangle, \langle 3, n \rangle\}$. The trap interpretation relates a configuration u and statement I if and only is $\lbag u \rbag \cap \lbag I \rbag \neq \emptyset$. Intuitively, once a configuration has some value in the inductive statement, it cannot remove all its values again from it – it gets "trapped". In the following, we refer to an inductive statement of the trap interpretation as a *trap*.

Siphon

We illustrate another interpretation in Figure 2.10 which we call the *siphon* interpretation. A trap I is satisfied by any configuration u if at least one of its values is part of $\lbag I \rbag$. Intuitively, a siphon I' requires that none of its values is part of the configuration that satisfies I'. That is, $u \models_{\mathcal{V}_{siphon}} I'$ if and only if $\lbag u \rbag \cap \lbag I' \rbag = \emptyset$.

In other words, if any configuration u does not share a value with a siphon (or, more specifically, an inductive statement of the siphon interpretation) then one can only reach configurations from u for which this is also true. Let us give an example of this concept.

Example 2.8: *Winning the lottery with siphons.*

Assume we have an array of agents. All of them draw a ticket. Exactly one of the tickets is winning. For this protocol, the agents pass a token down the line from the first to the last agent. However, the agent with the winning ticket marks the token in some way.
Let us model this lottery with an RTS $\mathcal{R}$. To this end, we introduce the alphabet $\Sigma = \{\ell, w, \underline{\ell}, \underline{w}, \overline{\ell}, \overline{w}\}$. Here, the letters express the following states:

ℓ: This describes an agent who drew a losing ticket.

w: This describes an agent who drew a winning ticket.

$\underline{x}$: This describes an agent with ticket x who currently holds an unmarked token.

$\overline{x}$: This describes an agent with ticket x who currently holds a marked token.

Initially, the first agent holds an unmarked token and exactly one of them drew a winning ticket. Therefore, we choose the initial language to be $\underline{\ell}\ \ell^*\ w\ \ell^* | \underline{w}\ \ell^*$. There are now different transitions to consider – depending on what ticket the agent with the token has. Let us introduce, first, an auxiliary notion that describes that no change occurs at that position; that is, $DC = \left(\begin{bmatrix} \ell \\ \ell \end{bmatrix} | \begin{bmatrix} w \\ w \end{bmatrix} | \begin{bmatrix} \underline{\ell} \\ \underline{\ell} \end{bmatrix} | \begin{bmatrix} \underline{w} \\ \underline{w} \end{bmatrix} | \begin{bmatrix} \overline{w} \\ \overline{w} \end{bmatrix} | \begin{bmatrix} \overline{\ell} \\ \overline{\ell} \end{bmatrix} \right)$. We separate the transitions into two cases:

- $DC^* \left(\begin{bmatrix} \underline{\ell} \\ \ell \end{bmatrix} \left(\begin{bmatrix} \ell \\ \underline{\ell} \end{bmatrix} | \begin{bmatrix} w \\ \underline{w} \end{bmatrix} \right) | \begin{bmatrix} \overline{\ell} \\ \ell \end{bmatrix} \left(\begin{bmatrix} \ell \\ \overline{\ell} \end{bmatrix} | \begin{bmatrix} w \\ \overline{w} \end{bmatrix} \right) \right) DC^*$ models an agent with a losing ticket passing down the token unchanged.
- In contrast, $DC^* \left(\begin{bmatrix} \underline{w} \\ w \end{bmatrix} | \begin{bmatrix} \overline{w} \\ w \end{bmatrix} \right) \left(\begin{bmatrix} \ell \\ \overline{\ell} \end{bmatrix} | \begin{bmatrix} w \\ \overline{w} \end{bmatrix} \right) DC^*$ models an agent with the winning ticket passing down a marked token.

In this example, it must be impossible for an unchanged token to reach the last agent if this last agent is not the one with the winning ticket. More formally, we expect that no word in $\Sigma^*\ \underline{\ell}$ can be reached.

Indeed, we can prove this by observing $\emptyset^* \ \{\ell, \underline{\ell}, \overline{\ell}\} \ \{\underline{\ell}\}^* \subseteq \text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R})$. First, let us consider the statements that are encoded in this language. In every word of this language, there is exactly one position where the letter is $\{\ell, \underline{\ell}, \overline{\ell}\}$. We call this position the *barrier*. Intuitively, any word of this language encodes the statement "there is no agent with a losing ticket that holds an unmarked token after the barrier". To verify this, recall that the siphon interpretation accepts if and only if the i-th letter in the configuration is not part of the i-th letter of the statement for all i.

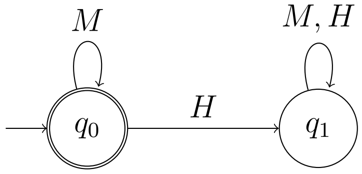
We need to verify that all the statements in this language are siphons. Consider, for any statement of this language, any position j after its barrier. Any transition t that changes the j-th letter into $\underline{\ell}$ changes the state of the $j-1$-th agent from $\underline{\ell}$ to ℓ. Since the statement does not allow the $j-1$-th letter to be $\underline{\ell}$ (as $\underline{\ell} \in \{\ell, \underline{\ell}, \overline{\ell}\}$ if $j-1$ is the barrier and $\underline{\ell} \in \{\underline{\ell}\}$ if it is not), the source of t cannot satisfy the statement. For the barrier itself, one can simply observe that no transition changes which lottery ticket the agent drew initially.

Every initial configuration has exactly one agent, say at position i, in either the state w or $\underline{w}$. Choose the statement of the siphons we introduced where the barrier is i. This initial configuration satisfies the chosen statement and proves the desired property. In this way, we established that, for this parameterized system,

> "there is no agent with a losing ticket that holds an unmarked token after the agent with the winning ticket".

Figure 2.10: *Illustration of $\mathcal{V}_{siphon}$.*

Again, we denote with H all pairs in $\langle v, I\rangle \in \Sigma \times 2^{\Sigma}$ such that $v \in I$ and M all pairs in $\langle v, I\rangle \in \Sigma \times 2^{\Sigma}$ such that $v \notin I$.

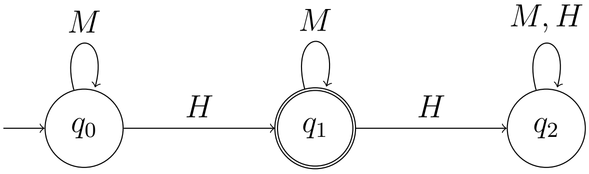


Flow

The third and last interpretation we are interested in is the *flow* interpretation $\mathcal{V}_{\mathit{flow}}$. This time, we want that *exactly at one position* the letter of the configuration is part of the set in the same position in the encoded statement. In other words, $u \models_{\mathcal{V}_{\mathit{flow}}} I$ if and only if $|\lbag u \rbag \cap \lbag I \rbag| = 1$. For example, for $\Sigma = \{a, b\}$, $a\ b\ a\ a\ a \models_{\mathcal{V}_{\mathit{flow}}} \{b\}\ \{b\}\ \{b\}\ \{b\}\ \{b\}$ while $a\ b\ a\ b\ a \not\models_{\mathcal{V}_{\mathit{flow}}} \{b\}\ \{b\}\ \{b\}\ \{b\}\ \{b\}$. In fact, the statement $\{b\}\ \{b\}\ \{b\}\ \{b\}\ \{b\}$ essentially states that the configuration contains exactly one b. We depict the automaton for this interpretation in Figure 2.11.

Figure 2.11: *Illustration of* $\mathcal{V}_{\mathit{flow}}$.

We denote with H all pairs in $\langle v, I\rangle \in \Sigma \times 2^{\Sigma}$ such that $v \in I$ and M all pairs in $\langle v, I\rangle \in \Sigma \times 2^{\Sigma}$ such that $v \notin I$.



Let us illustrate this interpretation in more detail with an example.

Example 2.9: *Flowing through previous examples.*

Recall the formalization of the lottery $\mathcal{R}$ from Example 2.8. One can verify that, in this example, there is at every moment in time exactly one agent who initially drew the winning ticket. This can be expressed via a language of statements for the interpretation $\mathcal{V}_{\mathit{flow}}$: namely, $\{w, \overline{w}, \underline{w}\}^* \subseteq \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$. Similarly, one can express that there is exactly one token in all reachable configurations with a similar language $\left\{\overline{\ell}, \underline{\ell}, \overline{w}, \underline{w}\right\}^*$.
Let us go back to the token passing algorithm of Example 2.2. The initial language of the formalization $\mathcal{R}$ was $t\ n^*$ and the language of all transitions was $\begin{bmatrix} n \\ n \end{bmatrix}^* \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix}^*$. In order to prove that there exists exactly one token at any moment in time we used the interpretation $\mathcal{V}_{\mathit{trap}}$ and the two languages of inductive statements $\emptyset^*\ \{n\}\ \emptyset^*\ \{n\}\ \emptyset^*$ and $\{t\}^*$. One should note here that the

transitions implicitly encode that there is exactly one t – otherwise no transition is applicable. However, since there is initially only one token, one could model a system with the same behavior by choosing $\left(\begin{bmatrix} n \\ n \end{bmatrix} | \begin{bmatrix} t \\ t \end{bmatrix}\right)^* \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \left(\begin{bmatrix} n \\ n \end{bmatrix} | \begin{bmatrix} t \\ t \end{bmatrix}\right)^*$ as transitions. Although the behavior of the system did not change, one cannot establish anymore (using only inductive statements for $\mathcal{V}_{trap}$) that there is exactly one token in every reachable configuration: To this end, assume that we want to prove that $t\ n\ t$ is unreachable from $t\ n\ n$. If this is possible, then there would be $I_1\ I_2\ I_3 \in \text{Inductive}_{\mathcal{V}_{trap}}$ such that $t\ n\ n \models_{\mathcal{V}_{trap}} I_1\ I_2\ I_3$ but $t\ n\ t \not\models_{\mathcal{V}_{trap}} I_1\ I_2\ I_3$. This is only possible if $n \in I_3$ – exploiting the only difference between the two configurations. This statement must be inductive with respect to the transitions. In particular, consider the transition $\begin{bmatrix} t \\ t \end{bmatrix} \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix}$. The target of this transition is the undesired configuration while the origin satisfies the statement because $n \in I_3$. This is a contradiction to the choice of $I_1\ I_2\ I_3$ since it must be inductive but also disallow $t\ n\ t$.

However, we can consider the statements $\{t\}^*$ for $\mathcal{V}_{flow}$. Essentially, the statement $\{t\}^n$ encodes (w. r. t. $\mathcal{V}_{flow}$) that all configurations of the instance of size n have exactly one token. Since all transitions of this instance do not alter the number of tokens, the statement is inductive. Moreover, it trivially establishes that one can only reach configurations in this instance with exactly one token. This argument holds for transitions $\left(\begin{bmatrix} n \\ n \end{bmatrix} | \begin{bmatrix} t \\ t \end{bmatrix}\right)^* \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \left(\begin{bmatrix} n \\ n \end{bmatrix} | \begin{bmatrix} t \\ t \end{bmatrix}\right)^*$ and $\begin{bmatrix} n \\ n \end{bmatrix}^* \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix}^*$.

Let us close this section with the formal definition of all these interpretations. For clarity, we refer the reader to the illustrations of those interpretations in Figure 2.7, Figure 2.10, and Figure 2.11.

Definition 2.8: *Concrete interpretations.*

Let us fix one RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$. We introduce three different interpretations: $\mathcal{V}_{trap}$, $\mathcal{V}_{siphon}$, and $\mathcal{V}_{flow}$. They are defined as

$$\mathcal{V}_{trap} = \left\langle \{q_0, q_1\}, q_0, 2^{\Sigma}, \delta, \{q_1\} \right\rangle \text{ with } \delta(q, \langle \sigma, I \rangle) = \begin{cases} q_0 & \text{if } q = q_0 \text{ and } \sigma \notin I \\ q_1 & \text{otherwise} \end{cases}$$

$$\mathcal{V}_{\mathit{siphon}} = \left\langle \{q_0, q_1\}, q_0, 2^{\Sigma}, \delta, \{q_0\} \right\rangle \text{ with } \delta(q, \langle \sigma, I \rangle) = \begin{cases} q_0 & \text{if } q = q_0 \text{ and } \sigma \notin I \\ q_1 & \text{otherwise} \end{cases}$$

$$\mathcal{V}_{\mathit{flow}} = \left\langle \{q_0, q_1, q_2\}, q_0, 2^{\Sigma}, \delta, \{q_1\} \right\rangle \text{ with } \delta(q, \langle \sigma, I \rangle) = \begin{cases} q_0 & \text{if } q = q_0 \text{ and } \sigma \notin I \\ q_1 & \text{if } q = q_0 \text{ and } \sigma \in I \\ q_1 & \text{if } q = q_1 \text{ and } \sigma \notin I \\ q_2 & \text{if } q = q_1 \text{ and } \sigma \in I \\ q_2 & \text{if } q = q_2 \end{cases}$$

2.5 Abstractions are (PSpace-)hard

In Theorem 2.2, we established that Problem 2.2 can be solved in ExpSpace for any interpretation. We ask ourselves if we can find better algorithms for the specific interpretations $\mathcal{V}_{\mathit{trap}}$, $\mathcal{V}_{\mathit{siphon}}$, and $\mathcal{V}_{\mathit{flow}}$. First, however, we are focusing on lower bounds for this problem.

In this section, we establish that Problem 2.2 is PSpace-hard for the interpretations $\mathcal{V}_{\mathit{trap}}$, $\mathcal{V}_{\mathit{siphon}}$, and $\mathcal{V}_{\mathit{flow}}$. We do this in three steps:

- First, we prove that Problem 2.2 is essentially the same for the interpretations $\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$; that is, one can reduce the problem in polynomial time in both directions (Theorem 2.3).
- Second, we prove PSpace hardness of Problem 2.2 for the interpretation $\mathcal{V}_{\mathit{siphon}}$ (Theorem 2.4).
- Finally, we prove PSpace hardness of Problem 2.2 for the interpretation $\mathcal{V}_{\mathit{flow}}$ (Theorem 2.5).

$\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$ are equally hard

Here, we prove that Problem 2.2 is essentially the same for the interpretations $\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$. First, let us illustrate the crucial ideas informally. Remember that $u \models_{\mathcal{V}_{\mathit{trap}}} I$ if and only if $\lbag u \rbag \cap \lbag I \rbag \neq \emptyset$ and $u \models_{\mathcal{V}_{\mathit{siphon}}} I$ if and only if $\lbag u \rbag \cap \lbag I \rbag = \emptyset$. Consequently, $u \models_{\mathcal{V}_{\mathit{trap}}} I$ if and only if $u \not\models_{\mathcal{V}_{\mathit{siphon}}} I$. This observation can be exploited further. To this end, let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ be an RTS. Consider some word[6] $I \in \left(2^\Sigma\right)^n$. If we look at I as a statement for $\mathcal{V}_{trap}$, then a transition $\langle u, v \rangle$ shows that I is *not* inductive if $u \models_{\mathcal{V}_{trap}} I$ and $v \not\models_{\mathcal{V}_{trap}} I$. On the other hand, the "flipped" transition $\langle v, u \rangle$ shows that I is *not an inductive statement for* $\mathcal{V}_{siphon}$ (because $u \not\models_{\mathcal{V}_{siphon}} I$ and $v \models_{\mathcal{V}_{siphon}} I$). More generally, we will establish that words in $\left(2^\Sigma\right)^*$ are traps[7] if and only if they are siphons in the RTS where all transitions are "flipped".

We make these observations now more precise. Thus, we introduce the concept of flipped transitions first. Then, we establish that traps and siphons coincide if one flips the transition of any RTS. Finally, we give the complete reduction which, at that point, is straightforward.

Definition 2.9: *Flipped relations and transducers.*

For any relation $X \subseteq \bigcup_{n \geq 0} \Sigma^n \times \Gamma^n$ we denoted with $\overset{\leftrightharpoons}{X}$ the relation $\{\langle v, u \rangle : \langle u, v \rangle \in X\}$.
For any Σ-Γ-transducer $\mathcal{T} = \langle Q, q_0, \Sigma \times \Gamma, \Delta, F \rangle$ we denote with $\overset{\leftrightharpoons}{\mathcal{T}}$ the Γ-Σ-transducer $\langle Q, q_0, \Sigma \times \Sigma, \nabla, F \rangle$ where $\left\langle q, \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix}, p \right\rangle \in \Delta$ if and only if $\left\langle q, \begin{bmatrix} \sigma_2 \\ \sigma_1 \end{bmatrix}, p \right\rangle \in \nabla$.

From the definition, one can immediately see that the flipped relation of a transducer is recognized by the "flipped" transducer.

Lemma 2.7. *If $\mathcal{T}$ is a Σ-Γ-transducer, then* $\left[\!\left[\overset{\leftrightharpoons}{\mathcal{T}}\right]\!\right] = \overset{\leftrightharpoons}{[\![\mathcal{T}]\!]}$.

Considering our initial intuition, the following result is expected.

Lemma 2.8. *For any $I \in \left(2^\Sigma\right)^n$ holds $w \models_{\mathcal{V}_{trap}} I$ if and only if $w \not\models_{\mathcal{V}_{siphon}} I$ for all $w \in \Sigma^n$.*

Proof. If $w_1 \ \ldots \ w_n \models_{\mathcal{V}_{trap}} I_1 \ \ldots \ I_n$ then there exists $1 \leq i \leq n$ with $w_i \in I_i$. This, however, implies immediately $w_1 \ \ldots \ w_n \not\models_{\mathcal{V}_{siphon}} I_1 \ \ldots \ I_n$.

On the other hand, if $w_1 \ \ldots \ w_n \not\models_{\mathcal{V}_{trap}} I_1 \ \ldots \ I_n$ then $w_i \notin I_i$ for all $1 \leq i \leq n$. Consequently, $w_1 \ \ldots \ w_n \models_{\mathcal{V}_{siphon}} I_1 \ \ldots \ I_n$. □

Combining these two results shows that traps in RTSs are siphons if all transitions are flipped and vice versa.

---

[6] We deliberately do *not* say statement because the word is not associated with an interpretation *yet*.

[7] Remember that we call inductive statements for $\mathcal{V}_{trap}$ traps.

Lemma 2.9. *For all* $\mathcal{R}_1 = \langle \Sigma, \mathcal{I}_1, \mathcal{T}_1 \rangle$ *and* $\mathcal{R}_2 = \langle \Sigma, \mathcal{I}_2, \mathcal{T}_2 \rangle$ *such that* $\mathcal{T}_1 = \overset{\leftrightharpoons}{\mathcal{T}}_2$

$$\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R}_1) = \text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R}_2).$$

Proof. Pick any $I \in \overline{\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R}_1)}$. Therefore, there is $w \rightsquigarrow_{\mathcal{T}} u$ with $w \models_{\mathcal{V}_{trap}} I$ but $u \not\models_{\mathcal{V}_{trap}} I$. Per Lemma 2.8 this means $u \models_{\mathcal{V}_{siphon}} I$ and $w \not\models_{\mathcal{V}_{siphon}} I$. Because of Lemma 2.7 we have $u \rightsquigarrow_{\mathcal{T}_2} w$ and, thus, $I \in \overline{\text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R}_2)}$. The same reasoning applies in the other direction. Consequently, $\overline{\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R}_1)} = \overline{\text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R}_2)}$ and, also, $\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R}_1) = \text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R}_2)$. ☐

Similarly, we can now establish that the potential reachability relation that is induced by the two interpretations $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ can be flipped as well.

Lemma 2.10. *For all* $\mathcal{R}_1 = \langle \Sigma, \mathcal{I}_1, \mathcal{T}_1 \rangle$ *and* $\mathcal{R}_2 = \langle \Sigma, \mathcal{I}_2, \mathcal{T}_2 \rangle$ *such that* $\mathcal{T}_1 = \overset{\leftrightharpoons}{\mathcal{T}}_2$

$$u \Rightarrow_{\mathcal{V}_{trap}} v \textit{ in } \mathcal{R}_1 \textit{ if and only if } v \Rightarrow_{\mathcal{V}_{siphon}} u \textit{ in } \mathcal{R}_2.$$

Proof. If $u \not\Rightarrow_{\mathcal{V}_{trap}} v$ in $\mathcal{R}_1$ then there exists $I \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R}_1)$ such that $u \models_{\mathcal{V}_{trap}} I$ but $v \not\models_{\mathcal{V}_{trap}} I$. Another application of Lemma 2.8 yields that $u \not\models_{\mathcal{V}_{siphon}} I$ and $v \models_{\mathcal{V}_{siphon}} I$. By Lemma 2.9 we get $I \in \text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R}_2)$ and, consequently, $v \not\Rightarrow_{\mathcal{V}_{siphon}} u$ in $\mathcal{R}_2$. Again, use the symmetry of all the lemmas to obtain that $v \not\Rightarrow_{\mathcal{V}_{siphon}} u$ in $\mathcal{R}_2$ necessarily means $u \not\Rightarrow_{\mathcal{V}_{trap}} v$ in $\mathcal{R}_1$. The statement follows. ☐

This leads immediately to the following reduction.

Theorem 2.3. *Problem 2.2 with* $\mathcal{V}_{trap}$ *can be reduced in polynomial space and time to Problem 2.2 with* $\mathcal{V}_{siphon}$ *and vice versa.*

Proof. For any instance $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and $\mathcal{B}$ for Problem 2.2 with $\mathcal{V}_{trap}$ construct

$$\mathcal{R}' = \left\langle \Sigma, \mathcal{B}, \overset{\leftrightharpoons}{\mathcal{T}} \right\rangle$$

and consider an instance of Problem 2.2 with $\mathcal{V}_{siphon}$ with $\mathcal{R}'$ and $\mathcal{I}$ as the automaton for the undesired configurations.

The correctness of this reduction is an immediate consequence of Lemma 2.10. The reduction in the other direction is symmetric. ☐

From this result, we can conclude that Problem 2.2 is equally complex for the interpretations $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$.

A PSpace-hard problem

For the following reduction, we use a PSpace-hard problem for Turing machines. Therefore, we introduce Turing machines now.

Definition 2.10: *Turing Machine.*

A Turing machine $\mathcal{M} = \langle Q, q_0, \Gamma, B, \delta \rangle$ is defined by

- a finite set of states Q
- where one of which is a dedicated initial state q_0,
- a finite set of letters Γ
- where one of which is a dedicated blank value B, and
- a transition function $\delta\colon Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \downarrow, \rightarrow\}$.

For any word $x \in \Gamma^+$, one can consider the behavior of $\mathcal{M}$ on that word. Conceptionally, the Turing machine maintains its current state (which is initially q_0) and a pointer into the word x (which is initially on the first letter of it). In every step of the computation of the machine, the transition function δ decides the behavior of $\mathcal{M}$. For example, assume the machine currently is in state q and the pointer points to the i-th letter u of x. If $\delta(q, u) = \langle p, v, \rightarrow \rangle$, then $\mathcal{M}$ replaces the i-th letter of x with v, changes its state to p, and moves its pointer one letter to the right. We assume that, if the machine ever moves its pointer out of the scope of the current word, then a B is silently added. We capture this intuition with the following definition:

Definition 2.11: *A run of a Turing machine.*

Let $\mathcal{M} = \langle Q, q_0, \Gamma, B, \delta \rangle$ be a Turing machine. We refer to a word $x_1 \;\ldots\; x_{i-1} \;\langle q, x_i \rangle\; x_{i+1} \;\ldots\; x_n$ where $x_j \in \Gamma$ for all $1 \leq j \leq n$ and $q \in Q$ as an *arrangement* of $\mathcal{M}$. In particular, we say that the *head position* is i, the *current state* is q, and the *content of cell* j is x_j. Also, we call n the *length of the arrangement* and the word $x_1 \ \dots \ x_n$ the *content of the tape.* For any arrangement α, we refer to the content of the cell j as $\alpha[j]$.

Consider an arrangement α where the head position is i, the current state is q, $\alpha[i]$ is x, the length is n and $\delta(q, x) = \langle p, y, m\rangle$. We say the arrangement β follows α (denoted with $\alpha \rightarrowtail \beta$) if it meets any of the following criteria:

No Overflow If $i \neq 1$ or $m \neq\leftarrow$, and $i \neq n$ or $m \neq\rightarrow$, and

- the length of β is n,
- the head position of β is $\begin{cases} i-1 & \text{if } m =\leftarrow \\ i & \text{if } m =\downarrow \\ i+1 & \text{if } m =\rightarrow \end{cases}$,
- the state of β is p, and
- $\beta[i]$ is y while $\alpha[j] = \beta[j]$ for all $1 \leq j \leq n$ with $i \neq j$.

Overflow Left If $i = 1$ and $m =\leftarrow$, and

- the length of β is $n + 1$,
- the head position of β is 1,
- the state of β is p, and
- $\beta[i]$ is y while $\alpha[j] = \beta[j+1]$ for all $1 < j \leq n$ with $i \leq j$ and $\beta[1] = B$.

Overflow Right If $i = n$ and $m =\rightarrow$, and

- the length of β is $n + 1$,
- the head position of β is $n + 1$,
- the state of β is p, and
- $\beta[i]$ is y while $\alpha[j] = \beta[j]$ for all $1 \leq j \leq n$ with $i \leq j$ and $\beta[n+1] = B$.

Moreover, we call, for any word $x_1 \ \dots \ x_m \in (\Gamma \setminus \{B\})^*$, the infinite sequence of arrangements $\alpha_0, \alpha_1, \dots$ such that α_{i+1} follows α_i and α_0 is an arrangement where

- the length of α_0 is m,
- the head position of α_0 is 1,

- the state of α_0 is q_0, and
- the content of the tape of α_0 is $x_1 \ \dots \ x_m$

the *run* of $\mathcal{M}$ on $x_1 \ \dots \ x_m$. We say $\mathcal{M}$ *reaches* an arrangement on $x_1 \ \dots \ x_m$ if it is part of this run.

Example 2.10: *A Turing machine.*

Let us introduce a Turing machine as a running example. This machine moves over a binary word from left to right and flips all bits. Once this machine encounters the end of the binary word it moves from right to left to start the process of flipping the bits again. This second phase is similar to an actual "carriage return" of a typewriter.

We fix $Q = \{q_\rightarrow, q_\leftarrow\}$ and $\Gamma = \{0, 1, B\}$. The initial state is $q_\rightarrow$. It remains to define δ; we do so, by setting

- $\delta(q_\rightarrow, 0) = \langle q_\rightarrow, 1, \rightarrow\rangle$,
- $\delta(q_\rightarrow, 1) = \langle q_\rightarrow, 0, \rightarrow\rangle$,
- $\delta(q_\rightarrow, B) = \langle q_\leftarrow, B, \leftarrow\rangle$,
- $\delta(q_\leftarrow, 0) = \langle q_\leftarrow, 0, \leftarrow\rangle$,
- $\delta(q_\leftarrow, 1) = \langle q_\leftarrow, 1, \leftarrow\rangle$, and
- $\delta(q_\leftarrow, B) = \langle q_\rightarrow, B, \rightarrow\rangle$.

Formally, we obtain $\mathcal{M} = \langle Q, q_\leftarrow, \Gamma, B, \delta\rangle$.

The run of $\mathcal{M}$ on 0 1 0 begins with

$$\begin{gathered}\langle q_\rightarrow, 0\rangle \ 1 \ 0 \rightarrowtail 1 \ \langle q_\rightarrow, 1\rangle \ 0 \rightarrowtail 1 \ 0 \ \langle q_\rightarrow, 0\rangle \rightarrowtail 1 \ 0 \ 1 \ \langle q_\rightarrow, B\rangle \\ \rightarrowtail 1 \ 0 \ \langle q_\leftarrow, 1\rangle \ B \rightarrowtail 1 \ \langle q_\leftarrow, 0\rangle \ 1 \ B \rightarrowtail \langle q_\leftarrow, 1\rangle \ 0 \ 1 \ B \rightarrowtail \langle q_\leftarrow, B\rangle \ 1 \ 0 \ 1 \ B.\end{gathered}$$

In this section, we prove different variations of Problem 2.2 PSPACE-hard. We do so by reducing some problem which is known to be PSPACE-hard to our instances of Problem 2.2. Or, more precisely, we give a sequence of reductions that start in the following PSpace-hard problem [Pap94, Theorem 19.9]:

Problem 2.3.

Given: *A Turing machine $\mathcal{M}$, an input word x, and a state q_f of $\mathcal{M}$*

Compute: *Does $\mathcal{M}$ reach an arrangement on x where the state is q_f before an arrangement with length $> |x|$?*

We want to simplify this problem slightly by only considering inputs x for which the sequence of arrangements of $\mathcal{M}$ is of constant size; that is, $|x|$. We call this property *boundedness*. Roughly speaking, this means that the machine must not move its head past its input.

Definition 2.12: *Bounded Turing machine.*

We call a Turing $\mathcal{M}$ *bounded* on input x if $\mathcal{M}$ does not reach an arrangement on x of any length but $|x|$.

One can immediately verify that the machine from Example 2.10 is not bounded on the input 010. In fact, this machine is not bounded on any input. We illustrate how to remedy this in a moment. First, however, we want to introduce the problem we are using for our reductions. It is essentially Problem 2.3 but with the guarantee that the machine is bounded on the given input.

Problem 2.4.

Given: *A Turing machine $\mathcal{M}$ that is bounded on x and a state q_f*

Compute: *Does $\mathcal{M}$ reach an arrangement on x with state q_f?*

The reduction from Problem 2.3 to Problem 2.4 can be achieved via standard methods: Roughly speaking, one can modify the machine of Problem 2.3 by adding a new letter # to its alphabet, and introducing a new initial state q' and a new final state p'. Moreover, we add $\delta(q_f, \gamma) = \langle p, \gamma, \downarrow\rangle$ for all $\gamma \in \Gamma$[8], $\delta(q, \#) = \langle q, \#, \downarrow\rangle$ for all states but q' and $\delta(q', \#) = \langle q_0, \#, \rightarrow\rangle$ where q_0 is the initial state of the original machine. Since q' cannot be reached after the first step, the values of δ for q' and any other letter are immaterial. We call this altered machine $\mathcal{M}'$. Consider now the instance of $\mathcal{M}'$, $\# \; x \; \#$ and p' for

---

[8] In other words, there is one “stutter” in the machine before reaching its final state. However, the machine does not move into the final state on the letter # but, there, it “stutters” indefinitely in the previously final state q_f.

Problem 2.3. $\mathcal{M}'$ can reach the final state p' only if it reaches the final state q_f of the machine $\mathcal{M}$ on any letter but #. Because $\mathcal{M}'$ “deadlocks” if it encounters the letter # in any step but the first, it does so, in particular, if the original machine ever left its input and, then, it cannot reach an arrangement with state p' anymore. Thus, the answer for this instance for Problem 2.4 coincides with the answer of the original instance for Problem 2.3.

Example 2.11: *A bounded Turing machine.*

We construct a variant of the Turing machine introduced in Example 2.10 where we introduce # as a new symbol that can be used as a delimiter for the input. Thus, we fix $\Gamma = \{0, 1, \#, B\}$ and $Q = \{q_0, q_\rightarrow, q_\leftarrow\}$ where q_0 is the new initial state. The definition of δ changes slightly:

- $\delta(q_0, \#) = \langle q_\rightarrow, \#, \rightarrow\rangle$, and $\delta(q_0, x) = \langle q_\rightarrow, x, \downarrow\rangle$ for all other letters x.
- As before we have $\delta(q_\rightarrow, 0) = \langle q_\rightarrow, 1, \rightarrow\rangle$, $\delta(q_\rightarrow, 1) = \langle q_\rightarrow, 0, \rightarrow\rangle$ and $\delta(q_\rightarrow, B) = \langle q_\leftarrow, B, \leftarrow\rangle$, and $\delta(q_\leftarrow, 0) = \langle q_\leftarrow, 0, \leftarrow\rangle$, $\delta(q_\leftarrow, 1) = \langle q_\leftarrow, 1, \leftarrow\rangle$ and $\delta(q_\leftarrow, B) = \langle q_\rightarrow, B, \rightarrow\rangle$.
- On the other hand, we expand the behavior for the letter # as expected with $\delta(q_\rightarrow, \#) = \langle q_\leftarrow, \#, \leftarrow\rangle$ and $\delta(q_\leftarrow, \#) = \langle q_\rightarrow, \#, \rightarrow\rangle$.

With this definition, the machine is bounded on every input that starts with any letter but # (since it deadlocks then). Moreover, for every input that starts with # and contains at least one other #, the machine also does not leave its input. In all other cases, it is not bounded because it will move its head past its input by one step to the right.

Abstractions are PSpace-hard for $\mathcal{V}_{siphon}$

For this section, we fix a Turing machine $\mathcal{M} = \langle Q, q_0, \Gamma, B, \delta\rangle$, and an input x on which it is bounded. In the following, we construct a RTS which captures the run of $\mathcal{M}$ on x. For this construction, we add for all arrangements one leading and one trailing B. We do so to avoid some edge cases later.

Since any instance of a RTS is of one fixed size, we cannot represent the infinite run in one single instance. However, we can represent the infinite run via the collection of all instances by letting every instance represent a finite prefix of the run.

A single *step* of the run is a pair of arrangements $\alpha \rightarrowtail \beta$. In our construction, we do not model this step in a single transition in the RTS. Instead, we exploit the fact that the arrangements α and β look, for the most part, the same and only differ in at most two adjacent positions. Roughly speaking[9], the letters of the RTS we want to construct, are the same as the letters we use to represent arrangements; that is, $\Gamma \cup Q \times \Gamma$. But, additionally, we introduce one more letter $\perp$. Intuitively, this new letter represents some *uninitialized* position. To encode the step from α to β directly, we want to build β letter by letter. That is, if $\alpha = x_1 \ \ldots \ x_{i-1} \ \langle q, x_i \rangle \ x_{i+1} \ \ldots \ x_n$, we obtain β gradually from a sequence $\perp \ \ldots \ \perp$ of n uninitialized positions. More specifically, β is constructed in n individual transitions – each of which updates one $\perp$ to the correct letter from $\Gamma \cup Q \times \Gamma$. Let us strengthen this intuition with an example.

Example 2.12: *Micro steps that form a macro step.*

Recall the machine of Example 2.10. The first step of the run of this machine on the input $\# \ 0 \ 1 \ 0 \ \#$ is $B \ \langle q_0, \# \rangle \ 0 \ 1 \ 0 \ \# \ B \rightarrowtail B \ \# \ \langle q_\rightarrow, 0 \rangle \ 1 \ 0 \ \# \ B$. In the RTS that we construct $B \ \# \ \langle q_\rightarrow, 0 \rangle \ 1 \ 0 \ \# \ B$ is obtained from $\perp \perp \perp \perp \perp \perp \perp$ in seven individual steps:

- From $\perp \perp \perp \perp \perp \perp \perp$ to $B \perp \perp \perp \perp \perp \perp$.
- From $B \perp \perp \perp \perp \perp \perp$ to $B \ \# \perp \perp \perp \perp \perp$.
- From $B \ \# \perp \perp \perp \perp \perp$ to $B \ \# \ \langle q_\rightarrow, 0 \rangle \perp \perp \perp \perp$.
- From $B \ \# \ \langle q_\rightarrow, 0 \rangle \perp \perp \perp \perp$ to $B \ \# \ \langle q_\rightarrow, 0 \rangle \ 1 \perp \perp \perp$.
- From $B \ \# \ \langle q_\rightarrow, 0 \rangle \ 1 \perp \perp \perp$ to $B \ \# \ \langle q_\rightarrow, 0 \rangle \ 1 \ 0 \perp \perp$.
- From $B \ \# \ \langle q_\rightarrow, 0 \rangle \ 1 \ 0 \perp \perp$ to $B \ \# \ \langle q_\rightarrow, 0 \rangle \ 1 \ 0 \ \# \perp$.
- From $B \ \# \ \langle q_\rightarrow, 0 \rangle \ 1 \ 0 \ \# \perp$ to $B \ \# \ \langle q_\rightarrow, 0 \rangle \ 1 \ 0 \ \# \ B$.

This means, intuitively, the constructed RTS starts on a configuration $\alpha_0 \ \left(\perp^{|x|+2}\right)^k$. Every $\perp^{|x|+2}$ block of this initial configuration will become one arrangement of the run of the machine on its input. The value k corresponds to the number of steps of the run

---

[9] In fact, the letters of the RTS will add a little bit of bookkeeping which we omit for the moment.

that are considered.

Example 2.13: *The construction of the prefix of a run.*

Picking up the previous example again, we want our RTS to (deterministically) move through the following configurations.

- $B\ \langle q_0, \#\rangle\ 0\,1\,0\,\#\,B \perp \perp \perp \perp \perp \perp \perp$
- $B\ \langle q_0, \#\rangle\ 0\,1\,0\,\#\,B\,B \perp \perp \perp \perp \perp \perp$
- $B\ \langle q_0, \#\rangle\ 0\,1\,0\,\#\,B\,B\,\# \perp \perp \perp \perp \perp$
- $B\ \langle q_0, \#\rangle\ 0\,1\,0\,\#\,B\,B\,\#\ \langle q_\rightarrow, 0\rangle \perp \perp \perp \perp$
- $B\ \langle q_0, \#\rangle\ 0\,1\,0\,\#\,B\,B\,\#\ \langle q_\rightarrow, 0\rangle\ 1 \perp \perp \perp$
- $B\ \langle q_0, \#\rangle\ 0\,1\,0\,\#\,B\,B\,\#\ \langle q_\rightarrow, 0\rangle\ 1\,0 \perp \perp$
- $B\ \langle q_0, \#\rangle\ 0\,1\,0\,\#\,B\,B\,\#\ \langle q_\rightarrow, 0\rangle\ 1\,0\,\# \perp$
- $B\ \langle q_0, \#\rangle\ 0\,1\,0\,\#\,B\,B\,\#\ \langle q_\rightarrow, 0\rangle\ 1\,0\,\#\,B$

Recall that the arrangement of a Turing machine $\mathcal{M}$ changes in one step in at most two positions: only the cell where the head currently is can change its content and the head can change its position to the right or to the left. Therefore, to decide what the letter of an arrangement at some position i is, it suffices to know

- what the content of the i-th cell in this arrangement is,
- whether the head position in this arrangement is i,
- and, if so, what the current state is.

All our arrangements have length $|x| + 2$ since we only consider machines that are bounded on the given input. Therefore, the content of the i-th cell in some arrangement can be deduced from the i-th letter of the previous arrangement because this letter encodes whether the head position of that previous arrangement is at exactly that cell and its content. Moreover, it suffices to inspect the letters of the previous arrangement at positions $i-1$, i, and $i+1$ to know whether the head position in this arrangement is i and, if so, what the state should be.

Example 2.14: *Local information in arrangements.*

Consider now the second step of the run of the machine of Example 2.10 on the input $\#\ 0\ 1\ 0\ \#$: $\#\ \langle q_\rightarrow, 0\rangle\ 1\ 0\ \# \rightarrowtail \#\ 1\ \langle q_\rightarrow, 1\rangle\ 0\ \#$. Again, the second arrangement is constructed letter by letter from five $\bot$.

- The first $\bot$ becomes $\#$ since the first three letters[a] of the previous configuration are $B\ \#\ \langle q_\rightarrow, 0\rangle$. In particular, the content of the cell does not change since the head position is on the second letter. Moreover, δ for $q_\rightarrow$ and 0 indicates that the head moves to the right and not back on the first letter.
- One can easily see that the second $\bot$ becomes 1 by inspecting the first three letters of the previous configuration $\#\ \langle q_\rightarrow, 0\rangle\ 1$. This time, we consult δ for $q_\rightarrow$ and 0 to see that 1 is the content of the cell after the previous arrangement. Additionally, we see that the head position moves one step to the right and, thus, does not remain on the second letter.

In this fashion, one can obtain from the letters in the positions $i-1$, i, and $i+1$ of the previous arrangement the i-th letter of the next arrangement.

---

[a] Here we see the introduction of a leading and a trailing B pay off since we can consider three letters although the leading B will never contribute.

Note that the configurations of the RTS are a seamless enumeration of the arrangements of the run. For arrangements we referred to *positions*; that is, values between 1 and $|x|+2$. For the configurations, we also want to talk about individual letters but for clarity, we use the word *index* here instead.

Example 2.15: *Positions and indices.*

In this configuration, we annotate the positions above and the indices below.

1	2	3	4	5	6	7	1	2	3	4	5	6	7
B	$\langle q_0, \#\rangle$	0	1	0	$\#$	B	B	$\#$	$\langle q_\rightarrow, 0\rangle$	$\bot$	$\bot$	$\bot$	$\bot$
1	2	3	4	5	6	7	8	9	10	11	12	13	14

We need to introduce one last idea before defining the actual RTS. In fact, this idea is closely related to the previous observation. As illustrated in Example 2.13, the individual transitions of RTS should change the first non-initialized symbol $\bot$ of the configuration, say at index i, to the correct letter of the run of $\mathcal{M}$ on x. For this, the transducer for the transitions consults the letters at the indices $(i - (|x| + 2)) - 1$, $(i - (|x| + 2))$, and $(i - (|x| + 2)) + 1$ of the configuration and updates the $\bot$ at index i accordingly. This illustrates a subtle but important notion: in this sequence of arrangements two letters at indices $i - (|x| + 2)$ and i are the same position in two arrangements $\alpha \rightarrowtail \beta$, respectively. From now on we set $n = |x| + 2$.

To obtain a deterministic automaton for these transitions, we mark the index $i - n$ in every configuration such that i is the index of the first $\bot$ of the configuration. Intuitively, the indices $i - n$ and i are the same position j of two arrangements $\alpha \rightarrowtail \beta$. In this way, $i - n$ and i separate the configuration into three sections: a section up to position j in the arrangement before the one we are currently constructing, a section from j the previous arrangement up to j of the current arrangement, and a trailing sequence of $\bot$.

Example 2.16: *The three sections of a configuration.*

In the previous example, we considered the configuration $B\ \langle q_0, \#\rangle\ 0\ 1\ 0\ \#\ B\ B\ \#\ \langle q_{\rightarrow}, 0\rangle\ 1\ \bot\ \bot\ \bot\ \bot$. In this configuration, the next position that is set in the second arrangement is $j = 5$. This corresponds to index twelve in the configuration. Thus, the configuration up to position j of the first arrangement is $B\ \langle q_0, \#\rangle\ 0\ 1$. The section in between is $0\ \#\ B\ B\ \#\ \langle q_{\rightarrow}, 0\rangle\ 1$. Naturally, the sequence of trailing $\bot$ is $\bot\ \bot\ \bot\ \bot$.

For our construction, we want to make these three sections explicit. Therefore, we annotate the letters of the alphabet with two values: f to indicate that they are part of the first section and s for the second section. With this, we conclude our preparation and, finally, give the construction.

The construction to capture a Turing machine as a regular transition system

We still use a Turing machine $\mathcal{M} = \langle Q, q_0, \Gamma, B, \delta\rangle$, an input $x = x_1\ \ldots\ x_m$ on which it is bounded, and $n = |x| + 2$. The alphabet of our RTS is $\Sigma = \{f, s\} \times (\Gamma \cup Q \times \Gamma) \cup \{\bot\}$. For the initial language, we give an automaton that recognizes

$$\underbrace{\langle f, B\rangle\ \langle s, \langle q_0, x_1\rangle\rangle\ \langle s, x_2\rangle\ \ldots\ \langle s, x_m\rangle\ \langle s, B\rangle}_{\text{First arrangement}}\ \underbrace{\langle s, B\rangle\ \bot^{|x|+1}}_{\text{Second arrangement}}\ \underbrace{(\bot^n)^*}_{\text{Other arrangements}}\ .$$

We already give the leading B of the second configuration because the transitions expect a non-empty first section.

The transducer for the transitions executes the following steps:

1. Scan for the first letter in the second section while storing the last read letter.
2. Store the first letter in the second section and expand the first section to this letter.
3. Check that the next letter is part of the second section and determine how to update the first $\perp$.
4. Move n steps checking that all letters are part of the second section and update the first $\perp$ accordingly.

One can construct a DFA that can recognize the initial language with $3 \cdot (n-1) + 1$ and a transducer that recognizes the transitions with

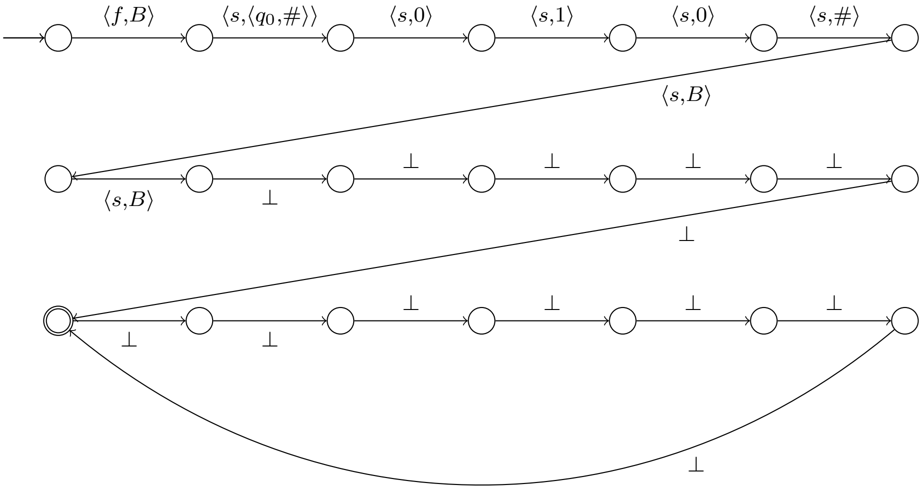
$$\underbrace{1}_{\text{initial state}} + \underbrace{(|Q|+1)\cdot|\Gamma|}_{\text{remember the previously read letter until the second section is found}} + \underbrace{((|Q|+1)\cdot|\Gamma|)^2}_{\text{remember letter before and first letter in second section}} + \underbrace{(|Q|+1)\cdot|\Gamma|\cdot(n-1)}_{\text{carry letter } n-2 \text{ steps and update } \perp} + \underbrace{1}_{\text{final state}} + \underbrace{1}_{\text{sink state}}$$

states. Instead of a formal definition, we give principled automata for the running example:

Example 2.17: *The initial and transducer language for the* $\mathcal{V}_{trap}$ *reduction.*

In the following automata every transition that is not explicitly given leads to a non-accepting sink state.

The initial language

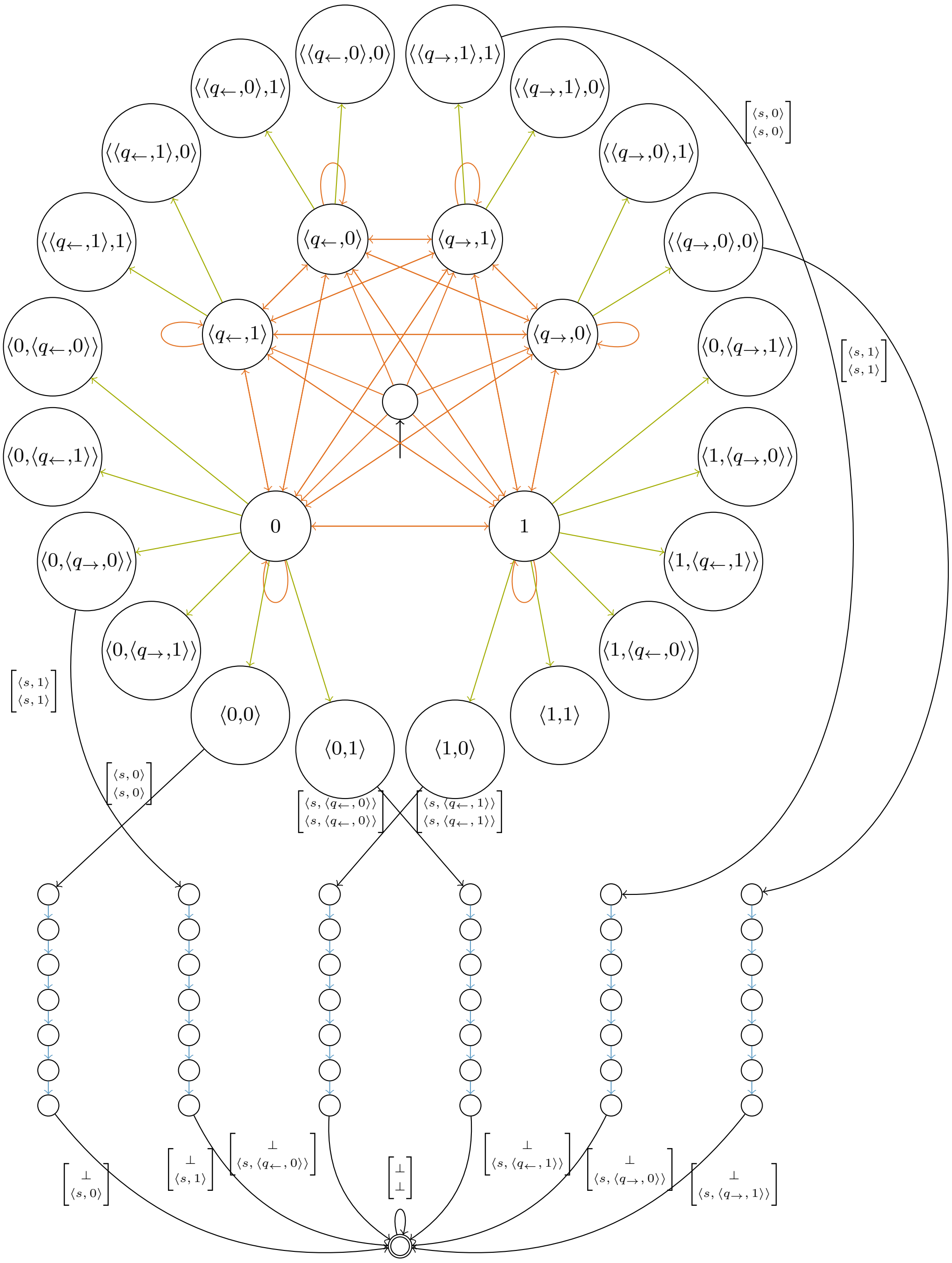


The transition language

We present the transducer of the transitions for the states $q_\rightarrow$ and $q_\leftarrow$, and the letters 0 and 1 in Figure 2.12. The restriction to two states and two letters helps to keep the picture readable. In this picture, the transducer can be separated into an upper and lower part. The upper part of the transducer implements a memory structure for two adjacent letters. On the other hand, the lower part of the transducer is responsible for updating the correct index to the correct letter. The labels of all (orange) transitions to states in the inner circle are $\begin{bmatrix} \langle f,x\rangle \\ \langle f,x\rangle \end{bmatrix}$ where x is the label of the target state. Similarly, the labels of all (green) transitions in the outer circle are $\begin{bmatrix} \langle s,x\rangle \\ \langle f,x\rangle \end{bmatrix}$ where x is the second component of the label of the target state. There the labels on the (blue) transitions in the matrix of nodes are $\begin{bmatrix} \langle s,x\rangle \\ \langle s,x\rangle \end{bmatrix}$ for all $x \in \Sigma \cup (Q \times \Sigma)$. Every column of the matrix is responsible for one of the possible letters (from left to right): $0, 1, \langle q_\leftarrow, 0\rangle, \langle q_\leftarrow, 1\rangle, \langle q_\rightarrow, 0\rangle, \langle q_\rightarrow, 1\rangle$. We only give for every column one transition that connects the memory structure above (which looks like a circle) and the update structure below (which resembles a matrix) to not clutter the picture.

Figure 2.12: *The transducer of the reduction.*

The transducer described in Example 2.17.



The behavior of this RTS is, in a strong sense, deterministic: Since every transition deterministically updates the first $\perp$ while expanding the first section by one step, there is exactly one sequence of configurations possible from every initial configuration. Moreover, every instance has exactly one possible initial configuration. Consequently, every index in every instance changes at most twice; once from $\perp$ into the first section where the letter of the arrangement that this index displays is determined, and from the first to the second section. For instance, consider the run (for simplicity we assume a leading and trailing B for every arrangement here as well) of $\mathcal{M}$ on x:

$$\alpha_1^0 \, \dots \, \alpha_n^0 \rightarrowtail \alpha_1^1 \, \dots \, \alpha_n^1 \rightarrowtail \alpha_1^2 \, \dots \, \alpha_n^2 \rightarrowtail \dots$$

This means, in the instance of size $2 \cdot n$, the letter at index $n+2$ changes from $\perp$ to $\langle f, \alpha_2^1 \rangle$ and, later, from there to $\langle s, \alpha_2^1 \rangle$. In the following, we prove that there is a siphon which enforces that, for every index, only one of these three letters is possible. A siphon encodes this statement by disallowing every other letter than those three at that position.

Definition 2.13: *Divergence in a run.*

For every $\alpha \in \Gamma \cup (\Gamma \times Q)$, we define the *divergence* of α (denoted with $\overset{\times}{\alpha}$) as the set

$$(\{f, s\} \times (\Gamma \cup (\Gamma \times Q)) \cup \{\perp\}) \setminus \{\langle f, \alpha \rangle, \langle s, \alpha \rangle, \perp\}.$$

Intuitively, if we assume the i-th position of the k-th arrangement of the run to be α, then $\overset{\times}{\alpha}$ at index $(k-1) \cdot n + i$ in a statement for $\mathcal{V}_{siphon}$, only allows any of $\langle f, \alpha \rangle$, $\langle s, \alpha \rangle$, or $\perp$ at index $(k-1) \cdot n + i$. Moreover, the index $(k-1) \cdot n + i$ corresponds to the letter of the i-th position of the k-th arrangement of the run in the constructed RTS. This observation, combined with the strong determinism of the constructed RTS, allows the abstraction of inductive statements of $\mathcal{V}_{siphon}$ to be very precise.

Lemma 2.11. *For any $m = k \cdot n$ we have*

$$\overset{\times}{\alpha}{}_1^0 \, \dots \, \overset{\times}{\alpha}{}_n^0 \, \dots \, \overset{\times}{\alpha}{}_1^{k-1} \, \dots \, \overset{\times}{\alpha}{}_n^{k-1} \in \mathrm{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R}).$$

Before the proof let us give an example.

Example 2.18: *Siphons for Example 2.10.*

We fix $m = 2 \cdot n$. Recall that, $B \ \langle q_0, \# \rangle \ 0 \ 1 \ 0 \ \# \ B \rightarrowtail B \ \# \ \langle q_{\rightarrow}, 0 \rangle \ 1 \ 0 \ \# \ B$ are the first two arrangements of the run of our example machine on the input $\# \ 0 \ 1 \ 0 \ \#$. Consequently, we expect

$$\overset{\times}{B} \ \overset{\times}{\langle q_0, \# \rangle} \ \overset{\times}{0} \ \overset{\times}{1} \ \overset{\times}{0} \ \overset{\times}{\#} \ \overset{\times}{B} \ \overset{\times}{B} \ \overset{\times}{\#} \ \overset{\times}{\langle q_{\rightarrow}, 0 \rangle} \ \overset{\times}{1} \ \overset{\times}{0} \ \overset{\times}{\#} \ \overset{\times}{B}$$

to be a siphon.

Proof of Lemma 2.11. Pick one arbitrary

$$I = \overset{\times}{\alpha_1^0} \ \ldots \ \overset{\times}{\alpha_n^0} \ \ldots \ \overset{\times}{\alpha_1^{k-1}} \ \ldots \ \overset{\times}{\alpha_n^{k-1}}.$$

Pick some transition $v \rightsquigarrow_{\mathcal{T}} u$ such that $v \models_{\mathcal{V}_{siphon}} I$. By the definition of the transitions v can be split into three parts $v = F \ S \ R$ such that $F \in \{f\} \times (\Gamma \cup (\Gamma \times Q))^{+}$, $S \in \{s\} \times (\Gamma \cup (\Gamma \times Q))^{n-1}$, and $R \in \bot^{*}$. This, however, entails

- $F = \langle f, \alpha_1^0 \rangle \ \ldots \ \langle f, \alpha_n^0 \rangle \ \ldots \ \left\langle f, \alpha_1^{k'-1} \right\rangle \ \ldots \ \left\langle f, \alpha_i^{k'-1} \right\rangle$ for some $1 \leq i < n - 1$ and $k' < k$, and
- $S = \left\langle s, \alpha_{i+1}^{k'-1} \right\rangle \ \ldots \ \left\langle s, \alpha_n^{k'-1} \right\rangle \ \ldots \ \left\langle s, \alpha_1^{k'} \right\rangle \ \ldots \ \left\langle s, \alpha_j^{k'} \right\rangle$ for some $1 \leq j < n$.

The choice that $i < n - 1$ and $j < n$ is made for notational convenience. Since the other cases only require slightly different choices for some indices below, they are omitted. In particular, using the strong determinism of the constructed RTS, u is fully determined by the structure of v. Notably, the first $\bot$ is deterministically changed to $\left\langle s, \alpha_{j+1}^{k'} \right\rangle$ and the first section is expanded by one step. Consequently, u can be split similarly into $u = F' \ S' \ R'$ such that

- $F' = \langle f, \alpha_1^0 \rangle \ \ldots \ \langle f, \alpha_n^0 \rangle \ \ldots \ \left\langle f, \alpha_1^{k'-1} \right\rangle \ \ldots \ \left\langle f, \alpha_{i+1}^{k'-1} \right\rangle$,
- $S' = \left\langle s, \alpha_{i+2}^{k'-1} \right\rangle \ \ldots \ \left\langle s, \alpha_n^{k'-1} \right\rangle \ \ldots \ \left\langle s, \alpha_1^{k'} \right\rangle \ \ldots \ \left\langle s, \alpha_j^{k'} \right\rangle \ \left\langle s, \alpha_{j+1}^{k'} \right\rangle$, and
- $R' \in \bot^{*}$.

Immediately, one can conclude that $u \models_{\mathcal{V}_{siphon}} I$. $\square$

On this basis, proving the actual PSPACE-hardness of the abstraction via inductive invariants of $\mathcal{V}_{siphon}$ is straightforward. Essentially, it suffices to find some configuration in this abstraction that contains the state q_f; that is, any letter from $\{f, s\} \times \{q_f\} \times \Gamma$, and no occurrence of $\bot$. Note here, that this language can be captured with a DFA $\mathcal{B}$ of constant size. Any configuration of this form describes an encoded prefix of a run of $\mathcal{M}$ on x which contains an arrangement with state q_f and, thus, corresponds to a positive instance of the original problem. On the other hand, this constructed RTS faithfully encodes prefixes of runs of $\mathcal{M}$ on x. Therefore, if an arrangement with state q_f is reached in the run, this RTS allows to encode precisely this run. Consequently, no matter which interpretation is used for the abstraction, the encoding of this run is part of it since it is actually reachable.

Theorem 2.4. *Problem 2.2 is* PSPACE*-hard for* $\mathcal{V}_{siphon}$.

Remark 2.3. *In Remark 2.2 we argue that, for the practical application of this paradigm, one should fix some interpretations that perform well experimentally and do not require the user to provide the interpretations alongside the regular model checking problem. However, one can, of course, consider Problem 2.2 in such a way that the interpretation is part of the input. The constructions in this thesis still show that the problem is in* EXPSPACE *for this variant. It was recently established that this variant is* EXPSPACE*-complete [Kra23]. At this moment in time, we are unable to prove Problem 2.2* EXPSPACE*-complete for any* fixed *interpretation.*

Answering safety questions via $\mathcal{V}_{flow}$

In the following, we prove that Problem 2.2 is PSPACE-hard for $\mathcal{V}_{flow}$. The construction is almost the same as for $\mathcal{V}_{siphon}$. The only thing we change is the language $\mathcal{B}$. First, however, we explore the abstraction of the reachable configurations with inductive statements for $\mathcal{V}_{flow}$.

Recall the observation, illustrated in Example 2.14, that we can obtain the content of the i-th cell in an arrangement from the i-th letter of the previous arrangement. We want to stress that the i-th letter of the previous arrangement encodes multiple information; that is, the content of the cell *and* whether the head position is i and, if so, what the state of the arrangement is. However, the content of the i-th cell does not suffice to determine the i-th letter of the arrangement: Since we do not have the information on what the head position is or what state the arrangement is in, we only know, if the content of the i-th cell is x, that the letter is any of $\{x\} \cup Q \times \{x\}$. We introduce nextContent : $\Gamma \cup (Q \times \Gamma) \to \Gamma$ with

$$\text{nextContent}(x) = \begin{cases} x & \text{if } x \in \Gamma \\ y & \text{if } x \in Q \times \Gamma \text{ such that } \delta(x) = \langle q, y, m \rangle \end{cases}$$

to formalize this notion.

This observation can be encoded as inductive statements for $\mathcal{V}_{\mathit{flow}}$. Roughly speaking, these statements state:

> "If the i-th *letter* of an arrangement is x, then the *content* of the i-th cell in the following arrangement is $\text{nextContent}(x)$."

We capture this via inductive statements for $\mathcal{V}_{\mathit{flow}}$ which encode, if we fix some $x \in \Gamma$ and index i, that exactly one of the following is true:

- the i-th letter of the configuration is $\bot$,
- the i-th letter of the configuration is in $\{s\} \times (\Gamma \cup Q \times \Gamma)$,
- the i-th letter of the configuration is in $\{f\} \times \big((\Gamma \cup Q \times \Gamma) \setminus \text{nextContent}^{-1}(x)\big)$, or
- the $i + n$-th letter of the configuration is in $\{f, s\} \times (\{x\} \cup Q \times \{x\})$.

This leads to the following formalization:

Lemma 2.12. *For every $x \in \Gamma$ holds $\emptyset^*\ A\ \emptyset^{n-1}\ B\ \emptyset^* \subseteq \text{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$ where*

- $A = \{\bot\} \cup \{s\} \times (\Gamma \cup Q \times \Gamma) \cup \{f\} \times \big((\Gamma \cup Q \times \Gamma) \setminus \text{nextContent}^{-1}(x)\big)$, *and*
- $B = \{f, s\} \times (\{x\} \cup Q \times \{x\})$.

Proof. Fix one statement $I = \emptyset^{i-1}\ A\ \emptyset^{n-1}\ B\ \emptyset^j$ and one transition $v \rightsquigarrow_{\mathcal{T}} u$ such that $v \models_{\mathcal{V}_{\mathit{flow}}} I$. By the definition of $\mathcal{T}$, one can separate $v = F\ S\ R$ such that

- $F = \langle f, \alpha_1 \rangle\ \ldots\ \langle f, \alpha_j \rangle$,
- $S = \langle s, \alpha_{j+1} \rangle\ \ldots\ \langle s, \alpha_{j+n} \rangle$, and
- $R = \bot^k$

and $u = F'\ S'\ R'$ such that

- $F' = \langle f, \alpha_1\rangle\ \dots\ \langle f, \alpha_j\rangle\ \langle f, \alpha_{j+1}\rangle$,
- $S' = \langle s, \alpha_{j+2}\rangle\ \dots\ \langle s, \alpha_{j+n+1}\rangle\ \langle s, \alpha_{j+n+1}\rangle$, and
- $R' = \bot^{k-1}$.

Exactly two indices change in every transition: $j+1$ and $j+n+1$. If neither i nor $i+n$ is any of these indices $u \models_{\mathcal{V}_{\mathit{flow}}} I$ immediately because then the i-th and $i+n$-th letters do not differ between v and u.

Case $i + n = j + 1$: In this case, by definition of B, either $\langle s, \alpha_{j+1}\rangle \in B$ and $\langle f, \alpha_{j+1}\rangle \in B$ or $\langle s, \alpha_{j+1}\rangle \notin B$ and $\langle f, \alpha_{j+1}\rangle \notin B$ holds. Because $v \models_{\mathcal{V}_{\mathit{flow}}} I$, $\langle f, \alpha_{j+1-n}\rangle \in A$ follows in the first case. Therefore, $u \models_{\mathcal{V}_{\mathit{flow}}} I$ because the letter at index $j+1-n$ does not differ between v and u. For the same reason, $u \models_{\mathcal{V}_{\mathit{flow}}} I$ in the second case because there $\langle f, \alpha_{j+1-n}\rangle \notin A$.

Case $i = j + 1$ and $i + n = j + 1 + n$: $\langle s, \alpha_{j+1}\rangle \in A$ since $v \models_{\mathcal{V}_{\mathit{flow}}} I$ and $\bot \notin B$. If $\langle s, \alpha_{j+n+1}\rangle \in B$, then, by the definition of the transitions, $\alpha_{j+1} \in \mathrm{nextContent}^{-1}(x)$. Consequently, $u \models_{\mathcal{V}_{\mathit{flow}}} I$ since $\langle f, \alpha_{j+1}\rangle \notin A$ and $\langle s, \alpha_{j+n+1}\rangle \in B$. If, on the other hand, $\langle s, \alpha_{j+n+1}\rangle \notin B$, then, using, again, the definition of the transitions, $\alpha_{j+1} \notin \mathrm{nextContent}^{-1}(x)$. Therefore, $\langle f, \alpha_{j+1}\rangle \in A$ and, thus, $u \models_{\mathcal{V}_{\mathit{flow}}} I$.

Case $i = j + 1 + n$: In this case, because $\bot \in A$, $\langle s, \alpha_{j+n+1}\rangle \in A$, and $\bot \notin B$ one can immediately see that $u \models_{\mathcal{V}_{\mathit{flow}}} I$. □

Relying on the intuition for these inductive statements for a moment, we know that in the configurations of our RTS a correct letter at position i in some arrangement enforces the correct content of the i-th cell in the next arrangement. But, again, an arrangement is not only the content of the tape but also the head position and the state. In this reduction, we use $\mathcal{B}$ to make sure these aspects of the run are correct. The idea behind the construction of $\mathcal{B}$ is as follows: if there is some letter $x \in Q \times \Gamma$, say at index i, such that $\delta(x) = \langle q, y, m\rangle$, then $\langle q, y\rangle$ is the letter in the next arrangement that encodes the head position. Moreover, the index of this letter is either $i + n - 1$, $i + n$, and $i + n + 1$ in case of $m = \leftarrow$, $m = \downarrow$, and $m = \rightarrow$, respectively.

$\mathcal{B}$ is also used to establish that all arrangements but the last are part of the first section. Moreover, the last arrangement is one with state q_f. That is, $\mathcal{B}$ only recognizes words from the *universe* $\mathcal{U} = (\{f\} \times (\Gamma \times Q \times \Gamma))^* \cdot (\{s\} \times (\Gamma \times \{q_f\} \times \Gamma))^n$. To define the language of $\mathcal{B}$ completely we introduce two notions:

- $\mathit{nextQ} : Q \times \Gamma \to Q$ maps the letter of one arrangement that encodes the head position to the state of the next arrangement: $\mathit{nextQ}(q, x) = p$ if $\delta(q, x) = \langle p, y, m\rangle$.
- $\mathit{nextM} : Q \times \Gamma \to \{n-1, n, n+1\}$ which encodes the distance between two head positions of two arrangements $\alpha \rightarrowtail \beta$ if they are written as one seamless word $\alpha\,\beta$:

$$\mathit{nextM}(q, x) = \begin{cases} n-1 & \text{if } \delta(q, x) = \langle p, y, \leftarrow\rangle \\ n & \text{if } \delta(q, x) = \langle p, y, \downarrow\rangle \\ n+1 & \text{if } \delta(q, x) = \langle p, y, \rightarrow\rangle \end{cases}.$$

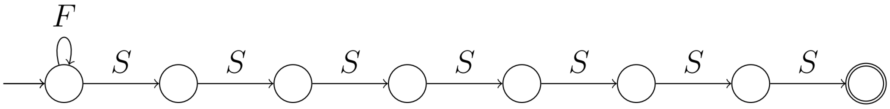
Now, $\mathcal{B}$ is chosen such that it recognizes the language

$$\left\{ \alpha_1 \;\ldots\; \alpha_m \in \mathcal{U} \;\middle|\; \begin{array}{l} \alpha_1 = \langle s, B\rangle \wedge \alpha_2 \in \{f, s\} \times \{q_0\} \times \Gamma \\ \wedge \text{for all } \alpha_i = \langle s, \langle p, x\rangle\rangle : \left(\begin{array}{r} \alpha_j \in \{f, s\} \times \Gamma \text{ for all } i < j < k \\ \text{and } \alpha_k \in \{f, s\} \times \mathit{nextQ}(q, x) \times \Gamma \\ \text{where } k = i + \mathit{nextM}(q, x) \end{array} \right) \end{array} \right\}$$

First, observe that we can recognize $\mathcal{U}$ with a DFA with $n+2$ many states. For the other conditions of the words in the language $\mathcal{B}$, one needs at most $|Q| \cdot n + 3$ states. Again, we rely on a principled example instead of a formal definition to demonstrate this.

Example 2.19: *The language of undesired words for the reduction for $\mathcal{V}_{\mathit{flow}}$.*

Let us demonstrate how to recognize $\mathcal{U}$ with $n+2$ states with our running example; that is, the Turing machine described in Example 2.11 on the input $\#\ 0\ 1\ 0\ \#$. For this, we introduce two short hands: we write F for all words in $\{f\} \times (\Gamma \cup (Q \times \Gamma))$ and S for all words in $\{s\} \times (\Gamma \cup (\{q_f\} \times \Gamma))$. All transitions that are not mentioned lead to a non-accepting sink state. Essentially, this automaton stays in the initial state until the final arrangement starts. For this final arrangement, it counts down the correct length; that is, $n = |x| + 2$ which, in this case, is 7.



A DFA which checks the remaining conditions on the language of $\mathcal{B}$ is presented in Figure 2.13. Conceptionally, this automaton can be separated into three columns – each of which is responsible for one state. In our example the first column ensures the occurrence of q_0 in an appropriate distance, the second column is responsible for $q_\rightarrow$, and the last column for $q_\leftarrow$. Each column can be used to skip up to n many steps. In this way, the appropriate amount of steps can be chosen by an appropriate initial offset. For instance, a head movement to the right executes all n steps and, thus, starts in the first state of the column. If the head does not move one can start in the second state of the column. Naturally, a movement to the left starts in the third state of the column. Again, we assume that all transitions that are not explicitly stated lead to a non-accepting sink.

Figure 2.13: *Automaton for undesired words of reduction for* $\mathcal{V}_{\mathit{flow}}$.

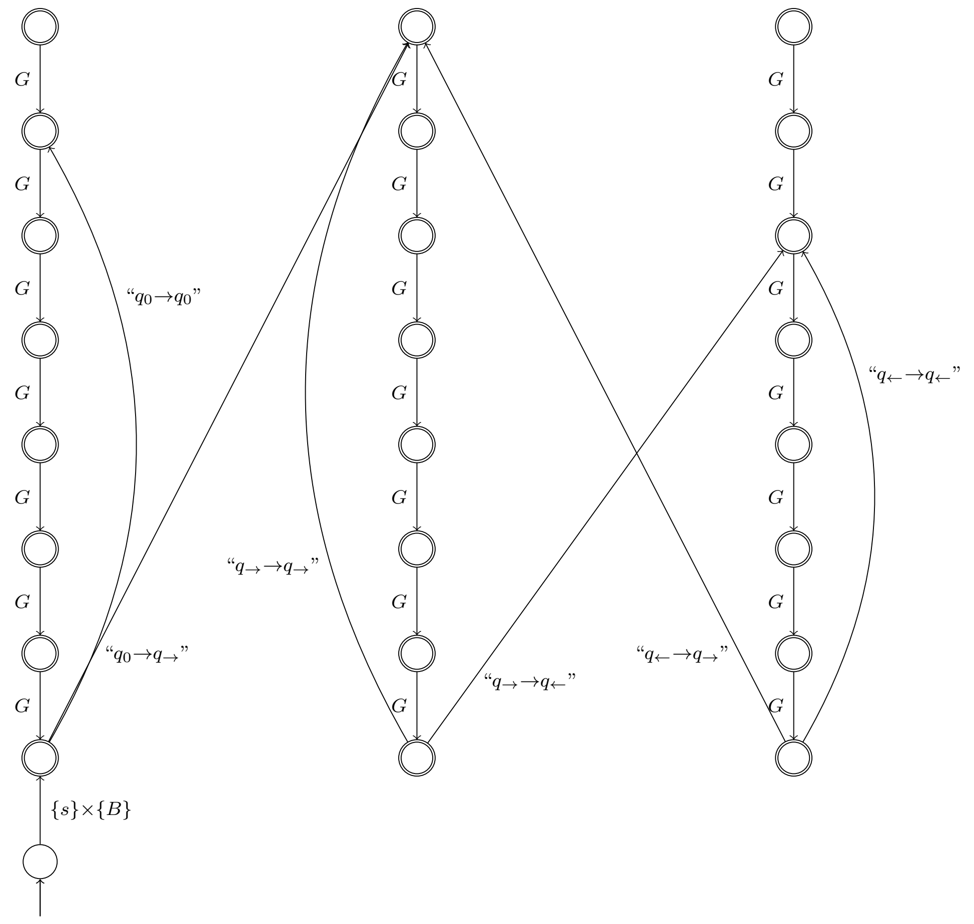
The DFA described in Example 2.19. We introduce the following shorthands:

Letter $G = \{f, s\} \times \Gamma$

State q_0**:** "$q_0 \to q_0$" $= \{f, s\} \times \{q_0\} \times \{0, 1, B\}$ and "$q_0 \to q_\rightarrow$" $= \{f, s\} \times \{q_0\} \times \{\#\}$

State $q_\rightarrow$**:** "$q_\rightarrow \to q_\rightarrow$" $= \{f, s\} \times \{q_\rightarrow\} \times \{0, 1\}$ and "$q_\rightarrow \to q_\leftarrow$" $= \{f, s\} \times \{q_\rightarrow\} \times \{\#, B\}$

State $q_\leftarrow$**:** "$q_\leftarrow \to q_\leftarrow$" $= \{f, s\} \times \{q_\leftarrow\} \times \{0, 1\}$ and "$q_\leftarrow \to q_\rightarrow$" $= \{f, s\} \times \{q_\leftarrow\} \times \{\#, B\}$



It remains to give a final small observation. Namely, we need to maintain that the initial arrangement cannot change after it is initially set. We rely on inductive statements of $\mathcal{V}_{\mathit{flow}}$ to do so.

Lemma 2.13.

First B**:** $\{\langle f, B\rangle, \langle s, B\rangle\}\ \emptyset^* \subseteq \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$

Second Letter: $\emptyset\ \{\langle f, q_0, x_1\rangle, \langle s, q_0, x_1\rangle\}\ \emptyset^* \subseteq \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$

Remaining Input: $\emptyset^i\ \{\langle f, x_i\rangle, \langle s, x_i\rangle\}\ \emptyset^* \subseteq \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$ *for all* $2 \leq i \leq m$

Last B**:** $\emptyset^{m+1}\ \{\langle f, B\rangle, \langle s, B\rangle\}\ \emptyset^* \subseteq \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$

Proof. These are immediate consequences of the fact that the transitions of the RTS can only advance letters from the first section to the second. □

Finally, we can prove the correctness of this reduction.

Lemma 2.14. *$Id(\mathcal{I})\circ \Rightarrow_{\mathcal{V}} \circ Id(\mathcal{B}) \neq \emptyset$ if and only if the Turing machine has a run on x in which an arrangement with state q_f occurs.*

Proof. For this argument assume that every arrangement has a leading and trailing B.

Let the Turing machine have a run

$$\alpha_1^0 \ldots \alpha_n^0 \rightarrowtail \ldots \rightarrowtail \alpha_1^k \ldots \alpha_n^k$$

on x which reaches an arrangement with state q_f. As argued before the constructed RTS faithfully models executions of $\mathcal{M}$. Therefore,

$$\left\langle f, \alpha_1^0\right\rangle \ldots \left\langle f, \alpha_n^0\right\rangle \ldots \left\langle s, \alpha_1^k\right\rangle \ldots \left\langle s, \alpha_n^k\right\rangle$$

is actually reachable from the initial configuration

$$\left\langle f, \alpha_1^0\right\rangle \ldots \left\langle s, \alpha_n^0\right\rangle \perp^{n-1+k\cdot n}.$$

Thus, the pair of this initial configuration and the last configuration is part of the abstraction of $\mathcal{V}_{\mathit{flow}}$. Moreover, the final configuration is accepted by $\mathcal{B}$.

Assume, on the other hand, that $\Rightarrow_{\mathcal{V}_{\mathit{flow}}}$ contains a pair of

$$\left\langle s, \alpha_1^0 \right\rangle \ \ldots \ \left\langle s, \alpha_n^0 \right\rangle \ \bot^{n-1+k\cdot n}$$

and

$$\left\langle f, \alpha_1^0 \right\rangle \ \ldots \ \left\langle f, \alpha_n^0 \right\rangle \ \ldots \ \left\langle s, \alpha_1^k \right\rangle \ \ldots \ \left\langle s, \alpha_n^k \right\rangle .$$

From this final configuration, we extract the sequence

$$\alpha_1^0 \ \ldots \ \alpha_n^0, \ldots, \alpha_1^k \ \ldots \ \alpha_n^k.$$

With Lemma 2.13 it is straightforward to argue that $\alpha_1^0 \ \ldots \ \alpha_n^0$ is the initial configuration of $\mathcal{M}$ on x by the choice of $\mathcal{I}$. Intuitively, the initial arrangement is enforced in the initial configuration and must never change again. Assume that this sequence, up to some arrangements $\alpha_1 \ \ldots \ \alpha_n$, is a prefix of the run of $\mathcal{M}$ on x. Consider $\beta_1 \ \ldots \ \beta_n$ which follows $\alpha_1 \ \ldots \ \alpha_n$ in the sequence. Using the construction of $\mathcal{B}$ one can immediately verify that the head position and movement from $\alpha_1 \ \ldots \ \alpha_n$ to $\beta_1 \ \ldots \ \beta_n$ is consistent with δ. Pick now one position i. The letter of the final configuration that corresponds to α_i is $\langle f, \alpha_i \rangle$, say at index j. Let $y \in \Gamma$ be the content of the i-th cell in the arrangement that follows $\alpha_1 \ \ldots \ \alpha_n$ which is identified by α_i. Lemma 2.12 gives an inductive statement for $\mathcal{V}_{\mathit{flow}}$ and y where the index j of the statement is the letter A while the index $j+n$ is the letter B. Note that the index $j+n$ corresponds to the letter $\langle z, \beta_i \rangle$ for some $z \in \{f, s\}$. We argue $\beta_i \in \{y\} \cup Q \times \{y\}$ because otherwise the inductive statement is not satisfied. The reason for this is that

$$\begin{aligned}
&\langle f, \alpha_i \rangle \notin \{\bot\} , \\
&\langle f, \alpha_i \rangle \notin \{s\} \times (\Gamma \cup Q \times \Gamma) , \text{ and} \\
&\langle f, \alpha_i \rangle \notin \{f\} \times \big((\Gamma \cup Q \times \Gamma) \setminus \mathrm{nextContent}^{-1}(y) \big) .
\end{aligned}$$

Since the statement is satisfied in the initial configuration (either because the letter at index j is $\bot$ or in $\{s\} \times (\Gamma \cup Q \times \Gamma)$) we have indeed $\beta_i \in \{y\} \cup Q \times \{y\}$. This means, by arbitrary choice of i, that the tape content of $\beta_1 \ \ldots \ \beta_n$ is consistent with the arrangement that follows $\alpha_1 \ \ldots \ \alpha_n$. Therefore, $\alpha_1 \ \ldots \ \alpha_n \rightarrowtail \beta_1 \ \ldots \ \beta_n$. Using this argument inductively, one can establish that the word accepted by $\mathcal{B}$ encodes the actual run of the Turing machine on x which reaches q_f. ☐

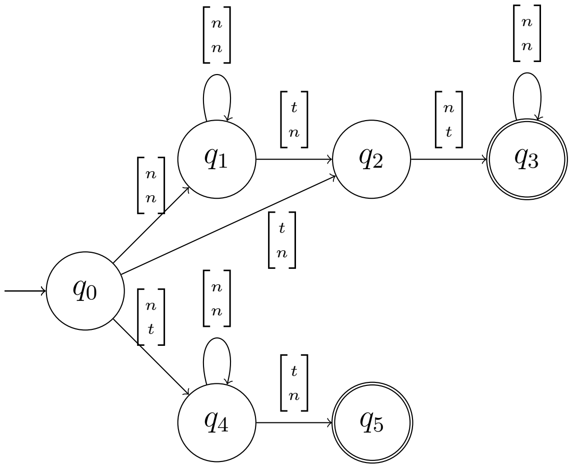
Theorem 2.5. *Problem 2.2 is* PSPACE*-hard for* $\mathcal{V}_{\mathit{flow}}$.

2.6 Trap in PSpace

This section is based on [ERW22b]. We fix $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$. We also introduce a running example on which we illustrate the concepts introduced in this section.

Example 2.20: *Circular token passing.*

We consider a modified version of Example 2.2. In this version, we introduce a transition which allows the token to move from the very last to the very first position. The initial language still is $t\ n^*$ and the language of transitions becomes $\left(\begin{bmatrix}n\\n\end{bmatrix}^* \begin{bmatrix}t\\n\end{bmatrix} \begin{bmatrix}n\\t\end{bmatrix} \begin{bmatrix}n\\n\end{bmatrix}^*\right) \mid \left(\begin{bmatrix}n\\t\end{bmatrix} \begin{bmatrix}n\\n\end{bmatrix}^* \begin{bmatrix}t\\n\end{bmatrix}\right)$. We introduce the transducer of this language now:



Without going into detail, we want to note that, in this example, $Id(\mathcal{I})\circ \Rightarrow_{\mathcal{V}_{\mathit{trap}}}$ coincides with $Id(\mathcal{I})\circ \rightsquigarrow^*_{\mathcal{T}}$ because of the following language of traps:

At least one token: $\{t\}^* \subseteq \text{Inductive}_{\mathcal{V}_{\mathit{trap}}}(\mathcal{R})$

At most one token: $\emptyset^*\ \{n\}\ \emptyset^*\ \{n\}\ \emptyset^* \subseteq \text{Inductive}_{\mathcal{V}_{\mathit{trap}}}(\mathcal{R})$

Preliminaries

Let us introduce a few notations for statements for $\mathcal{V}_{\mathit{trap}}$:

Union For two statements $I_1 \; \ldots \; I_n$ and $T_1 \; \ldots \; T_n$ we refer to $(I_1 \cup T_1) \; \ldots \; (I_n \cup T_n)$ as $I_1 \; \ldots \; I_n \sqcup T_1 \; \ldots \; T_n$.

Subset For two statements $I_1 \; \ldots \; I_n$ and $T_1 \; \ldots \; T_n$ such that $I_i \subseteq T_i$ for all $1 \leq p \leq n$ we write $I_1 \; \ldots \; I_n \sqsubseteq T_1 \; \ldots \; T_n$.

Strict subset For two statements $I_1 \; \ldots \; I_n$ and $T_1 \; \ldots \; T_n$ such that $I_i \subseteq T_i$ for all $1 \leq i \leq n$ and there exists $1 \leq j \leq n$ such that $I_j \subsetneq T_j$ we write $I_1 \; \ldots \; I_n \sqsubset T_1 \; \ldots \; T_n$.

Recall from before, that $u_1 \; \ldots \; u_n \models_{\mathcal{V}_{trap}} I_1 \; \ldots \; I_n$ if and only if there is $1 \leq i \leq n$ such that $u_i \in I_i$.

Traps in PSpace

The rough outline of this section is as follows:

- We introduce the concept of a *separator* for two configurations v and u. The separator S is one statement from $\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$, that is uniquely defined by u, such that $v \Rrightarrow_{\mathcal{V}_{trap}} u$ if and only if $v \not\models_{\mathcal{V}_{trap}} S$.
- We present how one can compute a *separator* for some configuration u. The relation $\Rrightarrow_{\mathcal{V}_{trap}}$ can then be decided by computing the separator S for u and checking whether v does not satisfy S.
- Finally, we show how one can construct a non-deterministic Σ-Σ-transducer which captures $\Rrightarrow_{\mathcal{V}_{trap}}$ by guessing S on the fly and verifying that it is the separator for u and that v does not satisfy it. The states of this transducer are all permutations of the states of $\mathcal{T}$. The argument concludes by observing that checking emptiness of the transducer $Id(\mathcal{I}) \circ \Rrightarrow_{\mathcal{V}_{trap}} \circ Id(\mathcal{B})$ can then be achieved in polynomial space.

Lemma 2.15. *If* $I_1 \; \ldots \; I_n, T_1 \; \ldots \; T_n \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$, *then* $I_1 \; \ldots \; I_n \sqcup T_1 \; \ldots \; T_n \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$.

Proof. Pick any $v_1 \; \ldots \; v_n \rightsquigarrow_{\mathcal{T}} u_1 \; \ldots \; u_n$ such that $v_1 \; \ldots \; v_n \models_{\mathcal{V}_{trap}} I_1 \; \ldots \; I_n \sqcup T_1 \; \ldots \; T_n$. Then there is $1 \leq i \leq n$ such that $v_i \in I_i \cup T_i$ and, therefore, (without loss of generality) $v_i \in I_i$. Thus, $v_1 \; \ldots \; v_n \models_{\mathcal{V}_{trap}} I_1 \; \ldots \; I_n$. Pick $1 \leq j \leq n$ such that $u_j \in I_j$ which exists because $I_1 \; \ldots \; I_n \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$. Consequently, $u_1 \; \ldots \; u_n \models_{\mathcal{V}_{trap}} I_1 \; \ldots \; I_n \sqcup T_1 \; \ldots \; T_n$ because $u_j \in I_j \cup T_j$. □

This means that $\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R}) \cap \left(2^{\Sigma}\right)^n$ is closed under the operation $\sqcup$ for all n. Moreover, $w \not\models_{\mathcal{V}_{trap}} P$ and $w \not\models_{\mathcal{V}_{trap}} Q$ implies $w \not\models_{\mathcal{V}_{trap}} P \sqcup Q$.

Corollary 2.1. *For every configuration $v \in \Sigma^n$ exists a unique maximal (w. r. t. $\sqsubseteq$) $S \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R}) \cap \left(2^{\Sigma}\right)^n$ such that $v \not\models_{\mathcal{V}_{trap}} S$.*

Proof. The set $\mathcal{Q} = \left\{Q \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R}) \cap \left(2^{\Sigma}\right)^n \mid w \not\models Q\right\}$ is finite but not empty because $\emptyset^n \in \mathcal{Q}$. Since $\mathcal{Q}$ is closed under $\sqcup$ and $Q \sqsubseteq Q \sqcup P$ for all $Q, P \in \left(2^{\Sigma}\right)^n$, $S = \sqcup\mathcal{Q}$ has the desired properties. ☐

In the following, we call the unique S of Corollary 2.1 the *separator* of the configuration v. This name is motivated by the following observation:

Lemma 2.16. *$v \Rightarrow_{\mathcal{V}_{trap}} u$ if and only if $v \not\models_{\mathcal{V}_{trap}} S$ where S is the separator of u.*

Proof. Observe that $v \models_{\mathcal{V}_{trap}} Q$ implies $v \models_{\mathcal{V}_{trap}} Q \sqcup Q'$ for all Q'. Thus, if $v \not\Rightarrow_{\mathcal{V}_{trap}} u$, then there exists some $Q \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$ such that $u \not\models_{\mathcal{V}_{trap}} Q$ and $v \models_{\mathcal{V}_{trap}} Q$. That means $Q \sqsubseteq S$ and, therefore, $v \models_{\mathcal{V}_{trap}} S$.

On the other hand, assume $v \Rightarrow_{\mathcal{V}_{trap}} u$. Thus, for all $Q \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$ with $v \models_{\mathcal{V}_{trap}} Q$ also $u \models_{\mathcal{V}_{trap}} Q$. $v \not\models_{\mathcal{V}_{trap}} S$ follows by contraposition since $u \not\models_{\mathcal{V}_{trap}} S$ by the definition of separator. ☐

We proceed now by presenting a process to compute the separator of any given word $w = w_1 \ \dots \ w_n$. The idea of this process is as follows: initially, we consider the largest (w. r. t. $\sqsubseteq$) statement for which w is not a model. Then, we gradually refine this statement until it becomes inductive. By the nature of this refinement process, we can show that it ends in the separator of w. The refinement process works as follows: let $P_1 \ \dots \ P_n$ be the current statement. Find some transition $\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \ \dots \ \begin{bmatrix} u_n \\ v_n \end{bmatrix}$ with $u_1 \ \dots \ u \models_{\mathcal{V}_{trap}} P_1 \ \dots \ P_n$ while $v_1 \ \dots \ v_n \not\models_{\mathcal{V}_{trap}} P_1 \ \dots \ P_n$. Refine the statement to $P_1 \setminus \{u_1\} \ \dots \ P_n \setminus \{u_n\}$. Let us illustrate this process by an example first, and, afterwards, we formalize it.

Example 2.21: *Computing a separator.*

Consider the (reachable) configuration $n \ t \ n \ n \ n \ n$ for our running example. The largest statement that is not satisfied by this configuration is $\{t\} \ \{n\} \ \{t\} \ \{t\} \ \{t\} \ \{t\}$ since it contains at every position all letters but the one that is at the same position in the original configuration. We now show how this statement is refined to become inductive. For this, we show below a series of statements and transitions such that the transition refines the previous statement to the next. In the following table we mark statements with • and the refining transitions with ▷.

$$\begin{array}{ccccccc}
\bullet & \{t\} & \{n\} & \{t\} & \{t\} & \{t\} & \{t\} \\
\triangleright & \begin{bmatrix} t \\ n \end{bmatrix} & \begin{bmatrix} n \\ t \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} \\
\bullet & \emptyset & \emptyset & \{t\} & \{t\} & \{t\} & \{t\} \\
\triangleright & \begin{bmatrix} n \\ t \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} t \\ n \end{bmatrix} \\
\bullet & \emptyset & \emptyset & \{t\} & \{t\} & \{t\} & \emptyset \\
\triangleright & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} t \\ n \end{bmatrix} & \begin{bmatrix} n \\ t \end{bmatrix} \\
\bullet & \emptyset & \emptyset & \{t\} & \{t\} & \emptyset & \emptyset \\
\triangleright & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} t \\ n \end{bmatrix} & \begin{bmatrix} n \\ t \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} \\
\bullet & \emptyset & \emptyset & \{t\} & \emptyset & \emptyset & \emptyset \\
\triangleright & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} t \\ n \end{bmatrix} & \begin{bmatrix} n \\ t \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} & \begin{bmatrix} n \\ n \end{bmatrix} \\
\bullet & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset
\end{array}$$

The refinement process ends here. As we show now this means $\emptyset\ \emptyset\ \emptyset\ \emptyset\ \emptyset\ \emptyset$ is indeed the separator for the configuration $n\ t\ n\ n\ n\ n$.

Definition 2.14: *Separator sequence.*

For any $w_1 \ \ldots \ w_n \in \Sigma^*$ we call a sequence of statements $\langle S_1^0 \ \ldots \ S_n^0, \ldots, S_1^k \ \ldots \ S_n^k \rangle$ a *separator sequence* of w if

- $S_i^0 = \Sigma \setminus \{w_i\}$, for all $1 \leq i \leq n$,
- for every $0 \leq i < k$ exists $v_1 \ \ldots \ v_n \rightsquigarrow_{\mathcal{T}} u_1 \ \ldots \ u_n$ such that $u_1 \ \ldots \ u_n \not\models_{\mathcal{V}_{trap}} P_i$, and
- $S_j^{i+1} = S_j^i \setminus \{v_j\}$.

We prove that every separator sequence converges to the same statement; namely, the separator of w.

Lemma 2.17. *Let $S_1 \ \ldots \ S_n$ be the separator of w and $\langle S_1^0 \ \ldots \ S_n^0, \ldots, S_1^k \ \ldots \ S_n^k \rangle$ a separator sequence for w. Then,*

1. $S_1^i \ \dots \ S_n^i \sqsupseteq S_1^{i+1} \ \dots \ S_n^{i+1}$ *for all* $0 \leq i < k$*, and*
2. $S_1^i \ \dots \ S_n^i \sqsupseteq S_1 \ \dots \ S_n$ *for all* $0 \leq i \leq k$*.*

Proof. Prove the properties by induction. The first property is immediate from the definition of separator sequences.

It remains to prove the second property. First, observe that $S_1 \ \dots \ S_n \sqsubseteq S_1^0 \ \dots \ S_n^0$ since $S_1^0 \ \dots \ S_n^0$ is chosen to include all statements for which w is not a model. Let $v_1 \ \dots \ v_n \rightsquigarrow_{\mathcal{T}} u_1 \ \dots \ u_n$ with $u_1 \ \dots \ u_n \not\models_{\mathcal{V}_{trap}} S_1^i \ \dots \ S_n^i$ and $S_j^{i+1} = S_j^i \setminus \{v_j\}$ for all $1 \leq j \leq n$. By induction hypothesis $S_1^i \ \dots \ S_n^i \sqsupseteq S_1 \ \dots \ S_n$. Since $u_1 \ \dots \ u_n \not\models_{\mathcal{V}_{trap}} S_1^i \ \dots \ S_n^i$ also $u_1 \ \dots \ u_n \not\models_{\mathcal{V}_{trap}} S_1 \ \dots \ S_n$. Consequently, $v_1 \ \dots \ v_n \not\models_{\mathcal{V}_{trap}} S_1 \ \dots \ S_n$ either, because $S_1 \ \dots \ S_n \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$. Therefore, $v_j \notin S_j$ for all $1 \leq j \leq n$. Hence, $S_1 \ \dots \ S_n \sqsubseteq S_1^{i+1} \ \dots \ S_n^{i+1}$. □

We can draw two interesting consequences from this observation:

Corollary 2.2. *Let* $\langle S_0, \dots, S_k \rangle$ *be a separator sequence for some word* w*. If* $S_k \in \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$*, then* S_k *is the separator of* w*.*

Proof. This is an immediate consequence of the fact that, otherwise, S_k is a counterexample to the maximality of the separator of w by Lemma 2.17. □

Corollary 2.3. *Let* $\langle S_0, \dots, S_k \rangle$ *be a separator sequence for some word* w*. If there is some* $v \in \Sigma^*$ *with* $u \not\models_{\mathcal{V}_{trap}} S_i$*, then* $v \not\models_{\mathcal{V}_{trap}} S_j$ *for all* $j > i$*.*

Proof. We know that a separator sequence is decreasing w. r. t. $\sqsubseteq$. Also, $v_1 \ \dots \ v_n \not\models_{\mathcal{V}_{trap}} S_1 \ \dots \ S_n$ is equivalent to $v_m \notin S_m$ for all $1 \leq m \leq n$. This implies, however, that $v_m \notin Q_m$ for all $1 \leq m \leq n$ if $Q_m \subseteq S_m$. Thus, $v_1 \ \dots \ v_n \not\models_{\mathcal{V}_{trap}} Q_1 \ \dots \ Q_m$ for all $Q_1 \ \dots \ Q_m \sqsubseteq S_1 \ \dots \ S_m$. □

These observations, in combination with Lemma 2.16, imply another interesting property of separator sequences:

Corollary 2.4. $v \Rightarrow_{\mathcal{V}_{trap}} u$ *if and only if there exists some separator sequence* $\langle S_0, \dots, S_k \rangle$ *for* u *such that* $v \not\models_{\mathcal{V}_{trap}} S_i$ *for some* $0 \leq i \leq k$*.*

Thus, we know that we can prove $v \Rightarrow_{\mathcal{V}_{trap}} u$ by computing some separator sequence for u and checking whether v does not satisfy some element of that sequence. In every separator sequence, every step is “justified” via some transition. Therefore, computing a separator sequence relies on finding transitions of $\mathcal{R}$. The set of all transitions is encoded by $\mathcal{T}$. That means, the transitions that "justify" some step in the separator sequence exist because there are excepting runs for them in $\mathcal{T}$.

Example 2.22: *A tableau for Example 2.21.*

Note that the transitions that we used for obtaining a separator sequence in Example 2.21 were

- $\begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix}$
- $\begin{bmatrix} n \\ t \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} t \\ n \end{bmatrix}$
- $\begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix}$
- $\begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix}$
- $\begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix} \begin{bmatrix} n \\ n \end{bmatrix}$

We know that these are transitions of the RTS because for every transition there is an accepting run of the transducer that encodes the transitions. These runs are

- $q_0 \quad q_2 \quad q_3 \quad q_3 \quad q_3 \quad q_3 \quad q_3$
- $q_0 \quad q_4 \quad q_4 \quad q_4 \quad q_4 \quad q_4 \quad q_5$
- $q_0 \quad q_1 \quad q_1 \quad q_1 \quad q_1 \quad q_2 \quad q_3$
- $q_0 \quad q_1 \quad q_1 \quad q_1 \quad q_2 \quad q_3 \quad q_3$
- $q_0 \quad q_1 \quad q_1 \quad q_2 \quad q_3 \quad q_3 \quad q_3$

We introduce a notation to represent the runs of refining transitions for a separator sequence. Essentially, this is a matrix of states of $\mathcal{T}$ which we call a *tableau*:

$$\begin{bmatrix} q_0 & q_2 & q_3 & q_3 & q_3 & q_3 & q_3 \\ q_0 & q_4 & q_4 & q_4 & q_4 & q_4 & q_5 \\ q_0 & q_1 & q_1 & q_1 & q_1 & q_2 & q_3 \\ q_0 & q_1 & q_1 & q_1 & q_2 & q_3 & q_3 \\ q_0 & q_1 & q_1 & q_2 & q_3 & q_3 & q_3 \end{bmatrix}$$

Additionally, this tableau can be used as a witness that there is a separator sequence for the configuration $n\ t\ n\ n\ n\ n$ which ends in $\emptyset\ \emptyset\ \emptyset\ \emptyset\ \emptyset\ \emptyset$. We say this tableau *computes* $\emptyset\ \emptyset\ \emptyset\ \emptyset\ \emptyset\ \emptyset$ for $n\ t\ n\ n\ n\ n$.

Definition 2.15: *Tableau.*

Let $Q_{\mathcal{T}}$ denote the states of $\mathcal{T}$. We call matrices $M = [q_{i,j}]_{0 \leq i \leq n, 1 \leq j \leq k} \in Q_{\mathcal{T}}^{k \times n+1}$ *tableaux* of $\mathcal{T}$. Let $v_1 \; \ldots \; v_n \in \Sigma^n$ be some configuration and $S_1^k \; \ldots \; S_n^k \in \left(2^\Sigma\right)^n$. We say M computes $S_1^k \; \ldots \; S_n^k \in \left(2^\Sigma\right)^n$ for $v_1 \; \ldots \; v_n$ if:

- there is a separator sequence $\left\langle S_1^0 \; \ldots \; S_n^0, \ldots, S_1^k \; \ldots \; S_n^k \right\rangle$ for $v_1 \; \ldots \; v_n$ such that
- there are transitions $u_1^1 \; \ldots \; u_n^1 \rightsquigarrow_{\mathcal{T}} w_1^1 \; \ldots \; w_n^1, \ldots, u_1^k \; \ldots \; u_n^k \rightsquigarrow_{\mathcal{T}} w_1^k \; \ldots \; w_n^k$ with
  - $q_{0,i} \; \ldots \; q_{n,i}$ is an accepting run for $u_1^i \; \ldots \; u_n^i \rightsquigarrow_{\mathcal{T}} w_1^i \; \ldots \; w_n^i$ for all $1 \leq i \leq k$,
  - $w_1^i \; \ldots \; w_n^i \not\models_{\mathcal{V}_{trap}} S_0^{i-1} \; \ldots \; S_n^{i-1}$ for all $1 \leq i \leq k$, and
  - $S_j^i \setminus \left\{u_j^i\right\} = S_j^{i+1}$ for all $0 \leq i < k$ and $1 \leq j \leq n$.

In other words, a tableau M computes some statement S for some configuration v if the lines of M are accepting runs for transitions that induce a separator sequence for v which ends in S. Therefore, it is natural to read tableaux line by line to follow along the separator sequence. However, tableaux can also be read “vertically”; that is, *column* by *column*.

Example 2.23: *Columns in a tableau.*

Consider the tableau from Example 2.22 again, but focus only on the last two columns:

$$\begin{bmatrix} q_3 & q_3 \\ q_4 & q_5 \\ q_2 & q_3 \\ q_3 & q_3 \\ q_3 & q_3 \end{bmatrix}$$

Every line of these two columns is the last step of accepting runs in $\mathcal{T}$ for transitions. Moreover, these transitions compute, for a word that ends in n, a statement that ends in $\emptyset$. The refining process of the separator sequence for the last position is captured by these columns:

- $\{t\}$ is refined by $\left\langle q_3, \begin{bmatrix} n \\ n \end{bmatrix}, q_3 \right\rangle$ to $\{t\}$.
- $\{t\}$ is refined by $\left\langle q_4, \begin{bmatrix} t \\ n \end{bmatrix}, q_5 \right\rangle$ to $\emptyset$.
- $\emptyset$ is refined by $\left\langle q_2, \begin{bmatrix} n \\ t \end{bmatrix}, q_3 \right\rangle$ to $\emptyset$.
- $\emptyset$ is refined by $\left\langle q_3, \begin{bmatrix} n \\ n \end{bmatrix}, q_3 \right\rangle$ to $\emptyset$.
- $\emptyset$ is refined by $\left\langle q_3, \begin{bmatrix} n \\ n \end{bmatrix}, q_3 \right\rangle$ to $\emptyset$.

We want to draw attention to two things in particular. First, the first step $\left\langle q_3, \begin{bmatrix} n \\ n \end{bmatrix}, q_3 \right\rangle$ does not have t as target of the letter. This is important because otherwise the target of the transition already satisfies the statement which renders the transition inappropriate for the separator sequence. Second, the same holds for the second step $\left\langle q_4, \begin{bmatrix} t \\ n \end{bmatrix}, q_5 \right\rangle$ but this step also shows that t is removed from this letter of the statement.

More generally, we see that adjacent columns of a tableau present a similar refinement process as the complete tableau but only use one single step of $\mathcal{T}$ instead of a complete run. For this reason, we transfer notions from tableaux to columns. For example, we say that these columns *compute* $\emptyset$ for n.

Definition 2.16: *Columns.*

Let $Q_{\mathcal{T}}$ denote the states of $\mathcal{T}$, and let $Q_0 \subseteq Q_{\mathcal{T}}$ and $F \subseteq Q_{\mathcal{T}}$ the initial and accepting states, respectively. We call words from $Q_{\mathcal{T}}^*$ *columns*. Additionally, we call a column c:

initial if $c \in Q_0^*$, and

accepting if $c \in F^*$.

We say two columns $c_1 \ \ldots \ c_k$ and $d_1 \ \ldots \ d_k$ compute $S \in 2^{\Sigma}$ for $v \in \Sigma$ if there are $\left\langle c_1, \begin{bmatrix} u_1 \\ w_1 \end{bmatrix}, d_1 \right\rangle, \ldots, \left\langle c_k, \begin{bmatrix} u_k \\ w_k \end{bmatrix}, d_k \right\rangle \in \Delta_{\mathcal{T}}$ such that

- $w_1 = v$,

- $w_{i+1} \in \{u_1, \dots, u_i\}$ for all $1 \leq i < k$, and
- $S = \Sigma \setminus \{u_1, \dots, u_k\}$.

In this case, we also call $u_1, \dots, u_k$ the *removal sequence*.

From the definition of columns, one can see that they are just a different perspective on a tableau:

Lemma 2.18. *There is a tableau M that computes $S_1 \dots S_n$ for $v_1 \dots v_n$ if and only if there are columns $c_0, \dots, c_n \in Q_T^k$ such that c_0 is initial, c_n is accepting, and c_{i-1} and c_i compute S_i for v_i for all $1 \leq i \leq n$.*

Proof. Observe that the columns of a tableau have the desired properties. Show that the given sequence of columns forms a tableau. For this, fix, for every $0 \leq i < n$, the removal sequence as $u_1^i, \dots, u_k^i$. By induction to j construct a matrix of the first j elements of $C_0, \dots, C_n$. This gives a tableau for w that computes $\Sigma \setminus \{u_1^1, \dots, u_j^1\} \dots \Sigma \setminus \{u_1^n, \dots, u_j^n\}$. The statement of the lemma follows once j reaches k. ☐

Although the following observation might seem trivial, it is important later on.

Example 2.24: *Expanding columns.*

In Example 2.23, we considered the columns

$$\begin{bmatrix} q_3 & q_3 \\ q_4 & q_5 \\ q_2 & q_3 \\ q_3 & q_3 \\ q_3 & q_3 \end{bmatrix}$$

The last two lines are the same. In fact, one can add the same lines over and over again after their first occurrence without changing the value the columns compute. Intuitively, one can simply repeat a previous step without changing the letter of the statement these columns compute. This is because, since the step was already used before, it is possible to be used and it does not add any new letter to the removal sequence. For instance, the following columns all compute $\emptyset$ for n:

$$\begin{bmatrix} q_3 & q_3 \\ q_4 & q_5 \\ q_2 & q_3 \\ q_3 & q_3 \\ q_3 & q_3 \end{bmatrix}, \begin{bmatrix} q_3 & q_3 \\ q_4 & q_5 \\ q_2 & q_3 \\ q_3 & q_3 \\ q_4 & q_5 \\ q_3 & q_3 \end{bmatrix}, \begin{bmatrix} q_3 & q_3 \\ q_3 & q_3 \\ q_4 & q_5 \\ q_2 & q_3 \\ q_3 & q_3 \\ q_3 & q_3 \end{bmatrix}, \begin{bmatrix} q_3 & q_3 \\ q_4 & q_5 \\ q_2 & q_3 \\ q_3 & q_3 \\ q_3 & q_3 \\ q_2 & q_3 \end{bmatrix}, \begin{bmatrix} q_3 & q_3 \\ q_4 & q_5 \\ q_2 & q_3 \\ q_3 & q_3 \\ q_3 & q_3 \\ q_4 & q_5 \\ q_3 & q_3 \end{bmatrix}.$$

Lemma 2.19. *Let $c_1 \ \dots \ c_k$ and $d_1 \ \dots \ d_k$ be two columns that compute S for v. For any $1 \leq i \leq k$ and $i \leq j \leq k$, the columns $c_1 \ \dots \ c_j \ c_i \ c_{j+1} \ \dots \ c_k$ and $d_1 \ \dots \ d_j \ d_i \ d_{j+1} \ \dots \ d_k$ compute S for v.*

Proof. In the names of Definition 2.16, the original columns use the step $\left\langle c_i, \begin{bmatrix} u_i \\ w_i \end{bmatrix}, d_i \right\rangle$ from $\mathcal{T}$. Thus, $w_i \in \{u_1, \dots, u_{i-1}\}$. Therefore, one can just repeat this step in the modified columns. If the removal sequence was $u_1, \dots, u_k$ from $c_1 \ \dots \ c_k$ to $d_1 \ \dots \ d_k$, then it is $u_1, \dots, u_j, u_i, u_{j+1}, \dots, u_k$ from $c_1 \ \dots \ c_j \ c_i \ c_{j+1} \ \dots \ c_k$ to $d_1 \ \dots \ d_j \ d_i \ d_{j+1} \ \dots \ d_k$. Regardless, both pairs of columns compute S. ☐

Repeating rows at later points again changes the individual columns by inserting states that occurred at some point at a later point again. We introduce an order on columns $c \preceq c'$ if c' can be obtained from c by any number of these repetitions.

Definition 2.17: *Column order.*

Let $c_1 \ \dots \ c_n$ be a column. For all $1 \leq i \leq j \leq n$ we write $c_1 \ \dots \ c_n \prec c_1 \ \dots \ c_{i-1} \ c_i \ c_{i+1} \ \dots \ c_j \ c_i \ c_{j+1} \ \dots \ c_n$. We denote with $\preceq$ the reflexive transitive closure of $\prec$.

Example 2.25: *An order on columns.*

$$\begin{bmatrix} q_3 \\ q_4 \\ q_2 \end{bmatrix} \preceq \begin{bmatrix} q_3 \\ q_4 \\ q_2 \end{bmatrix} \preceq \begin{bmatrix} q_3 \\ q_4 \\ q_2 \\ q_3 \end{bmatrix} \preceq \begin{bmatrix} q_3 \\ q_4 \\ q_2 \\ q_4 \\ q_2 \\ q_4 \\ q_3 \end{bmatrix} \not\preceq \begin{bmatrix} q_2 \\ q_3 \\ q_2 \\ q_4 \\ q_4 \\ q_2 \\ q_4 \\ q_3 \end{bmatrix}$$

$\preceq$ forms a partial order on $Q_{\mathcal{T}}^*$. The minimal elements of this order are columns in which each letter occurs at most once. For every column c, there exists a unique minimal element $\mathrm{reduce}(c)$ in $\{c' \mid c' \preceq c\}$. Essentially, one obtains $\mathrm{reduce}(c)$ by removing every letter from c but its first occurrence. Therefore, every column c in which every letter occurs at most once satisfies $\mathrm{reduce}(c) = c$. We call these columns *base columns* and denote the set of all base columns as $\mathit{bases}(\mathcal{T}) = \{\mathrm{reduce}(c) \colon c \in Q_{\mathcal{T}}^*\}$.

Example 2.26: *Base columns.*

$$\mathrm{reduce}\left(\begin{bmatrix} q_3 \\ q_4 \\ q_2 \end{bmatrix}\right) = \begin{bmatrix} q_3 \\ q_4 \\ q_2 \end{bmatrix} \text{ and } \mathrm{reduce}\left(\begin{bmatrix} q_2 \\ q_3 \\ q_2 \\ q_4 \\ q_4 \\ q_2 \\ q_4 \\ q_3 \end{bmatrix}\right) = \begin{bmatrix} q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

We introduce a technical observation. Specifically, we show that any two columns c and d that have the same base ancestor b can be merged into a common child.

Lemma 2.20. *There exists, for every columns c and d such that* $\mathrm{reduce}(c) = \mathrm{reduce}(d)$, *a column e with $c \preceq e$ and $d \preceq e$.*

Proof. Fix $c = c_1 \ \dots \ c_n$ and $d = d_1 \ \dots \ d_m$. Now, we inductively construct an (increasing) sequence of pairs $\langle \ell, r \rangle$ such that there exists, for every pair, a column e' with

- $c_1 \ \dots \ c_\ell \preceq e'$ and $d_1 \ \dots \ d_r \preceq e'$, and
- $\text{reduce}(c_1 \ \dots \ c_\ell) = \text{reduce}(d_1 \ \dots \ d_r) = \text{reduce}(e')$.

Initially, fix ε for $\langle 0, 0 \rangle$ and observe $\text{reduce}(\varepsilon) = \varepsilon \preceq \varepsilon$.

On the other hand, for any $\langle \ell, r \rangle$ and the corresponding column $e'_1 \ \dots \ e'_k$, distinguish three cases:

There is $i < \ell + 1$ such that $c_i = c_{\ell+1}$: Because $\text{reduce}(c_1 \ \dots \ c_\ell) = \text{reduce}(e'_1 \ \dots \ e'_k)$ there is some $j \leq r$ such that $e_j = c_i$. Consequently,

- $c_1 \ \dots \ c_{\ell+1} \preceq e'_1 \ \dots \ e'_k \ c_{\ell+1}$ since $c_1 \ \dots \ c_\ell \preceq e'_1 \ \dots \ e'_k$,
- $d_1 \ \dots \ d_r \preceq e'_1 \ \dots \ e'_k \ c_{\ell+1}$ since $d_1 \ \dots \ d_r \preceq e'_1 \ \dots \ e'_k \prec e'_1 \ \dots \ e'_k \ c_{\ell+1}$, and
- $\text{reduce}(c_1 \ \dots \ c_{\ell+1}) = \text{reduce}(e'_1 \ \dots \ e'_k \ c_{\ell+1}) = \text{reduce}(d_1 \ \dots \ d_r)$.

Thus, construct the pair $\langle \ell + 1, r \rangle$ with $e'_1 \ \dots \ e'_k \ c_{\ell+1}$.

There is $i < r + 1$ such that $d_i = d_{r+1}$: With analog reasoning as in the case before construct $\langle \ell, r + 1 \rangle$ with $e'_1 \ \dots \ e'_k \ d_{r+1}$.

Otherwise: $c_{\ell+1} = d_{r+1} = e'_{k+1}$ because $\text{reduce}(c_1 \ \dots \ c_n) = \text{reduce}(d_1 \ \dots \ d_m)$ and $\text{reduce}(c_1 \ \dots \ c_\ell) = \text{reduce}(d_1 \ \dots \ d_r) = \text{reduce}(e')$. Thus,

- $c_1 \ \dots \ c_{\ell+1} \preceq e'_1 \ \dots \ e'_k \ e'_{k+1}$ since $c_1 \ \dots \ c_\ell \preceq e'_1 \ \dots \ e'_k$,
- $d_1 \ \dots \ d_{r+1} \preceq e'_1 \ \dots \ e'_k \ e'_{k+1}$ since $d_1 \ \dots \ d_r \preceq e'_1 \ \dots \ e'_k$, and
- $\text{reduce}(c_1 \ \dots \ c_{\ell+1}) = \text{reduce}(e'_1 \ \dots \ e'_k \ e'_{k+1}) = \text{reduce}(d_1 \ \dots \ d_r \ d_{r+1}) = \text{reduce}(e'_1 \ \dots \ e'_k) \ e'_{k+1}$.

Consequently, constructing the pair $\langle \ell + 1, r + 1 \rangle$ with $e'_1 \ \dots \ e'_k \ e'_{k+1}$ is valid.

Finally, this construction gives e with the property of the statements once the induction reaches the pair $\langle n, m \rangle$. □

Let us illustrate the construction of this lemma with an example.

Example 2.27: *Constructing a common child for two columns.*

We consider two columns $q_2\ q_2\ q_3\ q_4\ q_3\ q_2$ and $q_2\ q_3\ q_3\ q_4\ q_3$. The base column for both of these is $q_2\ q_3\ q_4$. In the diagram below we give the first column on the left-hand side, the second column on the right-hand side, and the resulting column in the middle. Moreover, we draw arrows to indicate which element of which column motivates the presence of the element in the middle. Roughly, these arrows correspond to the case distinction in the proof of the lemma above: An arrow from the left indicates the first case, an arrow from the right the second case, and arrows from both sides the last case (which occurs exactly thrice – once for each element of the base column).

$$\begin{tikzcd}
q_2 \arrow[r] & q_2 & q_2 \arrow[l] \\
q_2 \arrow[r] & q_2 & q_3 \arrow[dl] \\
q_3 \arrow[r] & q_3 & q_3 \arrow[dl] \\
q_4 \arrow[dr] & q_3 & q_4 \arrow[dl] \\
q_3 \arrow[dr] & q_4 & q_3 \arrow[dddl] \\
q_2 \arrow[dr] & q_3 & \\
 & q_2 & \\
 & q_3 &
\end{tikzcd}$$

A transducer to find a separator

Based on the concept of columns, we construct now an (infinitely large) Σ-2^{Σ}-transducer $\mathcal{S}$ which captures how to compute separator sequences for configurations.

Definition 2.18: *Separator transducer.*

Let $Q_{\mathcal{T}}$ denote the states of $\mathcal{T}$, and let $Q_0 \subseteq Q_{\mathcal{T}}$ and $F \subseteq Q_{\mathcal{T}}$ the initial and accepting states, respectively. We call the Σ-2^{Σ}-transducer $\mathcal{S} = \left\langle Q_{\mathcal{T}}^{*}, Q_0^{*}, \Sigma \times 2^{\Sigma}, \Delta, F^{*} \right\rangle$

with

$$\left\langle c, \begin{bmatrix} v \\ S \end{bmatrix}, d \right\rangle \in \Delta \text{ if and only if } c \text{ and } d \text{ compute } S \text{ for } v$$

the separator transducer for $\mathcal{T}$.

Lemma 2.21. *$\langle v_1 \ \dots \ v_n, S_1 \ \dots \ S_n \rangle \in \llbracket \mathcal{S} \rrbracket$ if and only if there exists a tableau M which computes $S_1 \ \dots \ S_n$ for $v_1 \ \dots \ v_n$.*

Proof. This is an immediate consequence from Lemma 2.18. ☐

Due to its infinite size, this transducer does not have immediate use. However, the steps of this transducer are "upwards closed" with respect to $\preceq$; that is, if there is a step for some letter between columns c and d, then there is, for every c' with $c \preceq c'$, some d' such that $d \preceq d'$ and there is a step with the same letter between c' and d'. Moreover, this reasoning can also be applied "backward": for every step $\left\langle c, \begin{bmatrix} v \\ S \end{bmatrix}, d \right\rangle$ and $d \preceq d'$ there exists $c \preceq c'$ such that $\left\langle c', \begin{bmatrix} v \\ S \end{bmatrix}, d' \right\rangle$ is a step as well. We exploit this to construct a finite transducer for the same relation.

Lemma 2.22 (Monotonicity lemma)**.** *Let c and d be columns which compute S for v.*

Forwards: *For all c' with $c \preceq c'$ there exists d' such that $d \preceq d'$ and c' and d' also compute S for v.*

Backwards: *For all d' with $d \preceq d'$ there exists c' such that $c \preceq c'$ and c' and d' also compute S for v.*

Proof. Both directions of the statement can be proven in the same way. Therefore, it suffices to only consider the forward case. Because $c \preceq c'$ either $c = c'$ in which case the statement is trivial or there are $c_1^1 \ \dots \ c_{m_1}^1 \prec \dots \prec c_1^k \ \dots \ c_{m_k}^k$ such that

- $c = c_1^1 \ \dots \ c_{m_1}$, and
- $c' = c_1^k \ \dots \ c_{m_k}^k$.

Construct inductively a sequence $d_1^1 \ \dots \ d_{m_1}^1 \prec \dots \prec d_1^k \ \dots \ d_{m_k}^k$ such that

- $d = d_1^1 \ \dots \ d_{m_1}^1$, and
- $c_1^i \ \dots \ c_{m_i}^i$ and $d_1^i \ \dots \ d_{m_i}^i$ compute S for v for all $1 \leq i \leq k$.

Initially, the choice for $d^1_1 \ \dots \ d^1_{m_1}$ is valid since c and d compute S for v. For every step one can rely on Lemma 2.19 to construct $d^{i+1}_1 \ \dots \ d^{i+1}_{m_{i+1}}$. In particular, if $c^{i+1}_1 \ \dots \ c^{i+1}_{m_{i+1}}$ is obtained from $c^i_1 \ \dots \ c^i_{m_i}$ by repeating the ℓ-th letter at some position j, then one can construct $d^{i+1}_1 \ \dots \ d^{i+1}_{m_{i+1}}$ from $d^i_1 \ \dots \ d^i_{m_i}$ by repeating the ℓ-th letter at position j as well. □

Let us now shrink the separator transducer into a finite one.

Definition 2.19: *Reduced separator transducer.*

Let $Q_{\mathcal{T}}$ denote the states of $\mathcal{T}$, and let $Q_0 \subseteq Q_{\mathcal{T}}$ and $F \subseteq Q_{\mathcal{T}}$ the initial and accepting states, respectively. We call the Σ-2^Σ-transducer $\mathcal{S} = \left\langle \mathit{bases}(\mathcal{T}), \mathit{bases}(\mathcal{T}) \cap Q_0^*, \Sigma \times 2^\Sigma, \Delta, \mathit{bases}(\mathcal{T}) \cap F^* \right\rangle$ with

$$\left\langle c, \begin{bmatrix} v \\ S \end{bmatrix}, d \right\rangle \in \Delta \text{ if and only if}$$

$$\text{there are } c' \text{ and } d' \text{ such that } c \preceq c' \text{ and } d \preceq d' \text{ which compute } S \text{ for } v$$

the *reduced* separator transducer for $\mathcal{T}$.

Lemma 2.23. *The separator transducer and the reduced separator transducer accept the same language.*

Proof. By the definition of the transition relation of the reduced separator transducer, it is immediate that every run $c_1 \ \dots \ c_n$ in the separator transducer has a corresponding run $\mathrm{reduce}(c_1) \ \dots \ \mathrm{reduce}(c_n)$ in the reduced separator transducer on the same word. Therefore, the reduced separator transducer accepts all words that the separator transducer accepts.

Let Δ denote the steps from the separator transducer and Δ_r the steps from the reduced separator transducer. Pick any word $v_1 \ \dots \ v_n$ accepted by the reduced separator transducer. Thus, there is an accepting run $b_0 \ \dots \ b_n$ of the reduced separator transducer on $v_1 \ \dots \ v_n$. Proceed by induction to n to obtain a run $c_0 \ \dots \ c_n$ of the separator transducer on $v_1 \ \dots \ v_n$ such that $\mathrm{reduce}(c_0) \ \dots \ \mathrm{reduce}(c_n) = b_0 \ \dots \ b_n$:

$n = 0$: Because b_0 is a column this case is immediate.

$n > 0$: Let $c_0 \ \dots \ c_{n-1}$ be a run in the separator transducer on $v_1 \ \dots \ v_{n-1}$ with the desired properties. Since $\langle b_{n-1}, v_n, b_n \rangle \in \Delta_r$ there are d and e with $b_{n-1} \preceq d$ and $b \preceq e$ such that $\langle d, v_n, e \rangle \in \Delta$. Because $b_{n-1} \preceq c_{n-1}$ as well, there is, by Lemma 2.20, c'_{n-1}

with $c_{n-1} \preceq c'_{n-1}$ and $d \preceq c'_{n-1}$. Applying the backwards direction of Lemma 2.22 repeatedly, we obtain a run $c'_0 \ \dots \ c'_{n-1}$ in the separator transducer on $v_1 \ \dots \ v_{n-1}$ such that $\text{reduce}(c'_0) \ \dots \ \text{reduce}(c'_{n-1}) = b_0 \ \dots \ b_{n-1}$ because $c_i \preceq c'_i$ for all $1 \leq i \leq n-1$. Relying on the definition of Δ_r and the forwards direction of Lemma 2.22, we get some c'_n with $b_n \preceq e \preceq c'_n$ such that $\left\langle c'_{n-1}, v_n, c'_n \right\rangle \in \Delta$. In conclusion, $c'_0 \ \dots \ c'_n$ has the desired properties. □

Based on Lemma 2.16, one can modify the reduced separator transducer to obtain a Σ-Σ-transducer that captures $\Rightarrow_{\mathcal{V}_{trap}}$. The idea is to replace every step $\left\langle c, \begin{bmatrix} u \\ S \end{bmatrix}, d \right\rangle$ with steps $\left\langle c, \begin{bmatrix} v \\ u \end{bmatrix}, d \right\rangle$ for all $v \notin S$. In this way, we combine the computation of a separator and the check that the origin does not satisfy it into one transducer.

However, it remains to show that the steps of the reduced separator transducer can be computed effectively. For this, we introduce a single-player game that can be won if and only if the input is a transition of the reduced separator transducer.

Definition 2.20: *Step game.*

For $\left\langle b^1_1 \ \dots \ b^1_n, \begin{bmatrix} u \\ S \end{bmatrix}, b^2_1 \ \dots \ b^2_m \right\rangle$ where $b^1_1 \ \dots \ b^1_n, b^2_1 \ \dots \ b^2_m \in \mathit{bases}(\mathcal{T})$, $u \in \Sigma$, and $S \in 2^{\Sigma}$ we consider the following game: The current state of the game is represented by a triple $\langle \ell, I, r \rangle$ where

- $1 \leq \ell \leq n$,
- $I \supseteq S$, and
- $1 \leq r \leq m$.

In every turn, the player can play any step $\left\langle q, \begin{bmatrix} x \\ y \end{bmatrix}, p \right\rangle$ from $\mathcal{T}$. The player loses immediately if

- $y \in I$,
- $q \notin \{b^1_1 \ \dots \ b^1_\ell\}$ and, if $\ell < n$, $q \neq b^1_{\ell+1}$, or
- $p \notin \{b^2_1 \ \dots \ b^2_r\}$ and, if $r < m$, $p \neq b^1_{\ell+1}$.

Otherwise the game moves to a new state $\langle \ell', I', r' \rangle$ such that

- $\ell' = \ell$ if $q \in \{b^1_1 \ \dots \ b^1_\ell\}$ and, otherwise, $\ell' = \ell + 1$,

- $r' = r$ if $p \in \{b^2_1 \ \dots \ b^1_r\}$ and, otherwise, $r' = r + 1$, and
- $I' = I \setminus \{x\}$.

Initially, the state is $\langle 0, \Sigma \setminus \{u\}, 0\rangle$. The player wins the game if the state becomes $\langle n, S, m\rangle$.

Essentially, the player in this game constructs with a winning strategy two columns c_1 and c_2 with reduce$(c_1) = b^1_1 \ \dots \ b^1_n$ and reduce$(c_2) = b^2_1 \ \dots \ b^2_m$ which compute S for u.

Lemma 2.24. *There is a winning strategy in the step game for*

$$\left\langle b^1_1 \ \dots \ b^1_n, \left[\begin{smallmatrix} u \\ S \end{smallmatrix}\right], b^2_1 \ \dots \ b^2_m \right\rangle$$

if and only if

$$\left\langle b^1_1 \ \dots \ b^1_n, \left[\begin{smallmatrix} u \\ S \end{smallmatrix}\right], b^2_1 \ \dots \ b^2_m \right\rangle$$

is a step in the reduced separator transducer.

Proof. Assume a winning strategy $\left\langle q_1, \left[\begin{smallmatrix} x_1 \\ y_1 \end{smallmatrix}\right], p_1 \right\rangle, \dots, \left\langle q_k, \left[\begin{smallmatrix} x_k \\ y_k \end{smallmatrix}\right], p_k \right\rangle$ for the step game. This strategy implies a removal sequence $x_1, \dots, x_k$ for the columns $q_1 \ \dots \ q_k$ and $p_1 \ \dots \ p_k$ for u. From the rules of the game, reduce$(q_1 \ \dots \ q_k) = b^1_1 \ \dots \ b^1_n$ and reduce$(p_1 \ \dots \ p_k) = b^2_1 \ \dots \ b^2_m$ is immediate. Consequently, $\left\langle q_1 \ \dots \ q_k, \left[\begin{smallmatrix} u \\ S \end{smallmatrix}\right], p_1 \ \dots \ p_k \right\rangle$ is a step in the separator transducer and, therefore, $\left\langle b^1_1 \ \dots \ b^1_n, \left[\begin{smallmatrix} u \\ S \end{smallmatrix}\right], b^2_1 \ \dots \ b^2_m \right\rangle$ is a step in the reduced separator transducer.

On the other hand, assume $\left\langle b^1_1 \ \dots \ b^1_n, \left[\begin{smallmatrix} u \\ S \end{smallmatrix}\right], b^2_1 \ \dots \ b^2_m \right\rangle$ is a step in reduced separator transducer. Thus, there exists a step $\left\langle q_1 \ \dots \ q_k, \left[\begin{smallmatrix} u \\ S \end{smallmatrix}\right], p_1 \ \dots \ p_k \right\rangle$ in the separator transducer. Hence, $q_1 \ \dots \ q_k$ and $p_1 \ \dots \ p_k$ compute S for u. This means, there exists $\left\langle q_1, \left[\begin{smallmatrix} x_1 \\ y_1 \end{smallmatrix}\right], p_1 \right\rangle, \dots, \left\langle q_k, \left[\begin{smallmatrix} x_k \\ y_k \end{smallmatrix}\right], p_k \right\rangle$ such that

- $y_1 = u$,
- $y_{i+1} \in \{x_1, \dots, x_i\}$ for all $1 \leq i < k$, and
- $S = \Sigma \setminus \{x_1, \dots, x_k\}$.

This sequence of steps is a winning strategy for the step game. $\square$

Example 2.28: *Steps in a reduced separator transducer.*

For the running example, we give here some steps that originate in the unique initial state q_0. Moreover, we present a winning strategy in the step game to justify the step. In this winning strategy, we color those steps that contribute a new element to the removal sequence.

Step	Winning strategy
$\left\langle q_0, \begin{bmatrix} n \\ \{t\} \end{bmatrix}, q_1 \right\rangle$	$\left\langle q_0, \begin{bmatrix} n \\ n \end{bmatrix}, q_1 \right\rangle$
$\left\langle q_0, \begin{bmatrix} n \\ \emptyset \end{bmatrix}, q_2 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_2 \right\rangle$
$\left\langle q_0, \begin{bmatrix} n \\ \emptyset \end{bmatrix}, q_2\ q_4 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_2 \right\rangle$ $\left\langle q_0, \begin{bmatrix} n \\ t \end{bmatrix}, q_4 \right\rangle$
$\left\langle q_0, \begin{bmatrix} n \\ \emptyset \end{bmatrix}, q_2\ q_1 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_2 \right\rangle$ $\left\langle q_0, \begin{bmatrix} n \\ n \end{bmatrix}, q_1 \right\rangle$
$\left\langle q_0, \begin{bmatrix} n \\ \emptyset \end{bmatrix}, q_2\ q_1\ q_4 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_2 \right\rangle$ $\left\langle q_0, \begin{bmatrix} n \\ n \end{bmatrix}, q_1 \right\rangle$ $\left\langle q_0, \begin{bmatrix} n \\ t \end{bmatrix}, q_4 \right\rangle$
$\left\langle q_0, \begin{bmatrix} n \\ \emptyset \end{bmatrix}, q_2\ q_4\ q_1 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_2 \right\rangle$ $\left\langle q_0, \begin{bmatrix} n \\ t \end{bmatrix}, q_4 \right\rangle$ $\left\langle q_0, \begin{bmatrix} n \\ n \end{bmatrix}, q_1 \right\rangle$
$\left\langle q_0, \begin{bmatrix} t \\ \emptyset \end{bmatrix}, q_4 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_4 \right\rangle$
$\left\langle q_0, \begin{bmatrix} t \\ \emptyset \end{bmatrix}, q_4\ q_1 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_4 \right\rangle$ $\left\langle q_0, \begin{bmatrix} n \\ n \end{bmatrix}, q_1 \right\rangle$
$\left\langle q_0, \begin{bmatrix} t \\ \emptyset \end{bmatrix}, q_4\ q_2 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_4 \right\rangle$ $\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_2 \right\rangle$
$\left\langle q_0, \begin{bmatrix} t \\ \emptyset \end{bmatrix}, q_4\ q_1\ q_2 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_4 \right\rangle$ $\left\langle q_0, \begin{bmatrix} n \\ n \end{bmatrix}, q_1 \right\rangle$ $\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_2 \right\rangle$
$\left\langle q_0, \begin{bmatrix} t \\ \emptyset \end{bmatrix}, q_4\ q_2\ q_1 \right\rangle$	$\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_4 \right\rangle$ $\left\langle q_0, \begin{bmatrix} t \\ n \end{bmatrix}, q_2 \right\rangle$ $\left\langle q_0, \begin{bmatrix} n \\ n \end{bmatrix}, q_1 \right\rangle$

Any play in the step game has a clear notion of “making progress”. In particular, any move that does not advance the state $\langle \ell, I, r\rangle$ of the game; that is, leads to a state $\langle \ell', I', r'\rangle$ where either $\ell < \ell'$, $|I| > |I'|$, or $r < r'$, can be omitted without changing the outcome of the game. Consequently, one can bound the length of a winning strategy.

Lemma 2.25. *If there is a winning strategy in the step game for*

$$\left\langle b_1^1 \dots b_n^1, \begin{bmatrix} u \\ S \end{bmatrix}, b_1^2 \dots b_m^2 \right\rangle$$

then there is one of length $n + m + (|\Sigma \setminus S| - 1)$.

We can now put all these results together to obtain that Problem 2.2 for $\mathcal{V}_{trap}$ can be solved in PSpace (which, by Theorem 2.3, also holds for $\mathcal{V}_{siphon}$).

Theorem 2.6 ([ERW22b]). *Problem 2.2 for* $\mathcal{V}_{trap}$ *is in* PSpace.

Proof. Let $\mathcal{S}$ be the reduced separator transducer. Observed that $\Rightarrow_{\overset{\leftrightharpoons}{\mathcal{V}}_{trap}}$ is equivalent to

$$[\![\mathcal{S}]\!] \circ \overset{\leftrightharpoons}{\overline{[\![\mathcal{V}_{trap}]\!]}}$$

because of Lemma 2.21 and Lemma 2.23. Note that any state of this transducer for $\Rightarrow_{\mathcal{V}_{trap}}$ can be stored in polynomial space of the input to Problem 2.2 for $\mathcal{V}_{trap}$. Moreover, due to Lemma 2.24 and Lemma 2.25, the steps of this transducer can also be computed in polynomial space. Therefore, one can look for an accepted word in a transducer that captures $[\![\mathcal{I}]\!] \circ \Rightarrow_{\mathcal{V}_{trap}} \circ [\![\mathcal{B}]\!]$ in polynomial space. $\square$

2.7 Topologies

RMC captures a large class of parameterized systems. However, many parameterized systems can be described in simpler terms because there are natural restrictions for how they behave [FO97; WL89; KM95; Del00a; Lin+16]. We say groups of systems that can be captured by common restrictions share a *topology*.

The interest in these topologies is motivated by results from [ERW22a] and [Lin+16]. In particular, we want to introduce generalization procedures which, given one inductive statement, return a family of inductive statement. This can be used to inform a learning procedure on its own. Here, however, we introduce a learning procedure in Chapter 3 and use the observations of this section to obtain, in a similar way, generalization procedures which aide the learning process that we describe later.

For instance, we consider the *ring topology*. Roughly speaking, a parameterized system that adheres to the ring topology, or, for short, is a *ring*, is a system in which all agents form a ring and only two adjacent agents (in this ring) can interact. Moreover, all adjacent agents interact in the same way; that is, if there is some interaction possible between the first and the second agent the same interaction is possible between the second and third agent and the third and the fourth agent and so on.

Example 2.29: *Circular token passing as a ring.*

The RTS from Example 2.20 that models circular token passing is, almost, a ring. Specifically, we do not consider it a ring yet because, for every transition, only two adjacent agents interact but there is a restriction on the state of all the other agents: they only can be in state n. Thus, every transition enforces the invariant that there is exactly one token. We modify the language of all transitions to remove this invariant:

$$\left(\begin{bmatrix} n \\ n \end{bmatrix} \middle| \begin{bmatrix} t \\ t \end{bmatrix}\right)^* \left(\begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix}\right) \left(\begin{bmatrix} n \\ n \end{bmatrix} \middle| \begin{bmatrix} t \\ t \end{bmatrix}\right)^* \mid \begin{bmatrix} n \\ t \end{bmatrix} \left(\begin{bmatrix} n \\ n \end{bmatrix} \middle| \begin{bmatrix} t \\ t \end{bmatrix}\right)^* \begin{bmatrix} t \\ n \end{bmatrix}.$$

In this way, all adjacent agents share one common interaction: the first agent moves from t to n while the adjacent agent moves from n to t. Therefore, we consider this example a ring now.

In this section, we consider three topologies:

- The aforementioned rings.
- A slight variant of rings, called *bows*, where we allow one distinguished agent to behave differently from all others. For instance, consider Example 2.2 in the version where every transition does *not* enforce that there only is a single token (cp. Example 2.9). In this system, the first agent behaves differently from all others: this agent does not accept a token from the last agent.
- A topology we call *crowds*. In a crowd all agents are anonymous; that is, they do not have an identity but are interchangeable. More formally, the language of the transitions is closed under the permutation of the letters of words; e. g. if $\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \begin{bmatrix} a_3 \\ b_3 \end{bmatrix}$ is a transition, so are $\begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \begin{bmatrix} a_3 \\ b_3 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \end{bmatrix}$, $\begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} \begin{bmatrix} a_3 \\ b_3 \end{bmatrix}$, $\begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \begin{bmatrix} a_3 \\ b_3 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \end{bmatrix}$,

$\begin{bmatrix}a_3\\b_3\end{bmatrix}\begin{bmatrix}a_1\\b_1\end{bmatrix}\begin{bmatrix}a_2\\b_2\end{bmatrix}$, and $\begin{bmatrix}a_3\\b_3\end{bmatrix}\begin{bmatrix}a_2\\b_2\end{bmatrix}\begin{bmatrix}a_1\\b_1\end{bmatrix}$. In this topology, interaction is restricted to a set of agents of some constant size k meeting to change their states. Any such meeting, however, can only occur if some global condition on all other agents is met and, if it is possible, all other agents might react to the meeting.

The ring topology

Let us first take a look at rings. In a ring, the last and the first agent are adjacent to each other. We introduce some simplifying notation for this; that is, we write $v_{i\oplus 1}$ to refer to v_{i+1} if $i < n$ and, otherwise, v_1 in any word $v_1 \dots v_n$. Symmetrically, we refer to the position before i as $i \ominus 1$.

A ring is fully specified by all the possible interactions that two adjacent agents can do (cp. Example 2.29).

Definition 2.21: *Ring topology.*

We call any $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ a ring if there is a set $P \subseteq (\Sigma \times \Sigma) \times (\Sigma \times \Sigma)$ such that

$$\mathcal{L}(\mathcal{T}) = \bigcup_{\begin{bmatrix}v\\v'\end{bmatrix},\begin{bmatrix}u\\u'\end{bmatrix} \in P} I^* \begin{bmatrix}v\\v'\end{bmatrix}\begin{bmatrix}u\\u'\end{bmatrix} I^* \cup \begin{bmatrix}u\\u'\end{bmatrix} I^* \begin{bmatrix}v\\v'\end{bmatrix}$$

where $I = \left\{ \begin{bmatrix}v\\v\end{bmatrix} : v \in \Sigma \right\}$. We say P is the set of patterns of $\mathcal{R}$. Moreover, we call the transition $x_1 \dots x_n \rightsquigarrow_{\mathcal{T}} y_1 \dots y_n$ where $\begin{bmatrix}x_i\\y_i\end{bmatrix}\begin{bmatrix}x_{i\oplus 1}\\y_{i\oplus 1}\end{bmatrix} \in P$ and $x_j = y_j$ for all $j \notin \{i, i \oplus 1\}$ a *realization* of $\begin{bmatrix}x_i\\y_i\end{bmatrix}\begin{bmatrix}x_{i\oplus 1}\\y_{i\oplus 1}\end{bmatrix}$ at i.

Example 2.30: *Ring definition of circular token passing.*

The RTS from Example 2.29 is a ring because the set

$$P = \left\{ \begin{bmatrix}t\\n\end{bmatrix}\begin{bmatrix}n\\t\end{bmatrix} \right\}$$

yields exactly the transitions of the system.
The transitions $\begin{bmatrix}t\\n\end{bmatrix}\begin{bmatrix}n\\t\end{bmatrix}$, $\begin{bmatrix}t\\n\end{bmatrix}\begin{bmatrix}n\\t\end{bmatrix}\begin{bmatrix}t\\t\end{bmatrix}$, and $\begin{bmatrix}t\\n\end{bmatrix}\begin{bmatrix}n\\t\end{bmatrix}\begin{bmatrix}n\\n\end{bmatrix}$ are realizations of $\begin{bmatrix}t\\n\end{bmatrix}\begin{bmatrix}n\\t\end{bmatrix}$ at 1.

Based on the ring topology of a regular transition system $\mathcal{R}$ one can characterize $\overline{\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})}$, $\overline{\text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R})}$, $\text{Inductive}_{\mathcal{V}_{flow}}(\mathcal{R})$ in an alternative way. For this, recall that $v_1 \ \ldots \ v_n \models_{\mathcal{V}} I_1 \ \ldots \ I_n$ for

$\mathcal{V} = \mathcal{V}_{trap}$ if there exists $1 \leq i \leq n$ such that $v_i \in I_i$,

$\mathcal{V} = \mathcal{V}_{siphon}$ if there exists *no* $1 \leq i \leq n$ such that $v_i \in I_i$, and

$\mathcal{V} = \mathcal{V}_{flow}$ if there exists *exactly one* $1 \leq i \leq n$ such that $v_i \in I_i$.

In other words, all these interpretations can be described in terms of the size of the set $hit = \{i \in \{1, \ldots, n\} \mid v_i \in I_i\}$: $v_1 \ \ldots \ v_n \models_{\mathcal{V}_{trap}} I_1 \ \ldots \ I_n$ if and only if $|hit| > 0$, $v_1 \ \ldots \ v_n \models_{\mathcal{V}_{siphon}} I_1 \ \ldots \ I_n$ if and only if $|hit| = 0$, and $v_1 \ \ldots \ v_n \models_{\mathcal{V}_{flow}} I_1 \ \ldots \ I_n$ if and only if $|hit| = 1$.

In a ring, transitions can only make two adjacent agents change their states. Consequently, the size of the set *hit* can change by at most 2. Moreover, assume $x_1 \ \ldots \ x_n \rightsquigarrow_{\mathcal{T}} y_1 \ \ldots \ y_n$ is a realization of $\begin{bmatrix} x_i \\ y_i \end{bmatrix} \begin{bmatrix} x_{i\oplus 1} \\ y_{i\oplus 1} \end{bmatrix}$ at i. We can observe that

$$\{i \in \{1, \ldots, n\} \setminus \{i, i \oplus 1\} \mid x_i \in I_i\} = \{i \in \{1, \ldots, n\} \setminus \{i, i \oplus 1\} \mid y_i \in I_i\}$$

because $x_j = y_j$ for all $j \notin \{i, i \oplus 1\}$. In other words, there is some crucial interaction between the pattern $\begin{bmatrix} x_i \\ y_i \end{bmatrix} \begin{bmatrix} x_{i\oplus 1} \\ y_{i\oplus 1} \end{bmatrix}$ and the letters I_i and $I_{i\oplus 1}$ of the statement which renders the statement non-inductive.

Roughly speaking, for the interpretations $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ and every non-inductive statement $I_1 \ \ldots \ I_n$ for this interpretation, there are two adjacent letters $I_i \ I_{i\oplus 1}$ and a pattern $\begin{bmatrix} v \\ v' \end{bmatrix} \begin{bmatrix} u \\ u' \end{bmatrix}$ such that $v \ u$ satisfies $I_i \ I_{i\oplus 1}$ and $v' \ u'$ does not. This intuition translates to rigorous proofs.

For the interpretation $\mathcal{V}_{flow}$, the construction is more involved. This is because $\mathcal{V}_{flow}$ needs the size of the set *hit* to be exactly 1. Thus, whether a configuration $v_1 \ \ldots \ v_n$ satisfies a statement $I_1 \ \ldots \ I_n$ is a combination of two queries:

$\exists i \ . \ v_i \in I_i$**:** "Is there at least one index where the configuration letter is part of the letter of the statement?"

$\forall j \in \{1, \ldots, n\} \setminus \{i\} \ . \ v_j \notin I_j$**:** "Is there no other index where the configuration letter is part of the letter of the statement?"

This leads to a more nuanced characterization for $\text{Inductive}_{\mathcal{V}_{flow}}(\mathcal{R})$.

Non-inductive statements for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ in rings

Definition 2.22: *Non-inductive pairs in rings.*

Let P be the patterns of a ring $\mathcal{R}$ and $\mathcal{V}$ some interpretation. We call a pair $\langle A, B\rangle \in 2^{\Sigma} \times 2^{\Sigma}$ *non-inductive* for $\mathcal{V}$ if there exists $\begin{bmatrix} v \\ v' \end{bmatrix} \begin{bmatrix} u \\ u' \end{bmatrix} \in P$ such that $v\ u \models_{\mathcal{V}} A\ B$ and $v'\ u' \not\models_{\mathcal{V}} A\ B$.

Additionally, to simplify the following statements, we introduce a syntax to refer to adjacent letters of some statement $I_1\ \ldots\ I_n \in \left(2^{\Sigma}\right)^*$; namely, $\mathrm{adj}(I_1\ \ldots\ I_n) = \{\langle I_n, I_1\rangle\} \cup \{\langle I_i, I_{i+1}\rangle : i \in \{1, \ldots, n-1\}\}$. We proceed by describing $\overline{\mathrm{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})}$ and $\overline{\mathrm{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R})}$ for a ring $\mathcal{R}$, specifically. Let us, first, focus on $\overline{\mathrm{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})}$. Essentially, any statement that contains a non-inductive pair for $\mathcal{V}_{trap}$ is part of this set – with the exception of universally true statements. However, universally true statements can be identified syntactically since they contain at least one letter that is Σ.

Lemma 2.26. *Let P be the patterns of the ring $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$. Then*

$$\overline{\mathrm{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})} = \left\{ I \in \left(2^{\Sigma} \setminus \{\Sigma\}\right)^* \,\middle|\, \begin{array}{l} \textit{there is } \langle A, B\rangle \in \mathrm{adj}(I) \\ \textit{that is non-inductive for } \mathcal{V}_{trap} \end{array} \right\}. \tag{2.2}$$

Proof.

"$\subseteq$":
Pick $I_1\ \ldots\ I_n \in \overline{\mathrm{Inductive}_{\mathcal{V}_{trap}}}$. There is a transition $x_1\ \ldots\ x_n \rightsquigarrow_{\mathcal{T}} y_1\ \ldots\ y_n$ such that $x_1\ \ldots\ x_n \models_{\mathcal{V}_{trap}} I_1\ \ldots\ I_n$ and $y_1\ \ldots\ y_n \not\models_{\mathcal{V}_{trap}} I_1\ \ldots\ I_n$. In other words, $y_j \notin I_j$ for all $1 \leq j \leq n$ and, therefore, $I_j \neq \Sigma$. Moreover, $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \ldots \begin{bmatrix} x_n \\ y_n \end{bmatrix}$ is a realization of $\begin{bmatrix} x_i \\ y_i \end{bmatrix} \begin{bmatrix} x_{i\oplus 1} \\ y_{i\oplus 1} \end{bmatrix}$ at i. From $x_j = y_j$ for all $j \in \{1, \ldots, n\} \setminus \{i, i \oplus 1\}$ and $y_1\ \ldots\ y_n \not\models_{\mathcal{V}_{trap}} I_1\ \ldots\ I_n$ follows that $x_y \notin I_j$ for all $j \in \{1, \ldots, n\} \setminus \{i, i \oplus 1\}$. Therefore, either $x_i \in I_i$ or $x_{i\oplus 1} \in I_{i\oplus 1}$, and $y_i \notin I_i$ and $y_{i\oplus 1} \notin I_{i\oplus 1}$. Consequently, the pattern $\begin{bmatrix} x_i \\ y_i \end{bmatrix} \begin{bmatrix} x_{i\oplus 1} \\ y_{i\oplus 1} \end{bmatrix}$ proves that $\langle I_i, I_{i\oplus 1}\rangle$ is non-inductive. Thus, $I_1\ \ldots\ I_n \in \left\{ I \in \left(2^{\Sigma} \setminus \{\Sigma\}\right)^* \,\middle|\, \begin{array}{l} \text{there is } \langle A, B\rangle \in \mathrm{adj}(I) \\ \text{that is non-inductive for } \mathcal{V}_{trap} \end{array} \right\}$.

"$\supseteq$":
Pick $I_1\ \ldots\ I_n \in \left\{ I \in \left(2^{\Sigma} \setminus \{\Sigma\}\right)^* \,\middle|\, \begin{array}{l} \text{there is } \langle A, B\rangle \in \mathrm{adj}(I) \\ \text{that is non-inductive for } \mathcal{V}_{trap} \end{array} \right\}$. There is i such

that $\langle I_i, I_{i\oplus 1}\rangle$ is non-inductive for $\mathcal{V}_{\mathit{trap}}$. Therefore, there exists $\begin{bmatrix} x_i \\ y_i \end{bmatrix} \begin{bmatrix} x_{i\oplus 1} \\ y_{i\oplus 1} \end{bmatrix} \in P$ such that $x_i \in I_i$ or $x_{i\oplus 1} \in I_{i\oplus 1}$, and $y_i \notin I_i$ and $y_{i\oplus 1} \notin I_{i\oplus 1}$. Choose x_j, for all $j \in \{1, \ldots, n\} \setminus \{i, i\oplus 1\}$, such that $x_j \notin I_j$. Based on these choices, construct the realization $x_1 \ \ldots \ x_n \rightsquigarrow_{\mathcal{T}} x_1 \ \ldots \ x_{i-1} \ y_i \ y_{i+1} \ x_{i+2} \ \ldots \ x_n$ (or, if $i = n$, $x_1 \ \ldots \ x_n \rightsquigarrow_{\mathcal{T}} y_{i\oplus 1} \ x_2 \ \ldots \ x_{n-1} \ y_i$) of $\begin{bmatrix} x_i \\ y_i \end{bmatrix} \begin{bmatrix} x_{i\oplus 1} \\ y_{i\oplus 1} \end{bmatrix}$ at i. This transition proves that $I_1 \ \ldots \ I_n \in \overline{\mathsf{Inductive}_{\mathcal{V}_{\mathit{trap}}}(\mathcal{R})}$ because, by construction, the origin of this transition satisfies the statement and the target does not. □

Example 2.31: *A non-trap in circular token passing.*

Recall the circular token passing system from Example 2.30. In order to capture this system as a ring, we changed the transitions in such a way that every single transition does not enforce any more that there is only one unique token. Recall that, for the interpretation $\mathcal{V}_{\mathit{trap}}$, the language $\emptyset^* \ \{n\} \ \emptyset^* \ \{n\} \ \emptyset^*$ is a language of inductive statements – but only if all transitions enforce the invariant of a single token (cp. Example 2.9). Specifically, the statement $\{n\} \ \emptyset \ \{n\}$ is not inductive without this invariant in every transition. According to Lemma 2.26 this means there is one non-inductive pair in $\mathsf{adj}(\{n\} \ \emptyset \ \{n\})$ (w. r. t. to the patterns $\left\{\begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix}\right\}$). For instance, $\emptyset \ \{n\}$ is non-inductive because of $\begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix}$: $t \, n \models_{\mathcal{V}_{\mathit{trap}}} \emptyset \ \{n\}$ and $n \, t \not\models_{\mathcal{V}_{\mathit{trap}}} \emptyset \ \{n\}$.

Note here that the pairs $\{n\} \ \{n\}$ and $\{n\} \ \emptyset$ both are *not* non-inductive[a]. For the former, observe that $n \, t \models_{\mathcal{V}_{\mathit{trap}}} \{n\} \ \{n\}$, and, for the latter, $t \, n \not\models_{\mathcal{V}_{\mathit{trap}}} \{n\} \ \emptyset$.

---

[a] Or, in other words, these two pairs are inductive.

For the interpretation $\mathcal{V}_{\mathit{siphon}}$, the proof works in the same way. Here, however, any statement where one letter is Σ is unsatisfiable instead of universally true. We get a result of the same kind.

Lemma 2.27. *Let P be the patterns of the ring $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$. Then*

$$\overline{\mathsf{Inductive}_{\mathcal{V}_{\mathit{siphon}}}(\mathcal{R})} = \left\{ I \in \left(2^{\Sigma} \setminus \{\Sigma\}\right)^* \,\middle|\, \begin{array}{l} \textit{there is } \langle A, B\rangle \in \mathsf{adj}(I) \\ \textit{that is non-inductive for } \mathcal{V}_{\mathit{siphon}} \end{array} \right\}. \tag{2.3}$$

Proof. The proof works analogously as for Lemma 2.26. □

From this observation, it is straightforward to construct DFAs for the languages

$\overline{\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})}$ and $\overline{\text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R})}$: On the one hand, one can construct a DFA for the language $\left(2^\Sigma \setminus \{\Sigma\}\right)^*$ with two states. On the other hand, one can construct a DFA which remembers the first symbol it reads, and, at any moment, the last symbol it read. If this DFA encounters a non-inductive pair, then it moves into an accepting sink. Otherwise, it may also accept if the last symbol and the first symbol form a non-inductive pair. Because this automaton needs to remember two symbols at any moment, it can be constructed with $\mathcal{O}(\left|2^\Sigma\right|^2)$ states. Since the intersection of these languages is $\left\{ I \in \left(2^\Sigma \setminus \{\Sigma\}\right)^* \,\middle|\, \begin{array}{l}\text{there is } \langle A, B\rangle \in \text{adj}(I) \\ \text{that is non-inductive for } \mathcal{V}_{siphon}\end{array} \right\}$, one can obtain a DFA for the language itself via the product construction for DFAs.

Corollary 2.5. *Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$ be a ring. One can effectively construct two DFAs, each of them with $\mathcal{O}(\left|2^\Sigma\right|^2)$ states, recognizing $\overline{\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})}$ and $\overline{\text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R})}$, respectively.*

Corollary 2.6. *Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$ be a ring. One can effectively construct two DFAs, each of them with $\mathcal{O}(\left|2^\Sigma\right|^2)$ states, recognizing $\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$ and $\text{Inductive}_{\mathcal{V}_{siphon}}(\mathcal{R})$, respectively.*

Inductive statements for $\mathcal{V}_{flow}$ in rings

For the interpretation $\mathcal{V}_{flow}$ the construction is more elaborate. Recall that

$$v_1 \ \ldots \ v_n \models_{\mathcal{V}_{flow}} I_1 \ \ldots \ I_n \text{ if and only if } |\underbrace{\{i \in \{1, \ldots, n\} \mid v_i \in I_i\}}_{hit}| = 1.$$

In a ring, only two adjacent agents can change their state in any transition. Therefore, for an inductive statement, these state changes must not change the size of the set *hit*. Let us illustrate this with an example.

Example 2.32: *Flows in circular token passing.*

We consider, again, the ring with patterns $\left\{ \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \right\}$. Observed that, for the pairs $\{n\}\ \{n\}$ and $\{t\}\ \{t\}$, changes according to the pattern of this ring does not change the size of the set *hit*. For this, we chart these sets for the pairs, and the configurations $t\ n$ and $n\ t$:

Pair	$t\ n$	$n\ t$
$\{n\}\ \{n\}$	$\{2\}$	$\{1\}$
$\{t\}\ \{t\}$	$\{1\}$	$\{2\}$

For this reason, all statements from the languages $\{n\}^*$ and $\{t\}^*$ are inductive for the interpretation $\mathcal{V}_{\mathit{flow}}$.

There are cases where there is no realization of a pattern such that the origin of the resulting transition can satisfy the statement. For instance, consider the pair $\{t\}\ \{n\}$ and the configuration $t\ n$. Here the set *hit* is $\{1, 2\}$. Thus, any realization of the pattern $\begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix}$ at some index i cannot render a statement non-inductive if the i-th and $i \oplus 1$-th letters of the statement are $\{t\}\ \{n\}$. Therefore, $\{t\}\ \{n\}$ is an inductive statement for the interpretation $\mathcal{V}_{\mathit{flow}}$ since $t\ n \not\models_{\mathcal{V}_{\mathit{flow}}} \{t\}\ \{n\}$.

Based on these observations, we introduce a notion to characterize pairs of letters in statements which

- do not change the size of the set *hit* in realizations of the patterns of the ring, or
- already enforce the set *hit* to have at least two elements for any realization of the pattern.

Definition 2.23: *Compatible patterns for* $\mathcal{V}_{\mathit{flow}}$.

Let P be the patterns of a ring $\mathcal{R}$. We call $\langle A_1, A_2 \rangle$ *compatible* with $\begin{bmatrix} v_1 \\ u_1 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix}$ for $\mathcal{V}_{\mathit{flow}}$ if either $|\{i \in \{1,2\} \mid v_i \in A_i\}| = |\{i \in \{1,2\} \mid u_i \in A_i\}| \in \{0,1\}$ or $|\{i \in \{1,2\} \mid v_i \in A_i\}| = 2$. Moreover, we call the pair $\langle A_1, A_2 \rangle$ *compatible* with P if $\langle A_1, A_2 \rangle$ is *compatible* with $\begin{bmatrix} v_1 \\ u_1 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix}$ for $\mathcal{V}_{\mathit{flow}}$ for all $\begin{bmatrix} v_1 \\ u_1 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix} \in P$.

Based on the results for the interpretations $\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$, one would expect that all inductive statements are those where every adjacent pair is compatible with the patterns for $\mathcal{V}_{\mathit{flow}}$. Although all statements where every adjacent pair is compatible are inductive, there are more. For this, consider the following example.

Example 2.33: *Flows with incompatible pairs.*

Consider a ring $\mathcal{R} = \langle \{a, b\}, \mathcal{I}, \mathcal{T} \rangle$ with patterns $P = \left\{ \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \right\}$. We demonstrate that, in this ring, the statement $\emptyset\ \{a\}\ \emptyset\ \{a, b\}$ is inductive, although the pairs $\emptyset\ \{a\}$ and $\{a\}\ \emptyset$ are not compatible with P for $\mathcal{V}_{flow}$. For this, we give in the following table all transitions of length 4 and the sets *hit* for the origin and the target of these transitions.

hit before	Transition	*hit* after
$\{2, 4\}$	$a\ a\ a\ a \rightsquigarrow_{\mathcal{T}} b\ b\ a\ a$	$\{4\}$
$\{2, 4\}$	$a\ a\ a\ b \rightsquigarrow_{\mathcal{T}} b\ b\ a\ b$	$\{4\}$
$\{2, 4\}$	$a\ a\ b\ a \rightsquigarrow_{\mathcal{T}} b\ b\ b\ a$	$\{4\}$
$\{2, 4\}$	$a\ a\ b\ b \rightsquigarrow_{\mathcal{T}} b\ b\ b\ b$	$\{4\}$
$\{2, 4\}$	$a\ a\ a\ a \rightsquigarrow_{\mathcal{T}} a\ b\ b\ a$	$\{4\}$
$\{2, 4\}$	$b\ a\ a\ a \rightsquigarrow_{\mathcal{T}} b\ b\ b\ a$	$\{4\}$
$\{2, 4\}$	$a\ a\ a\ b \rightsquigarrow_{\mathcal{T}} a\ b\ b\ b$	$\{4\}$
$\{2, 4\}$	$b\ a\ a\ b \rightsquigarrow_{\mathcal{T}} b\ b\ b\ b$	$\{4\}$
$\{2, 4\}$	$a\ a\ a\ a \rightsquigarrow_{\mathcal{T}} a\ a\ b\ b$	$\{2, 4\}$
$\{4\}$	$a\ b\ a\ a \rightsquigarrow_{\mathcal{T}} a\ b\ b\ b$	$\{4\}$
$\{2, 4\}$	$b\ a\ a\ a \rightsquigarrow_{\mathcal{T}} b\ a\ b\ b$	$\{2, 4\}$
$\{4\}$	$b\ b\ a\ a \rightsquigarrow_{\mathcal{T}} b\ b\ b\ b$	$\{4\}$
$\{2, 4\}$	$a\ a\ a\ a \rightsquigarrow_{\mathcal{T}} b\ a\ a\ b$	$\{2, 4\}$
$\{2, 4\}$	$a\ a\ b\ a \rightsquigarrow_{\mathcal{T}} b\ a\ b\ b$	$\{2, 4\}$
$\{4\}$	$a\ b\ a\ a \rightsquigarrow_{\mathcal{T}} b\ b\ a\ b$	$\{4\}$
$\{4\}$	$a\ b\ b\ a \rightsquigarrow_{\mathcal{T}} b\ b\ b\ b$	$\{4\}$

One letter of this statement is Σ. This counteracts the occurrence of the not compatible pairs $\emptyset\ \{a\}$ and $\{a\}\ \emptyset$. Specifically, any realization of the pattern at the position of these pairs yields a transition where the origin does not satisfy this statement. This is because the pattern itself contributes one index to the set *hit* and the letter Σ contributes another. Roughly speaking, the index of the letter Σ is part of every set *hit* and, for this reason, although there are patterns that are not compatible with the pair, their realizations at the position of the pair does not render this statement non-inductive because the patterns necessarily also contribute a index to the set *hit*. We call pairs *hitting* if every pattern is either compatible with it or contributes at least one index to the set *hit*.

Definition 2.24: *Hitting and missing pairs.*

Let P be the patterns of a ring $\mathcal{R}$. We call $\langle A_1, A_2\rangle$ *hitting* for $\begin{bmatrix} v_1 \\ u_1 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix} \in P$ if $|\{i \in \{1,2\} \mid v_i \in A_i\}| > 0$ and *missing* for $\begin{bmatrix} v_1 \\ u_1 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix} \in P$ if $|\{i \in \{1,2\} \mid v_i \in A_i\}| = 0$. Moreover, we call the pair $\langle A_1, A_2\rangle$ *hitting* for P if $\langle A_1, A_2\rangle$ is *compatible with or hitting for* $\begin{bmatrix} v_1 \\ u_1 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix}$ for $\mathcal{V}_{\mathit{flow}}$ for all $\begin{bmatrix} v_1 \\ u_1 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix} \in P$. Analogously, we call the pair $\langle A_1, A_2\rangle$ *missing* for P if $\langle A_1, A_2\rangle$ is *compatible with or missing for* $\begin{bmatrix} v_1 \\ u_1 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix}$ for $\mathcal{V}_{\mathit{flow}}$ for all $\begin{bmatrix} v_1 \\ u_1 \end{bmatrix} \begin{bmatrix} v_2 \\ u_2 \end{bmatrix} \in P$.

Lemma 2.28. *Let $\mathcal{R}$ be any ring with patterns P. The set*

$$\left\{ I_1 \ \ldots \ I_n \in \left(2^\Sigma\right)^* \;\middle|\; \begin{array}{l} I_i = \Sigma \text{ for exactly one } 1 \le i \le n, \\ \langle I_j, I_{j\oplus 1}\rangle \text{ are hitting for all } 1 \le j \le n \text{ with } j \notin \{i \ominus 1, i\}, \\ \langle I_{i\ominus 1}, I_i\rangle \text{ are compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}}, \text{ and} \\ \langle I_i, I_{i\oplus 1}\rangle \text{ are compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

only contains inductive statements for $\mathcal{V}_{\mathit{flow}}$.

Proof. Pick any $I_1 \ \ldots \ I_n$ from this set. Consider the realization $u_1 \ \ldots \ u_n \rightsquigarrow_{\mathcal{T}} v_1 \ \ldots \ v_n$ of $\begin{bmatrix} v_j \\ u_j \end{bmatrix} \begin{bmatrix} v_{j\oplus 1} \\ u_{j\oplus 1} \end{bmatrix}$ at j such that $u_1 \ \ldots \ u_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_n$. There is exactly one $1 \le i \le n$ such that $u_i \in I_i$. Hence, $I_i = \Sigma$. If $j \in \{i \ominus 1, i\}$ then $v_1 \ \ldots \ v_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_n$ since $\langle I_{i\ominus 1}, I_i\rangle$ and $\langle I_i, I_{i\oplus 1}\rangle$ are compatible with $\begin{bmatrix} v_j \\ u_j \end{bmatrix} \begin{bmatrix} v_{j\oplus 1} \\ u_{j\oplus 1} \end{bmatrix}$. Otherwise $u_j \notin I_j$ and $u_{j\oplus 1} \notin I_{j\oplus 1}$ and, therefore, $v_j \notin I_j$ and $v_{j\oplus 1} \notin I_{j\oplus 1}$ because $\langle I_j, I_{j\oplus 1}\rangle$ must be compatible with $\begin{bmatrix} v_j \\ u_j \end{bmatrix} \begin{bmatrix} v_{j\oplus 1} \\ u_{j\oplus 1} \end{bmatrix}$. □

There is an analogous special case for a missing pair which is only surrounded by $\emptyset$:

Lemma 2.29. *Let $\mathcal{R}$ be any ring with patterns P. The set*

$$\left\{ I \in \mathcal{L}(\emptyset^+\ A\ B\ \emptyset^* | \emptyset^*\ A\ B\ \emptyset^+ | B\ \emptyset^+\ A) \;\middle|\; \begin{array}{l} \langle A, B\rangle \text{ is missing,} \\ \langle A, \emptyset\rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle \emptyset, B\rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$
$$\cup \{A\ B | \langle A, B\rangle \text{ and } \langle B, A\rangle \text{ are missing}\}$$

only contains inductive statements for $\mathcal{V}_{\mathit{flow}}$.

Proof. Pick any $I_1\ \dots\ I_n$ from this set. Consider the realization $u_1\ \dots\ u_n \rightsquigarrow_{\mathcal{T}} v_1\ \dots\ v_n$ of $\begin{bmatrix} v_j \\ u_j \end{bmatrix} \begin{bmatrix} v_{j\oplus 1} \\ u_{j\oplus 1} \end{bmatrix}$ at j such that $u_1\ \dots\ u_n \models_{\mathcal{V}_{\mathit{flow}}} I_1\ \dots\ I_n$. If $I_j = \emptyset$ and $I_{j\oplus 1} = \emptyset$, then $v_j \notin I_j$ and $v_{j\oplus 1} \notin I_{j\oplus 1}$ and, thus, $v_1\ \dots\ v_n \models_{\mathcal{V}_{\mathit{flow}}} I_1\ \dots\ I_n$ because $u_k = v_k$ for all $k \notin \{j, j \oplus 1\}$.

Otherwise, $I_j \neq \emptyset$ or $I_{j\oplus 1} \neq \emptyset$. There is exactly one $1 \leq i \leq n$ such that $u_i \in I_i$. Consequently, I_i is either A or B. Therefore, $\begin{bmatrix} v_j \\ u_j \end{bmatrix} \begin{bmatrix} v_{j\oplus 1} \\ u_{j\oplus 1} \end{bmatrix}$ is compatible with $\langle I_j, I_{j\oplus 1}\rangle$ because $\langle I_j, I_{j\oplus 1}\rangle$ cannot be a missing pair for $\begin{bmatrix} v_j \\ u_j \end{bmatrix} \begin{bmatrix} v_{j\oplus 1} \\ u_{j\oplus 1} \end{bmatrix}$ since, then, $\langle I_j, I_{j\oplus 1}\rangle = \langle A, B\rangle$ or $\langle I_j, I_{j\oplus 1}\rangle = \langle B, A\rangle$ but either $v_j \in I_j$ or $v_{j\oplus 1} \in I_{j\oplus 1}$. Again, $v_1\ \dots\ v_n \models_{\mathcal{V}_{\mathit{flow}}} I_1\ \dots\ I_n$ follows because the size of the *hit* set does not change. ☐

Based on this, we can characterize $\text{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$.

Lemma 2.30. *Let P be the patterns of a ring $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$. Then, the set of all* *inductive statements for $\mathcal{V}_{\mathit{flow}}$ is*

$$\begin{aligned}
&\{\varepsilon\} \cup 2^{\Sigma} \\
&\cup \left\{ I \in \left(2^{\Sigma}\right)^{*} \mid \textit{all } \langle A, B \rangle \in \operatorname{adj}(I) \textit{ are compatible with } P \right\} \\
&\cup \left\{ I_1 \, \dots \, I_n \in \left(2^{\Sigma}\right)^{*} \,\middle|\, \begin{array}{l} I_i = \Sigma \textit{ for exactly one } 1 \leq i \leq n, \\ \langle I_j, I_{j \oplus 1} \rangle \textit{ are hitting for all } 1 \leq j \leq n \textit{ with } j \notin \{i \ominus 1, i\}, \\ \langle I_{i \ominus 1}, I_i \rangle \textit{ are compatible with } P \textit{ for } \mathcal{V}_{\mathit{flow}}, \textit{ and} \\ \langle I_i, I_{i \oplus 1} \rangle \textit{ are compatible with } P \textit{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\} \\
&\cup \left\{ I \in \mathcal{L}(\emptyset^{+} \; A \; B \; \emptyset^{*} | \emptyset^{*} \; A \; B \; \emptyset^{+} | B \; \emptyset^{+} \; A) \,\middle|\, \begin{array}{l} \langle A, B \rangle \textit{ is missing,} \\ \langle A, \emptyset \rangle \textit{ is compatible with } P \textit{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle \emptyset, B \rangle \textit{ is compatible with } P \textit{ for } \mathcal{V}_{\mathit{flow}}, \end{array} \right\} \\
&\cup \{ A \; B | \langle A, B \rangle \textit{ and } \langle B, A \rangle \textit{ are missing} \} \\
&\cup \left(2^{\Sigma}\right)^{*} \; \Sigma \; \left(2^{\Sigma}\right)^{*} \; \Sigma \; \left(2^{\Sigma}\right)^{*}.
\end{aligned} \tag{2.4}$$

Proof.

"⊆" Observe that

$$\{\varepsilon\} \cup 2^{\Sigma} \subseteq \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R}) \text{ and } \left(2^{\Sigma}\right)^{*} \; \Sigma \; \left(2^{\Sigma}\right)^{*} \; \Sigma \; \left(2^{\Sigma}\right)^{*} \subseteq \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R}).$$

The latter follows immediately from the observation that, for any

$$I_1 \, \dots \, I_n \in \left(2^{\Sigma}\right)^{*} \; \Sigma \; \left(2^{\Sigma}\right)^{*} \; \Sigma \; \left(2^{\Sigma}\right)^{*},$$

there is no $w \in \Sigma^n$ such that $w \models_{\mathcal{V}_{\mathit{flow}}} I_1 \, \dots \, I_n$.

Lemma 2.28 shows that

$$\left\{ I_1 \, \dots \, I_n \in \left(2^{\Sigma}\right)^{*} \,\middle|\, \begin{array}{l} I_i = \Sigma \text{ for exactly one } 1 \leq i \leq n, \\ \langle I_j, I_{j \oplus 1} \rangle \text{ are hitting for all } 1 \leq j \leq n \text{ with } j \notin \{i \ominus 1, i\}, \\ \langle I_{i \ominus 1}, I_i \rangle \text{ are compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}}, \text{ and} \\ \langle I_i, I_{i \oplus 1} \rangle \text{ are compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

only contains words from $\mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$.

Similarly, Lemma 2.29 shows that

$$\left\{ I \in \mathcal{L}(\emptyset^+ \; A \; B \; \emptyset^* | \emptyset^* \; A \; B \; \emptyset^+ | B \; \emptyset^+ \; A) \;\middle|\; \begin{array}{l} \langle A, B\rangle \text{ is missing,} \\ \langle A, \emptyset\rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle \emptyset, B\rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}}, \end{array} \right\}$$

$$\cup \left\{ A \; B | \langle A, B\rangle \text{ and } \langle B, A\rangle \text{ are missing} \right\}$$

only contains words from $\mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$.

Pick now any $I_1 \; \ldots \; I_n$ from

$$\left\{ I \in \left(2^{\Sigma}\right)^* \mid \text{all } \langle A, B\rangle \in \mathrm{adj}(I) \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \right\}$$

and any realization $u_1 \; \ldots \; u_n \rightsquigarrow_{\mathcal{T}} v_1 \; \ldots \; v_n$ of $\left[\begin{smallmatrix} v_j \\ u_j \end{smallmatrix}\right] \left[\begin{smallmatrix} v_{j\oplus 1} \\ u_{j\oplus 1} \end{smallmatrix}\right]$ at j such that $u_1 \; \ldots \; u_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \; \ldots \; I_n$. This means that there is exactly one $1 \leq i \leq n$ such that $u_i \in I_i$. Distinguish two cases:

$i \notin \{j \ominus 1, j\}$: Thus, $u_j \notin I_j$ and $u_{j\oplus 1} \notin I_{j\oplus 1}$ and, because $I_j \; I_{j\oplus 1}$ is compatible with $\left[\begin{smallmatrix} v_j \\ u_j \end{smallmatrix}\right] \left[\begin{smallmatrix} v_{j\oplus 1} \\ u_{j\oplus 1} \end{smallmatrix}\right]$ and $v_j \notin I_j$ and $v_{j\oplus 1} \notin I_{j\oplus 1}$. $v_1 \; \ldots \; v_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \; \ldots \; I_n$ follows by $v_i = u_i \in I_i$ and $v_k = u_k$ for all $1 \leq k \leq n$ such that $k \notin \{j, j \oplus 1\}$.

$i \in \{j \ominus 1, j\}$: Now either $v_j \in I_j$ and $v_{j\oplus 1} \in I_{j\oplus 1}$ (but not both), because $I_j \; I_{j\oplus 1}$ is compatible with $\left[\begin{smallmatrix} v_j \\ u_j \end{smallmatrix}\right] \left[\begin{smallmatrix} v_{j\oplus 1} \\ u_{j\oplus 1} \end{smallmatrix}\right]$. Again, because $v_k = u_k$ for all $1 \leq k \leq n$ such that $k \notin \{j, j \oplus 1\}$, it follows that $v_1 \; \ldots \; v_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \; \ldots \; I_n$.

"$\supseteq$" It remains to show that (2.4) contains all words of $\mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$. For the sake of contradiction, assume there is some $I_1 \; \ldots \; I_n \in \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$ that is not part of the set (2.4). Again, distinguish two cases:

With Σ: There is exactly one $1 \leq j \leq n$ such that $I_j = \Sigma$. Since $I_1 \; \ldots \; I_n$ must not be part of

$$\left\{ I_1 \; \ldots \; I_n \in \left(2^{\Sigma}\right)^* \;\middle|\; \begin{array}{l} I_i = \Sigma \text{ for exactly one } 1 \leq i \leq n, \\ \langle I_j, I_{j\oplus 1}\rangle \text{ are hitting for all } 1 \leq j \leq n \text{ with } j \notin \{i \ominus 1, i\}, \\ \langle I_{i\ominus 1}, I_i\rangle \text{ are compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}}, \text{ and} \\ \langle I_i, I_{i\oplus 1}\rangle \text{ are compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

distinguish two more cases:

If $\langle I_j, I_{j\oplus 1}\rangle$ ($\langle I_{j\ominus 1}, I_j\rangle$) is not compatible with $\begin{bmatrix} u_j \\ v_j \end{bmatrix} \begin{bmatrix} u_j \oplus 1 \\ v_{j\oplus 1} \end{bmatrix}$ ($\begin{bmatrix} u_{j\ominus 1} \\ v_{j\ominus 1} \end{bmatrix} \begin{bmatrix} u_j \\ v_j \end{bmatrix}$), then one can pick, for any $1 \leq k \leq n$ such that $k \notin \{j, j \oplus 1\}$ ($k \notin \{j, j \ominus 1\}$), some $u_k \notin \Sigma \setminus I_k$. This gives the transition $u_1 \ \dots \ u_n \rightsquigarrow_{\mathcal{T}} v_1 \ \dots \ v_n$ such that $u_1 \ \dots \ u_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_n$ and $v_1 \ \dots \ v_n \not\models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_n$ because $v_j \in I_j$ and $v_{j\oplus 1} \in I_{j\oplus 1}$ ($v_j \in I_j$ and $v_{j\ominus 1} \in I_{j\ominus 1}$).

Otherwise, there is $1 \leq k \leq n$ with $k \notin \{j \ominus 1, j\}$ such that $\langle I_k, I_{k\oplus 1}\rangle$ is not hitting. Thus, there is a pattern $\begin{bmatrix} u_k \\ v_k \end{bmatrix} \begin{bmatrix} u_{k\oplus 1} \\ v_{k\oplus 1} \end{bmatrix}$ such that $u_k \notin I_k$ and $u_{k\oplus 1} \notin I_{k\oplus 1}$ but either $v_k \in I_k$ or $v_{k\oplus 1} \in I_{k\oplus 1}$ (or both). Choose an arbitrary u_j and, as before, for any $1 \leq x \leq n$ such that $x \notin \{i, k \oplus 1, k\}$ some $u_x \notin \Sigma \setminus I_x$. This gives $u_1 \ \dots \ u_n \rightsquigarrow_{\mathcal{T}} v_1 \ \dots \ v_n$ such that $u_1 \ \dots \ u_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_n$ and $v_1 \ \dots \ v_n \not\models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_n$.

Without Σ: Since $I_1 \ \dots \ I_n$ must not be part of

$$\left\{ I \in \left(2^{\Sigma}\right)^* \mid \text{all } \ \langle A, B\rangle \in \operatorname{adj}(I) \text{ are compatible with } P \right\},$$

there is $\langle I_i, I_{i\oplus 1}\rangle$ that is not compatible with P. Thus, there is $\begin{bmatrix} u_i \\ v_i \end{bmatrix} \begin{bmatrix} u_{i\oplus 1} \\ v_{i\oplus 1} \end{bmatrix} \in P$ such that $\langle I_i, I_{i\oplus 1}\rangle$ is not compatible with $\begin{bmatrix} u_i \\ v_i \end{bmatrix} \begin{bmatrix} u_{i\oplus 1} \\ v_{i\oplus 1} \end{bmatrix}$. Here, distinguish, again, two more cases.

If $|\{j \; \{i, i \oplus 1\} \mid u_j \in I_j\}| = 1$ but $|\{j \; \{i, i \oplus 1\} \mid v_j \in I_j\}| \neq 1$, then, as before, one can construct a transition $u_1 \ \dots \ u_n \rightsquigarrow_{\mathcal{T}} v_1 \ \dots \ v_n$ as an instance of this pattern which disproves that $I_1 \ \dots \ I_n$ is an inductive statement for $\mathcal{V}_{\mathit{flow}}$. Specifically, one can choose $u_x = v_x \notin I_x$ for all $x \in \{1, \dots, n\} \setminus \{i, i \oplus 1\}$ because there is no letter that is Σ.

Otherwise, $\langle I_i, I_{i\oplus 1}\rangle$ is missing. Thus, because the pattern $\begin{bmatrix} u_i \\ v_i \end{bmatrix} \begin{bmatrix} u_{i\oplus 1} \\ v_{i\oplus 1} \end{bmatrix}$ is not compatible with this pair, $|\{j \in \{i, i \oplus 1\} \mid u_j \in I_j\}| = 0$ and $|\{j \in \{i, i \oplus 1\} \mid v_j \in I_j\}| > 0$. However, $I_1 \ \dots \ I_n$ must not be part of

$$\left\{ I \in \mathcal{L}(\emptyset^+ \ A \ B \ \emptyset^* | \emptyset^* \ A \ B \ \emptyset^+ | B \ \emptyset^+ \ A) \left| \begin{array}{l} \langle A, B\rangle \ \text{ is missing,} \\ \langle A, \emptyset\rangle \ \text{ is compatible with } P \text{ for } \ \mathcal{V}_{\mathit{flow}}, \\ \langle \emptyset, B\rangle \ \text{ is compatible with } P \text{ for } \ \mathcal{V}_{\mathit{flow}}, \end{array} \right. \right\}$$

$$\cup \left\{ A \ B | \langle A, B\rangle \ \text{ and } \ \langle B, A\rangle \ \text{ are missing} \right\}.$$

Thus, there is at least one $j \in \{1, \ldots, n\} \setminus \{i, i \oplus 1\}$ such that $I_j \neq \emptyset$. Choose $u_j = v_j \in I_j$ and $u_k = v_k \notin I_k$ for all $1 \leq k \leq n$ with $k \notin \{i, i \oplus 1, j\}$. This yields a transition $u_1 \, \ldots \, u_n \rightsquigarrow_{\mathcal{T}} v_1 \, \ldots \, v_n$ such that $u_1 \, \ldots \, u_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \, \ldots \, I_n$ and $v_1 \, \ldots \, v_n \not\models_{\mathcal{V}_{\mathit{flow}}} I_1 \, \ldots \, I_n$. Specifically, $|\{m \in \{i, i \oplus 1\} \mid u_m \in I_m\}| > 0$ and $v_j \in I_j$.

$\square$

We can use this characterization to construct an automaton that recognizes all inductive statements for $\mathcal{V}_{\mathit{flow}}$. Conceptually, the automaton stores the first letter of the statement and the last letter that was read. Based on this information, all of the different sets from Lemma 2.30 can be recognized with a constant amount of additional information.

Corollary 2.7. *One can effectively construct a DFA that recognizes* $\mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$ *with* $\mathcal{O}(\left|2^{\Sigma}\right|^2)$ *states.*

Proof. Consider $Q = 2^{\Sigma} \times \left(2^{\Sigma} \cup \{\square\}\right) \cup \{q_0\}$ and δ which is defined as follows:

- $\delta(q_0, I) = \langle I, \square \rangle$,
- $\delta(\langle I, \square \rangle, A) = \langle I, A \rangle$, and
- $\delta(\langle I, B \rangle, A) = \langle I, A \rangle$.

Consider the different sets from Lemma 2.30:

- For the statements of $\left\{I \in \left(2^{\Sigma}\right)^* \mid \text{all } \langle A, B \rangle \in \mathrm{adj}(I) \text{ are compatible with } P\right\}$, fix $Q_\alpha = \{\top, \bot\}$ and $\alpha\colon (Q \times Q_\alpha) \times 2^{\Sigma} \to Q_\alpha$ with
  - $\alpha(\langle q_0, \top \rangle, A) = \top$
  - $\alpha(\langle \langle B, \square \rangle, \top \rangle, A) = \begin{cases} \top & \text{if } \langle B, A \rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \\ \bot & \text{otherwise} \end{cases}$
  - $\alpha(\langle \langle S, B \rangle, \top \rangle, A) = \begin{cases} \top & \text{if } \langle B, A \rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \\ \bot & \text{otherwise} \end{cases}$
  - $\alpha(\langle q, \bot \rangle, A) = \bot$.

Additionally, set $F_\alpha = \left\{ \langle q, \top \rangle \in Q \times \{\top\} \,\middle|\, \begin{array}{l} q = \langle A, B \rangle \text{ and } \langle B, A \rangle \\ \text{is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$ and $q_0^\alpha = \top$.

- For the statements of

$$\left\{ I_1 \;\ldots\; I_n \in \left(2^\Sigma\right)^* \,\middle|\, \begin{array}{l} I_i = \Sigma \text{ for exactly one } 1 \le i \le n, \\ \langle I_j, I_{j\oplus 1} \rangle \text{ are hitting for all } 1 \le j \le n \text{ with } j \notin \{i \ominus 1, i\}, \\ \langle I_{i\ominus 1}, I_i \rangle \text{ are compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}}, \text{ and} \\ \langle I_i, I_{i\oplus 1} \rangle \text{ are compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\},$$

  fix $Q_\beta = \{0, 1, \bot\}$ and $\beta\colon (Q \times Q_\beta) \times 2^\Sigma \to Q_\beta$ with

  - $\beta(\langle q_0, 0 \rangle, A) = \begin{cases} 0 & \text{if } A \ne \Sigma \\ 1 & \text{otherwise} \end{cases}$
  - $\beta(\langle \langle B, \Box \rangle, 0 \rangle, A) = 0$ if $A \ne \Sigma$ and $B \ne \Sigma$ and $\langle B, A \rangle$ hitting
  - $\beta(\langle \langle \Sigma, \Box \rangle, 1 \rangle, A) = 1$ if $A \ne \Sigma$ and $\langle \Sigma, A \rangle$ is compatible with P for $\mathcal{V}_{\mathit{flow}}$
  - $\beta(\langle \langle B, \Box \rangle, 0 \rangle, \Sigma) = 1$ if $B \ne \Sigma$ and $\langle B, \Sigma \rangle$ is compatible with P for $\mathcal{V}_{\mathit{flow}}$
  - $\beta(\langle \langle S, B \rangle, 0 \rangle, A) = 0$ if $A \ne \Sigma$ and $B \ne \Sigma$ and $\langle B, A \rangle$ hitting
  - $\beta(\langle \langle S, B \rangle, 0 \rangle, \Sigma) = 0$ if $B \ne \Sigma$ and $\langle B, \Sigma \rangle$ is compatible with P for $\mathcal{V}_{\mathit{flow}}$
  - $\beta(\langle \langle S, B \rangle, 1 \rangle, A) = 1$ if $A \ne \Sigma$ and $B \ne \Sigma$ and $\langle B, A \rangle$ hitting
  - $\beta(\langle \langle S, B \rangle, 1 \rangle, \Sigma) = 1$ if $B \ne \Sigma$ and $\langle B, \Sigma \rangle$ is compatible with P for $\mathcal{V}_{\mathit{flow}}$
  - $\beta(\langle \langle S, \Sigma \rangle, 1 \rangle, B) = 1$ if $B \ne \Sigma$ and $\langle \Sigma, B \rangle$ is compatible with P for $\mathcal{V}_{\mathit{flow}}$
  - $\beta(\langle q, q_\beta \rangle, A) = \bot$ in all other cases

  Additionally, set $q_0^\beta = 0$ and

$$F_\beta = \left\{ \langle q, 1 \rangle \in Q \times \{1\} \,\middle|\, \begin{array}{r} q = \langle \Sigma, A \rangle, \langle A, \Sigma \rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \\ \text{or } q = \langle B, \Sigma \rangle, \langle \Sigma, B \rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}} \\ \text{or } q = \langle A, B \rangle, B \ne \Sigma, A \ne \Sigma, \langle B, A \rangle \text{ is hitting} \end{array} \right\}.$$

- For the statements of

$$\left\{ I \in \mathcal{L}(\emptyset^+\ A\ B\ \emptyset^* | \emptyset^*\ A\ B\ \emptyset^+ | B\ \emptyset^+\ A) \middle| \begin{array}{l} \langle A, B\rangle \text{ is missing,} \\ \langle A, \emptyset\rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle \emptyset, B\rangle \text{ is compatible with } P \text{ for } \mathcal{V}_{\mathit{flow}}, \end{array} \right\}$$

$$\cup \left\{A\ B | \langle A, B\rangle \text{ and } \langle B, A\rangle \text{ are missing}\right\},$$

  define adequate[10] Q_γ, γ, F_γ, and q_0^γ.

- For the statements of

$$\left(2^\Sigma\right)^*\ \Sigma\ \left(2^\Sigma\right)^*\ \Sigma\ \left(2^\Sigma\right)^*,$$

  define adequate Q_ϵ, ϵ, F_ϵ, and q_0^ϵ.

Then, the DFA is

$$\left\langle Q \times Q_\alpha \times Q_\beta \times Q_\gamma \times Q_\epsilon, q_0', 2^\Sigma, \delta', F \right\rangle$$

where $q_0' = \left\langle q_0, q_0^\alpha, q_0^\beta, q_0^\gamma, q_0^\epsilon \right\rangle$,

$$\begin{aligned} &\delta'(\langle q, q_\alpha, q_\beta, q_\gamma, q_\epsilon\rangle, A) \\ &= \langle \delta(q, A), \alpha((q, q_\alpha), A), \beta((q, q_\beta), A), \gamma((q, q_\gamma), A), \epsilon((q, q_\epsilon), A)\rangle, \end{aligned}$$

and

$$F' = \left\{ \langle q, q_\alpha, q_\beta, q_\gamma, q_\epsilon\rangle \in Q \times Q_\alpha \times Q_\beta \times Q_\gamma \times Q_\epsilon \middle| \begin{array}{l} \quad q \in \{q_0\} \cup \left(2^\Sigma \times \{\Box\}\right) \\ \text{or } \langle q, q_\alpha\rangle \in F_\alpha \\ \text{or } \langle q, q_\beta\rangle \in F_\beta \\ \text{or } \langle q, q_\gamma\rangle \in F_\gamma \\ \text{or } \langle q, q_\epsilon\rangle \in F_\epsilon \end{array} \right\}.$$

The correctness of the construction is an immediate consequence of Lemma 2.30. □

---

[10] It is only necessary to observe here that a constant amount of information (in addition to the information from Q) suffices to recognize this language – for instance, whether the word started with a non-empty sequence of $\emptyset$, whether one encountered two (adjacent) letters different than $\emptyset$, and so on.

The bow topology

We consider a slight variation of the ring topology – *bows*. While a ring is a continuous, seamless band of agents, a bow allows for a seam: one single index that is allowed to interact differently with its neighbors than all the others.

Example 2.2 without the invariant of a single token in every transition is a bow. The reason for this is that the first agent does not accept a token from the last agent. This behavior distinguishes the first agent, which, thus, becomes the seam of this bow. We (arbitrarily) chose the first agent as the seam for all bows.

Roughly speaking, a bow is specified with three sets of patterns: P_L, P_R, and P_M. Here, P_R captures the interactions of the first index with its *right* neighbor, P_L models the interaction with the last agent (which is on the *left* of the first agent), and the patterns in P_M can be realized for all other positions.

Definition 2.25: *Bow topology.*

We call any $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ a *bow* if there are finite sets $P_L, P_R, P_M \subseteq (\Sigma \times \Sigma) \times (\Sigma \times \Sigma)$ such that

$$\bigcup_{\left\langle \begin{bmatrix} v \\ v' \end{bmatrix}, \begin{bmatrix} u \\ u' \end{bmatrix} \right\rangle \in P_M} I \begin{bmatrix} v \\ v' \end{bmatrix} \begin{bmatrix} u \\ u' \end{bmatrix} I^* \cup \bigcup_{\left\langle \begin{bmatrix} v \\ v' \end{bmatrix}, \begin{bmatrix} u \\ u' \end{bmatrix} \right\rangle \in P_L} \begin{bmatrix} u \\ u' \end{bmatrix} I^* \begin{bmatrix} v \\ v' \end{bmatrix} \cup \bigcup_{\left\langle \begin{bmatrix} v \\ v' \end{bmatrix}, \begin{bmatrix} u \\ u' \end{bmatrix} \right\rangle \in P_R} \begin{bmatrix} v \\ v' \end{bmatrix} \begin{bmatrix} u \\ u' \end{bmatrix} I^*$$

where $I = \left\{ \begin{bmatrix} v \\ v \end{bmatrix} : v \in \Sigma \right\}$ is the set of all transitions.

Example 2.34: *Token passing as a bow.*

Recall Example 2.2 in the variant where the transitions do not enforce a single token. Remember that the initial language of this system is $t\ n^*$ and the transitions of the system are $\left(\begin{bmatrix} t \\ t \end{bmatrix} \middle| \begin{bmatrix} n \\ n \end{bmatrix} \right)^* \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \left(\begin{bmatrix} t \\ t \end{bmatrix} \middle| \begin{bmatrix} n \\ n \end{bmatrix} \right)^*$. Thus, this system is a bow with

$$P_M = P_R = \left\{ \left\langle \begin{bmatrix} t \\ n \end{bmatrix}, \begin{bmatrix} n \\ t \end{bmatrix} \right\rangle \right\} \text{ and } P_L = \emptyset.$$

As before for rings, we can now capture the set of all inductive statements for the concrete interpretations in terms of neighboring letters in the statement. Due to the special structure of a bow, the first agent and its neighbors are treated separately.

Regardless, the same arguments as for rings apply to bows. Therefore, we omit proofs for the following results since these are straightforward adaptations of the proofs before.

Non-inductive statements for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ in bows

Definition 2.26: *Non-inductive pairs in bows.*

Let P_L, P_R, P_M be the patterns of the bow $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and $\mathcal{V}$ some interpretation. We call a pair $\langle A, B \rangle \in 2^\Sigma \times 2^\Sigma$ a

- *non-inductive left pair* for $\mathcal{V}$ if there exists $\left[\begin{smallmatrix} v \\ v' \end{smallmatrix}\right] \left[\begin{smallmatrix} u \\ u' \end{smallmatrix}\right] \in P_L$ such that $v\ u \models_\mathcal{V} A\ B$ and $v'\ u' \not\models_\mathcal{V} A\ B$,
- *non-inductive right pair* for $\mathcal{V}$ if there exists $\left[\begin{smallmatrix} v \\ v' \end{smallmatrix}\right] \left[\begin{smallmatrix} u \\ u' \end{smallmatrix}\right] \in P_R$ such that $v\ u \models_\mathcal{V} A\ B$ and $v'\ u' \not\models_\mathcal{V} A\ B$, and
- *non-inductive middle pair* for $\mathcal{V}$ if there exists $\left[\begin{smallmatrix} v \\ v' \end{smallmatrix}\right] \left[\begin{smallmatrix} u \\ u' \end{smallmatrix}\right] \in P_M$ such that $v\ u \models_\mathcal{V} A\ B$ and $v'\ u' \not\models_\mathcal{V} A\ B$.

To formulate the results we introduce slight variations of the concept of adjacent indices. Namely, we distinguish the pair that is the first and second letter as well as the last and first letter (and all *other* adjacent pairs as well). This leads to the definitions $\mathrm{adj}_L(I_1\ \dots\ I_n) = \langle I_n, I_1 \rangle$, $\mathrm{adj}_M(I_1\ \dots\ I_n) = \{\langle I_2, I_3 \rangle, \dots, \langle I_{n-1}, I_n \rangle\}$, and $\mathrm{adj}_R(I_1\ \dots\ I_n) = \langle I_1, I_2 \rangle$. In the same fashion as for rings before, we obtain now the following results:

Lemma 2.31. *Let P_L, P_R, P_M be the patterns of the bow $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and $\mathcal{V}$ be either $\mathcal{V}_{trap}$ or $\mathcal{V}_{siphon}$. Then, the set of non-inductive statements for $\mathcal{V}$ is*

$$\begin{aligned} & \left\{ I \in \left(2^\Sigma \setminus \{\Sigma\}\right)^* \,\middle|\, \begin{array}{l} \textit{there is } \langle A, B \rangle \in \mathrm{adj}_M(I) \\ \textit{that is a non-inductive middle pair for } \mathcal{V} \end{array} \right\} \\ \cup & \left\{ I \in \left(2^\Sigma \setminus \{\Sigma\}\right)^* \middle| \mathrm{adj}_L(I) \textit{is a non-inductive left pair for } \mathcal{V} \right\} \\ \cup & \left\{ I \in \left(2^\Sigma \setminus \{\Sigma\}\right)^* \middle| \mathrm{adj}_R(I) \textit{is a non-inductive right pair for } \mathcal{V} \right\}. \end{aligned} \tag{2.5}$$

Inductive statements for $\mathcal{V}_{\mathit{flow}}$ in bows

To present a result akin to Lemma 2.30, we introduce some additional notation. Specifically, we consider the statements that contain the letter Σ. For these, we distinguish four individual sets of statements – all of which contain the letter Σ at different positions:

First letter:

$$F = \left\{ I_1 \; \ldots \; I_n \in \left(2^{\Sigma}\right)^* \;\middle|\; \begin{array}{l} I_i = \Sigma \text{ if and only if } i = 1, \\ \langle I_j, I_{j+1} \rangle \text{ are hitting for } P_M \text{ for all } 1 < j < n, \\ \langle I_n, I_1 \rangle \text{ is compatible with } P_L \text{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle I_1, I_2 \rangle \text{ is compatible with } P_R \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

Second letter:

$$S = \left\{ I_1 \; \ldots \; I_n \in \left(2^{\Sigma}\right)^* \;\middle|\; \begin{array}{l} I_i = \Sigma \text{ if and only if } i = 2, \\ \langle I_j, I_{j+1} \rangle \text{ are hitting for } P_M \text{ for all } 2 < j < n, \\ \langle I_n, I_1 \rangle \text{ is hitting for } P_L, \\ \langle I_1, I_2 \rangle \text{ is compatible with } P_R \text{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle I_2, I_3 \rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

In the middle:

$$M = \left\{ I_1 \; \ldots \; I_n \in \left(2^{\Sigma}\right)^* \;\middle|\; \begin{array}{l} I_i = \Sigma \text{ for exactly one } 2 < i < n, \\ \langle I_j, I_{j+1} \rangle \text{ are hitting for } P_M \text{ for all } 1 < j < i - 2, \\ \langle I_j, I_{j+1} \rangle \text{ are hitting for } P_M \text{ for all } i + 1 < j < n, \\ \langle I_{i-1}, I_i \rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle I_i, I_{i+1} \rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle I_n, I_1 \rangle \text{ is hitting for } P_L, \\ \langle I_1, I_2 \rangle \text{ is hitting for } P_R \end{array} \right\}$$

Last letter:

$$L = \left\{ I_1 \;\ldots\; I_n \in \left(2^{\Sigma}\right)^* \;\middle|\; \begin{array}{l} I_i = \Sigma \text{ if and only if } i = n, \\ \langle I_j, I_{j+1}\rangle \text{ are hitting for } P_M \text{ for all } 1 < j < n-1, \\ \langle I_{n-1}, I_n\rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle I_n, I_1\rangle \text{ is compatible with } P_L \text{ for } \mathcal{V}_{\mathit{flow}}, \\ \langle I_1, I_2\rangle \text{ is hitting for } P_R \end{array} \right\}$$

Similar to the statements of Lemma 2.29, we consider statements that have a single adjacent pair of non-empty letters. For these, we also have to consider the various positions in which this single adjacent pair of non-empty letters can be:

First:

$$F' = \left\{ I \in \mathcal{L}(A\; B\; \emptyset^+) \;\middle|\; \begin{array}{l} \langle A, B\rangle \text{ is missing for } P_R \\ \langle B, \emptyset\rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}} \\ \langle \emptyset, A\rangle \text{ is compatible with } P_L \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

Second:

$$S' = \left\{ I \in \mathcal{L}(\emptyset\; A\; B\; \emptyset^+) \;\middle|\; \begin{array}{l} \langle A, B\rangle \text{ is missing for } P_M \\ \langle B, \emptyset\rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}} \\ \langle \emptyset, A\rangle \text{ is compatible with } P_R \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

Middle:

$$M' = \left\{ I \in \mathcal{L}(\emptyset\; \emptyset^+\; A\; B\; \emptyset^+) \;\middle|\; \begin{array}{l} \langle A, B\rangle \text{ is missing for } P_M \\ \langle B, \emptyset\rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}} \\ \langle \emptyset, A\rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

End:

$$E' = \left\{ I \in \mathcal{L}(\emptyset^+\; A\; B) \;\middle|\; \begin{array}{l} \langle A, B\rangle \text{ is missing for } P_M \\ \langle B, \emptyset\rangle \text{ is compatible with } P_L \text{ for } \mathcal{V}_{\mathit{flow}} \\ \langle \emptyset, A\rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

Broken:

$$B' = \left\{ I \in \mathcal{L}(B\ \emptyset^+\ A) \;\middle|\; \begin{array}{l} \langle A, B\rangle \text{ is missing for } P_L \\ \langle B, \emptyset\rangle \text{ is compatible with } P_R \text{ for } \mathcal{V}_{\mathit{flow}} \\ \langle \emptyset, A\rangle \text{ is compatible with } P_M \text{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\}$$

Alone:

$$A' = \left\{ A\ B \in \left(2^\Sigma\right)^2 \;\middle|\; \begin{array}{l} \langle A, B\rangle \text{ is missing for } P_R \\ \langle B, A\rangle \text{ is missing for } P_L \end{array} \right\}$$

Lemma 2.32. *Let P_L, P_M, P_R be the patterns of a bow $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$. Then, the set of all inductive statements for $\mathcal{V}_{\mathit{flow}}$ is*

$$\begin{aligned} &\{\varepsilon\} \cup 2^\Sigma \\ &\cup \left\{ I \in \left(2^\Sigma\right)^* \;\middle|\; \begin{array}{l} \textit{all } \langle A, B\rangle \in \operatorname{adj}_M(I) \textit{ are compatible with } P_M \textit{ for } \mathcal{V}_{\mathit{flow}} \\ \textit{and } \operatorname{adj}_L(I) \textit{ is compatible with } P_L \textit{ for } \mathcal{V}_{\mathit{flow}} \\ \textit{and } \operatorname{adj}_R(I) \textit{ is compatible with } P_R \textit{ for } \mathcal{V}_{\mathit{flow}} \end{array} \right\} \\ &\cup F \cup S \cup M \cup L \\ &\cup F' \cup S' \cup M' \cup E' \cup B' \cup A' \\ &\cup \left(2^\Sigma\right)^*\ \Sigma\ \left(2^\Sigma\right)^*\ \Sigma\ \left(2^\Sigma\right)^*. \end{aligned} \tag{2.6}$$

By the same construction as for rings, these observations lead to automata that recognize inductive statements for the interpretations $\mathcal{V}_{\mathit{trap}}$, $\mathcal{V}_{\mathit{siphon}}$, and $\mathcal{V}_{\mathit{flow}}$.

Corollary 2.8. *Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$ be a bow. For $\mathcal{V} \in \{\mathcal{V}_{\mathit{trap}}, \mathcal{V}_{\mathit{siphon}}, \mathcal{V}_{\mathit{flow}}\}$, one can effectively construct a DFA with $\mathcal{O}(\left|2^\Sigma\right|^2)$ states for* $\text{Inductive}_{\mathcal{V}}(\mathcal{R})$.

Crowd

A *crowd* represents a collection of anonymous agents. Every transition of this topology consists of two parts: one part is an interaction of a fixed number of agents, and the other part is a collection of state updates for all other agents. One can understand such a transition as a small number of agents proposing some change to which all other agents have to react to. However, if there is one agent that cannot react to this change, it must not be done.

Let us start with a rough introduction of the model and two examples. To this end, we only consider transitions in which a single agent proposes some change $\begin{bmatrix} u \\ v \end{bmatrix}$. All other agents react by performing one change in $\left\{\begin{bmatrix} u_1 \\ v_1 \end{bmatrix}, \dots, \begin{bmatrix} u_k \\ v_k \end{bmatrix}\right\}$. In this way, a pair $\left\langle \begin{bmatrix} u \\ v \end{bmatrix}, \left\{\begin{bmatrix} u_1 \\ v_1 \end{bmatrix}, \dots, \begin{bmatrix} u_k \\ v_k \end{bmatrix}\right\}\right\rangle$ corresponds to the transitions

$$\left(\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \mid \dots \mid \begin{bmatrix} u_k \\ v_k \end{bmatrix}\right)^* \begin{bmatrix} u \\ v \end{bmatrix} \left(\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \mid \dots \mid \begin{bmatrix} u_k \\ v_k \end{bmatrix}\right)^*.$$

Note here that one single agent is changing its state from u to v and all other agents pick any change in $\left\{\begin{bmatrix} u_1 \\ v_1 \end{bmatrix}, \dots, \begin{bmatrix} u_k \\ v_k \end{bmatrix}\right\}$.

We start with two examples. The first one is a basic mutual exclusion algorithm. In this algorithm, we allow an atomic global check of the state of all other agents and, if none of them is in a designated critical section, the checking agent may advance into this critical section. This example illustrates that the reaction of all other agents is more than a broadcast because it can be used to check global conditions: in this case, we use the reaction to check that no agent is currently in the critical section.

Example 2.35: *Mutual Exclusion.*

Consider a set of agents where each agent is in one of two states; either the agent is idling (i) or the agent is in a critical section (c). Initially, all agents are idling and at any moment in time if no other agent is in the critical state an agent may advance into it. Any agent in the critical state might return to the idle state at any moment in time. The regular transition system to model this algorithm is straightforward. Choose $\{i, c\}$ as the alphabet, the initial language to be i^*, and the set of all transitions as the regular language $\begin{bmatrix} i \\ i \end{bmatrix}^* \begin{bmatrix} i \\ c \end{bmatrix} \begin{bmatrix} i \\ i \end{bmatrix}^* \mid \left(\begin{bmatrix} i \\ i \end{bmatrix} \mid \begin{bmatrix} c \\ c \end{bmatrix}\right)^* \begin{bmatrix} c \\ i \end{bmatrix} \left(\begin{bmatrix} i \\ i \end{bmatrix} \mid \begin{bmatrix} c \\ c \end{bmatrix}\right)^*$. This example can be modeled as a crowd. It suffices to consider two pairs to obtain the same transitions:

$$\left\langle \begin{bmatrix} i \\ c \end{bmatrix}, \left\{\begin{bmatrix} i \\ i \end{bmatrix}\right\}\right\rangle \text{ and } \left\langle \begin{bmatrix} c \\ i \end{bmatrix}, \left\{\begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} c \\ c \end{bmatrix}\right\}\right\rangle.$$

Note that in this example the first pair models an atomic global check of some condition on all other agents. The second pattern only sends an informative broadcast of the state change – to which no agent reacts.

Our second example is taken from the world of cache protocols. Namely, we model the MESI protocol as it is described in [Del00b]. In this example, the transitions work as local changes that emit broadcasts: particularly, the reaction of the other agents never restricts a local agent from executing some transition.

Example 2.36: *MESI.*

In the MESI cache coherence protocol, there are four distinct states for each cache cell. These states represent a cell that does not hold any value for the considered memory address (denoted by i for *invalid*), a cell that holds an exclusive value among all cache cells which *agrees* with the value for that memory location in the permanent memory (denoted by e), a cell that holds an exclusive value among all cache cells which *disagrees* with the value of the memory location in the permanent memory (denoted by m), and, finally, a cell that does hold value for some memory location but this value might also be recorded in a different cell (denoted by s). Initially, no cell holds a value of the memory address. Therefore, the initial language is i^*. The protocol allows for five different operations for a single cached address. These operations are two reading operations, distinguished by the fact that the cache cell holds some value for the address or not. In the first case, this means the cell is observed to be in one of the three states e, m, s while no cell changes its value. We call this a *hit* because the value of the address is in the cache. In the latter case, no other cell changes its value but the cell that is read now holds the value of the memory address; thus, it moves into state s. This case is called a *miss* because the value must be read from the memory address and was not present in the cache. The languages that capture these behaviors are

Read Hit: $\left(\begin{bmatrix} i \\ i \end{bmatrix} \mid \begin{bmatrix} e \\ e \end{bmatrix} \mid \begin{bmatrix} m \\ m \end{bmatrix} \mid \begin{bmatrix} s \\ s \end{bmatrix}\right)^* \left(\begin{bmatrix} e \\ e \end{bmatrix} \mid \begin{bmatrix} m \\ m \end{bmatrix} \mid \begin{bmatrix} s \\ s \end{bmatrix}\right) \left(\begin{bmatrix} i \\ i \end{bmatrix} \mid \begin{bmatrix} e \\ e \end{bmatrix} \mid \begin{bmatrix} m \\ m \end{bmatrix} \mid \begin{bmatrix} s \\ s \end{bmatrix}\right)^*$

Read Miss: $\left(\begin{bmatrix} i \\ i \end{bmatrix} \mid \begin{bmatrix} e \\ s \end{bmatrix} \mid \begin{bmatrix} m \\ s \end{bmatrix} \mid \begin{bmatrix} s \\ s \end{bmatrix}\right)^* \begin{bmatrix} i \\ s \end{bmatrix} \left(\begin{bmatrix} i \\ i \end{bmatrix} \mid \begin{bmatrix} e \\ s \end{bmatrix} \mid \begin{bmatrix} m \\ s \end{bmatrix} \mid \begin{bmatrix} s \\ s \end{bmatrix}\right)^*$

Additionally, there are two writing operations, again distinguished by the fact that the cache cell already holds some value for that address or not. Here, the first case moves the cache cell into the state m if it was the only cell to hold the value of the address; that is if the cell was in either state m or e. If, however, the value was shared among multiple cells it is written into the actual memory and all but the addressed cache cell invalidate their value. In consequence, the addressed cell moves into state e while all others move into state i. The same behavior can be observed if the addressed cell does not hold a value for the memory; i.e., is in state i. Again, let us give the languages that model these situations here:

Write Hit:
$$\begin{aligned} & \left(\begin{bmatrix} i \\ i \end{bmatrix} \middle| \begin{bmatrix} e \\ e \end{bmatrix} \middle| \begin{bmatrix} m \\ m \end{bmatrix} \middle| \begin{bmatrix} s \\ s \end{bmatrix}\right)^* \begin{bmatrix} m \\ m \end{bmatrix} \left(\begin{bmatrix} i \\ i \end{bmatrix} \middle| \begin{bmatrix} e \\ e \end{bmatrix} \middle| \begin{bmatrix} m \\ m \end{bmatrix} \middle| \begin{bmatrix} s \\ s \end{bmatrix}\right)^* \\ \cup & \left(\begin{bmatrix} i \\ i \end{bmatrix} \middle| \begin{bmatrix} e \\ e \end{bmatrix} \middle| \begin{bmatrix} m \\ m \end{bmatrix} \middle| \begin{bmatrix} s \\ s \end{bmatrix}\right)^* \begin{bmatrix} e \\ m \end{bmatrix} \left(\begin{bmatrix} i \\ i \end{bmatrix} \middle| \begin{bmatrix} e \\ e \end{bmatrix} \middle| \begin{bmatrix} m \\ m \end{bmatrix} \middle| \begin{bmatrix} s \\ s \end{bmatrix}\right)^* \\ \cup & \left(\begin{bmatrix} i \\ i \end{bmatrix} \middle| \begin{bmatrix} e \\ i \end{bmatrix} \middle| \begin{bmatrix} m \\ i \end{bmatrix} \middle| \begin{bmatrix} s \\ i \end{bmatrix}\right)^* \begin{bmatrix} s \\ e \end{bmatrix} \left(\begin{bmatrix} i \\ i \end{bmatrix} \middle| \begin{bmatrix} e \\ i \end{bmatrix} \middle| \begin{bmatrix} m \\ i \end{bmatrix} \middle| \begin{bmatrix} s \\ i \end{bmatrix}\right)^* \end{aligned}$$

Write Miss: $\left(\begin{bmatrix} i \\ i \end{bmatrix} \middle| \begin{bmatrix} e \\ i \end{bmatrix} \middle| \begin{bmatrix} m \\ i \end{bmatrix} \middle| \begin{bmatrix} s \\ i \end{bmatrix}\right)^* \begin{bmatrix} i \\ e \end{bmatrix} \left(\begin{bmatrix} i \\ i \end{bmatrix} \middle| \begin{bmatrix} e \\ i \end{bmatrix} \middle| \begin{bmatrix} m \\ i \end{bmatrix} \middle| \begin{bmatrix} s \\ i \end{bmatrix}\right)^*$

Finally, we also allow for the fact that the cache cell is written with some value of a different address. This simply means the cell moves from any stage to the state i while all other cells maintain their current state:

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

This system can now equally be represented as a crowd. To this end, let us give the patterns that produce the same languages as described above.

Read Hit:
$$\begin{aligned} & \left\langle \begin{bmatrix} e \\ e \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ e \end{bmatrix}, \begin{bmatrix} m \\ m \end{bmatrix}, \begin{bmatrix} s \\ s \end{bmatrix} \right\} \right\rangle \\ & \left\langle \begin{bmatrix} m \\ m \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ e \end{bmatrix}, \begin{bmatrix} m \\ m \end{bmatrix}, \begin{bmatrix} s \\ s \end{bmatrix} \right\} \right\rangle \\ & \left\langle \begin{bmatrix} s \\ s \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ e \end{bmatrix}, \begin{bmatrix} m \\ m \end{bmatrix}, \begin{bmatrix} s \\ s \end{bmatrix} \right\} \right\rangle \end{aligned}$$

Read Miss: $\left\langle \begin{bmatrix} i \\ s \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ s \end{bmatrix}, \begin{bmatrix} m \\ s \end{bmatrix}, \begin{bmatrix} s \\ s \end{bmatrix} \right\} \right\rangle$

Write Hit:
$$\begin{aligned} & \left\langle \begin{bmatrix} m \\ m \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ e \end{bmatrix}, \begin{bmatrix} m \\ m \end{bmatrix}, \begin{bmatrix} s \\ s \end{bmatrix} \right\} \right\rangle \\ & \left\langle \begin{bmatrix} e \\ m \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ e \end{bmatrix}, \begin{bmatrix} m \\ m \end{bmatrix}, \begin{bmatrix} s \\ s \end{bmatrix} \right\} \right\rangle \\ & \left\langle \begin{bmatrix} s \\ e \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ i \end{bmatrix}, \begin{bmatrix} m \\ i \end{bmatrix}, \begin{bmatrix} s \\ i \end{bmatrix} \right\} \right\rangle \end{aligned}$$

Write Miss: $\left\langle \begin{bmatrix} i \\ e \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ i \end{bmatrix}, \begin{bmatrix} m \\ i \end{bmatrix}, \begin{bmatrix} s \\ i \end{bmatrix} \right\} \right\rangle$

Replacement:
$$\begin{aligned} & \left\langle \begin{bmatrix} m \\ i \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ e \end{bmatrix}, \begin{bmatrix} m \\ m \end{bmatrix}, \begin{bmatrix} s \\ s \end{bmatrix} \right\} \right\rangle \\ & \left\langle \begin{bmatrix} e \\ i \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ e \end{bmatrix}, \begin{bmatrix} m \\ m \end{bmatrix}, \begin{bmatrix} s \\ s \end{bmatrix} \right\} \right\rangle \\ & \left\langle \begin{bmatrix} s \\ i \end{bmatrix}, \left\{ \begin{bmatrix} i \\ i \end{bmatrix}, \begin{bmatrix} e \\ e \end{bmatrix}, \begin{bmatrix} m \\ m \end{bmatrix}, \begin{bmatrix} s \\ s \end{bmatrix} \right\} \right\rangle \end{aligned}$$

In the following, we give a broader definition of this topology than these examples need. The idea of the general definition is that not only a single agent can execute a local change, but also a finite set of agents.

Definition 2.27: *Crowd topology.*

Let Π_k describe the set of all permutations of the set $\{1,\dots,k\}$. We call $p = \left\langle \begin{bmatrix} u_1 \\ v_1 \end{bmatrix}, \dots, \begin{bmatrix} u_k \\ v_k \end{bmatrix}, \left\{ \begin{bmatrix} s_1 \\ t_1 \end{bmatrix}, \dots, \begin{bmatrix} s_m \\ t_m \end{bmatrix} \right\} \right\rangle$ a k-ary pattern. We define the language of p as

$$\mathcal{L}(p) = \bigcup_{\pi \in \Pi_k} R^* \begin{bmatrix} u_{\pi(1)} \\ v_{\pi(1)} \end{bmatrix} R^* \begin{bmatrix} u_{\pi(2)} \\ v_{\pi(2)} \end{bmatrix} R^* \dots R^* \begin{bmatrix} u_{\pi(k)} \\ v_{\pi(k)} \end{bmatrix} R^*$$

where $R = \left(\begin{bmatrix} s_1 \\ t_1 \end{bmatrix} | \dots | \begin{bmatrix} s_m \\ t_m \end{bmatrix} \right)$.
We call any $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ a k-crowd if $\mathcal{L}(\mathcal{T}) = \bigcup_{p \in P} \mathcal{L}(p)$ for some finite set P of k-ary patterns.

Recall that for rings one only needed to check adjacent letters of any statement. This is different for crowds where agents are anonymous. This means, that for any transition we obtain other transitions of the system by permuting the letters of the transition. More formally, we observe that

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \dots \begin{bmatrix} u_n \\ v_n \end{bmatrix} \in \mathcal{L}(\mathcal{T}) \text{ if and only if } \begin{bmatrix} u_{\pi(1)} \\ v_{\pi(1)} \end{bmatrix} \dots \begin{bmatrix} u_{\pi(n)} \\ v_{\pi(n)} \end{bmatrix} \in \mathcal{L}(\mathcal{T}) \text{ for all } \pi \in \Pi_n. \tag{2.7}$$

Hence, with respect to crowds, any statement for the interpretations $\mathcal{V}_{\mathit{trap}}$, $\mathcal{V}_{\mathit{siphon}}$, and $\mathcal{V}_{\mathit{flow}}$ can be reordered arbitrarily while maintaining whether it is a member of $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$.

Lemma 2.33. *Let $\langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ be a k-crowd and $I_1 \ \dots \ I_n \in \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ for any $\mathcal{V} \in \{\mathcal{V}_{\mathit{trap}}, \mathcal{V}_{\mathit{siphon}}, \mathcal{V}_{\mathit{flow}}\}$. Then $I_{\pi(1)} \ \dots \ I_{\pi(n)} \in \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ for all $\pi \in \Pi_n$.*

Proof. Assume, for the sake of contradiction, that there exists a counterexample to the statement of the lemma. That means, we have $I_1 \ \dots \ I_n \in \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ and $\pi \in \Pi_n$ such that $I_{\pi(1)} \ \dots \ I_{\pi(n)} \notin \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$. Hence, there exists $u_1 \ \dots \ u_n \leadsto_{\mathcal{T}} v_1 \ \dots \ v_n$ with $u_1 \ \dots \ u_n \models_{\mathcal{V}} I_{\pi(1)} \ \dots \ I_{\pi(n)}$ but $v_1 \ \dots \ v_n \not\models_{\mathcal{V}} I_{\pi(1)} \ \dots \ I_{\pi(n)}$. $u_{\pi(1)} \ \dots \ u_{\pi(n)} \leadsto_{\mathcal{T}} v_{\pi(1)} \ \dots \ v_{\pi(n)}$ is also a transition because $\pi^{-1} \in \Pi_n$ is a permutation and (2.7). The words accepted by the interpretations $\mathcal{V}_{\mathit{trap}}, \mathcal{V}_{\mathit{siphon}}, \mathcal{V}_{\mathit{flow}}$ are closed under permutations. Thus, $u_{\pi^{-1}(1)} \ \dots \ u_{\pi^{-1}(n)} \models_{\mathcal{V}} I_{\pi^{-1}(\pi(1))} \ \dots \ I_{\pi^{-1}(\pi(n))}$ and $v_{\pi^{-1}(1)} \ \dots \ v_{\pi^{-1}(n)} \not\models_{\mathcal{V}} I_{\pi^{-1}(\pi(1))} \ \dots \ I_{\pi^{-1}(\pi(n))}$. This contradicts the assumption that $I_1 \ \dots \ I_n = I_{\pi^{-1}(\pi(1))} \ \dots \ I_{\pi^{-1}(\pi(n))} \in \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$. □

This means $\text{Inductive}_{\mathcal{V}}(\mathcal{R})$ is closed under reordering of the letters of any statement for the interpretations that we consider. In other words, for every statement, the order of its letters is not important but only their occurrence. Specifically, we can establish that, for crowds, there is a cut-off point such that more occurrences of the same letter do not invalidate membership in $\text{Inductive}_{\mathcal{V}}(\mathcal{R})$ for the interpretations $\mathcal{V} \in \{\mathcal{V}_{\mathit{trap}}, \mathcal{V}_{\mathit{siphon}}, \mathcal{V}_{\mathit{flow}}\}$. Namely, this cut-off point for any k-crowd is $k+1$ for $\mathcal{V}_{\mathit{trap}}$ or $\mathcal{V}_{\mathit{siphon}}$ and $k+3$ for $\mathcal{V}_{\mathit{flow}}$.

Intuitively, the idea is that every interaction between a statement and transition can already be observed if letters in the statement occur fewer than $k+1$ $(k+3)$ times. To make this idea more precise we introduce notation that describes counting letters up to some threshold.

Definition 2.28: *Counting occurrences.*

We let $\text{occ}_A(I_1 \ \ldots \ I_n)$ denote the set $|\{i \in \{1, \ldots, n\} \mid I_i = A\}|$ and $\text{occ}_A^{\leq t}(I)$ the set $\min\{\text{occ}_A(I), t\}$. Moreover, we generalize this to $\text{occ}(I) : 2^\Sigma \to \mathbb{N}$ with $\text{occ}(I)(A) = \text{occ}_A(I)$ and $\text{occ}^{\leq t}(I) : 2^\Sigma \to \{0, \ldots, t\}$ where $\text{occ}^{\leq t}(I)(A) = \text{occ}_A^{\leq t}(I)$ for all $A \in 2^\Sigma$.

Now, we characterize sets of inductive statements based on this notion.

Lemma 2.34. *For all k-crowds $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and $\mathcal{V} \in \{\mathcal{V}_{\mathit{trap}}, \mathcal{V}_{\mathit{siphon}}\}$, the set of all inductive statements for $\mathcal{V}$ is*

$$\left\{I \in \left(2^\Sigma\right)^* \mid \exists I' \in \text{Inductive}_{\mathcal{V}}(\mathcal{R}) \ . \ \text{occ}(I') = \text{occ}^{\leq k+1}(I') = \text{occ}^{\leq k+1}(I)\right\}.$$

This result can be obtained from the observation that adding one letter that already occurs at least $k+1$ times to any statement does not change whether a statement for the interpretations $\mathcal{V}_{\mathit{trap}}$ or $\mathcal{V}_{\mathit{siphon}}$ is inductive or not.

Lemma 2.35. *For all k-crowds $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$, $\mathcal{V} \in \{\mathcal{V}_{\mathit{trap}}, \mathcal{V}_{\mathit{siphon}}\}$, and statements I, I' such that there exists $A \in 2^\Sigma$ with $k+1 < \text{occ}_A(I) = \text{occ}_A(I')+1$ and $\text{occ}_B(I) = \text{occ}_B(I')$ for all $B \in 2^\Sigma \setminus \{A\}$, we have*

$$I \in \text{Inductive}_{\mathcal{V}}(\mathcal{R}) \textit{ if and only if } I' \in \text{Inductive}_{\mathcal{V}}(\mathcal{R}).$$

Proof. Consider the interpretation $\mathcal{V}_{\mathit{trap}}$ and fix $I = I_1 \ \ldots \ I_{n+1}$ and $I' = I'_1 \ \ldots \ I'_n$.

Assume, for now, $I \notin \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$. Thus, there is $u_1 \,\ldots\, u_{n+1} \rightsquigarrow_{\mathcal{T}} v_1 \,\ldots\, v_{n+1}$ such that $u_1 \,\ldots\, u_{n+1} \models_{\mathcal{V}_{trap}} I_1 \,\ldots\, I_{n+1}$ and $v_1 \,\ldots\, v_{n+1} \not\models_{\mathcal{V}_{trap}} I_1 \,\ldots\, I_{n+1}$. Additionally, there exists a k-ary pattern $\left\langle \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \ldots, \begin{bmatrix} x_k \\ y_k \end{bmatrix}, B \right\rangle$ such that

- there are $p_1, \ldots, p_k$ with $u_{p_i} = x_i$ and $v_{p_i} = y_i$ for all $1 \leq i \leq k$ and
- $\begin{bmatrix} u_j \\ v_j \end{bmatrix} \in B$ for all $1 \leq j \leq n+1$ with $j \notin \{p_1, \ldots, p_k\}$.

Because there are $k+2$ occurrences of the letter A in I, there are $i, j \notin \{p_1, \ldots, p_k\}$ such that $I_i = I_j = A$. This means $v_i \notin A$ and $v_j \notin A$. Without loss of generality $u_i \notin A$ or $u_j \in A$. There is a transition $u_1 \,\ldots\, u_{i-1}\, u_{i+1} \,\ldots\, u_n \rightsquigarrow_{\mathcal{T}} v_1 \,\ldots\, v_{i-1}\, v_{i+1} \,\ldots\, v_n$ because $\begin{bmatrix} u_i \\ v_i \end{bmatrix} \in B$. Moreover, by choice of i and j, $u_1 \,\ldots\, u_{i-1}\, u_{i+1} \,\ldots\, u_n \models_{\mathcal{V}_{trap}} I_1 \,\ldots\, I_{i-1}\, I_{i+1} \,\ldots\, I_n$ and $v_1 \,\ldots\, v_{i-1}\, v_{i+1} \,\ldots\, v_n \models_{\mathcal{V}_{trap}} I_1 \,\ldots\, I_{i-1}\, I_{i+1} \,\ldots\, I_n$. With one application of Lemma 2.33 one concludes $I' \notin \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$.

On the other hand, assume $I' \notin \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$. Thus, there is $u_1 \,\ldots\, u_{n+1} \rightsquigarrow_{\mathcal{T}} v_1 \,\ldots\, v_n$ such that $u_1 \,\ldots\, u_n \models_{\mathcal{V}_{trap}} I_1 \,\ldots\, I'_n$ and $v_1 \,\ldots\, v_n \not\models_{\mathcal{V}_{trap}} I_1 \,\ldots\, I'_n$. Additionally, there exists a k-ary pattern $\left\langle \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \ldots, \begin{bmatrix} x_k \\ y_k \end{bmatrix}, B \right\rangle$ such that

- there are $p_1, \ldots, p_k$ with $u_{p_i} = x_i$ and $v_{p_i} = y_i$ for all $1 \leq i \leq k$ and
- $\begin{bmatrix} u_j \\ v_j \end{bmatrix} \in B$ for all $1 \leq j \leq n$ with $j \notin \{p_1, \ldots, p_k\}$.

Because there are $k+1$ occurrences of the letter A in I', there is $i \notin \{p_1, \ldots, p_k\}$ such that $I_i = A$. Therefore, $v_i \notin A$ and, thus, for the transition $u_1 \,\ldots\, u_n\, u_i \rightsquigarrow_{\mathcal{T}} v_1 \,\ldots\, v_n\, v_i$ (which exists because $\begin{bmatrix} u_i \\ v_i \end{bmatrix} \in B$) $u_1 \,\ldots\, u_n\, u_i \models_{\mathcal{V}_{trap}} I'_1 \,\ldots\, I'_n\, A$ and $v_1 \,\ldots\, v_n\, v_i \not\models_{\mathcal{V}_{trap}} I'_1 \,\ldots\, I'_n\, A$ holds. Another application of Lemma 2.33 gives $I \notin \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$.

The proof for $\mathcal{V}_{siphon}$ is analogous. □

Repeatedly applying Lemma 2.35 can be used to grow or shrink any statement of $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ for $\mathcal{V} \in \{\mathcal{V}_{trap}, \mathcal{V}_{siphon}\}$ by adding or removing letter that exceeds the threshold. One final application of Lemma 2.33 gives Lemma 2.34.

On this basis we can capture $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ via the finite automaton with $\mathcal{O}((k+2)^{2^{\Sigma}})$ states.

Corollary 2.9. *Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ be a k-crowd and $\mathcal{V}$ either $\mathcal{V}_{trap}$ or $\mathcal{V}_{siphon}$. One can effectively construct a DFA with $\mathcal{O}((k+2)^{2^{\Sigma}})$ states for* $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$.

Proof. The states of the automaton can be identified with all functions from 2^Σ to $\{0, \dots, k+1\}$. While reading a statement the automaton updates its state to the function that represents $\mathrm{occ}^{\leq k+1}$ if applied to the statement. For every state there is a canonical statement $A_1^{\mathrm{occ}^{\leq k+1}(A_1)} \ \dots \ A_m^{\mathrm{occ}^{\leq k+1}(A_m)}$ that corresponds to $\mathrm{occ}^{\leq k+1}$ where $A_1, \ \dots, \ A_m$ is some enumeration of 2^Σ. Making those states accepting for which the canonical statement is part of $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ concludes the construction. The correctness is an immediate consequence of Lemma 2.34. $\square$

We prove similar results for the interpretation $\mathcal{V}_{\mathit{flow}}$. Here, however, the cut-off point for repeating letters is $k+3$.

Lemma 2.36. *For all k-crowds* $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$, *and statements* I, I' *such that there exists* $A \in 2^\Sigma$ *with* $k + 3 < \mathrm{occ}_A(I) = \mathrm{occ}_A(I') + 1$ *and* $\mathrm{occ}_B(I) = \mathrm{occ}_B(I')$ *for all* $B \in 2^\Sigma \setminus \{A\}$ *holds*

$$I \in \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R}) \text{ if and only if } I' \in \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R}).$$

Proof. For the remainder of the proof fix $I = I_1 \ \dots \ I_{n+1}$ and $I' = I'_1 \ \dots \ I'_n$.

Assume, for now, $I \notin \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$. Thus, there is $u_1 \ \dots \ u_{n+1} \rightsquigarrow_{\mathcal{T}} v_1 \ \dots \ v_{n+1}$ such that $u_1 \ \dots \ u_{n+1} \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_{n+1}$ and $v_1 \ \dots \ v_{n+1} \not\models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_{n+1}$. Additionally, there exists a k-ary pattern $\left\langle \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \dots, \begin{bmatrix} x_k \\ y_k \end{bmatrix}, B \right\rangle$ such that

- there are $p_1, \dots, p_k$ with $u_{p_i} = x_i$ and $v_{p_i} = y_i$ for all $1 \leq i \leq k$ and
- $\begin{bmatrix} u_j \\ v_j \end{bmatrix} \in B$ for all $1 \leq j \leq n+1$ with $j \notin \{p_1, \dots, p_k\}$.

Because there are $k+4$ occurrences of the letter A in I, there are $i_1, i_2, i_3, i_4 \notin \{p_1, \dots, p_k\}$ such that $I_{i_1} = I_{i_2} = I_{i_3} = I_{i_4} = A$.

Since $u_1 \ \dots \ u_{n+1} \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_{n+1}$ there is at most one $i \in \{i_1, i_2, i_3, i_4\}$ such that $u_i \in A$. Therefore, $u_i \notin A$ holds for, say, $i \in \{i_1, i_2, i_3\}$. Moreover, because $v_1 \ \dots \ v_{n+1} \not\models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_{n+1}$, there is either $i \in \{i_1, i_2, i_3\}$ such that $v_i \notin A$ or $v_i \in A$ holds for $i \in \{i_1, i_2, i_3\}$. Let $j \in \{i_1, i_2, i_3\}$ such that $v_j \notin A$ or $v_i \in A$ holds for $i \in \{i_1, i_2, i_3\} \setminus \{j\}$. In either case, there is a transition $u_1 \ \dots \ u_{j-1} \ u_{j+1} \ \dots \ u_n \rightsquigarrow_{\mathcal{T}} v_1 \ \dots \ v_{j-1} \ v_{j+1} \ \dots \ v_n$ because $\begin{bmatrix} u_j \\ v_j \end{bmatrix} \in B$ and $u_1 \ \dots \ u_{j-1} \ u_{j+1} \ \dots \ u_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_{j-1} \ I_{j+1} \ \dots \ I_n$ and $v_1 \ \dots \ v_{j-1} \ v_{j+1} \ \dots \ v_n \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \dots \ I_{j-1} \ I_{j+1} \ \dots \ I_n$. With one application of Lemma 2.33, one concludes $I' \notin \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$.

On the other hand, assume $I' \notin \text{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$. Thus, there is $u_1 \, \ldots \, u_{n+1} \rightsquigarrow_{\mathcal{T}} v_1 \, \ldots \, v_n$ such that $u_1 \, \ldots \, u_n \models_{\mathcal{V}_{\mathit{trap}}} I_1 \, \ldots \, I'_n$ and $v_1 \, \ldots \, v_n \not\models_{\mathcal{V}_{\mathit{trap}}} I_1 \, \ldots \, I'_n$. Additionally, there exists a k-ary pattern $\left\langle \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \ldots, \begin{bmatrix} x_k \\ y_k \end{bmatrix}, B \right\rangle$ such that

- there are $p_1, \ldots, p_k$ with $u_{p_i} = x_i$ and $v_{p_i} = y_i$ for all $1 \leq i \leq k$ and
- $\begin{bmatrix} u_j \\ v_j \end{bmatrix} \in B$ for all $1 \leq j \leq n$ with $j \notin \{p_1, \ldots, p_k\}$.

Because there are $k+3$ occurrences of the letter A in I', there are $i_1, i_2, i_3 \notin \{p_1, \ldots, p_k\}$ such that $I_{i_1} = I_{i_2} = I_{i_3} = A$. Again, $u_i \notin A$ holds for, say, $i \in \{i_1, i_2\}$. Therefore, the transition $u_1 \, \ldots \, u_n \; u_j \rightsquigarrow_{\mathcal{T}} v_1 \, \ldots \, v_n \; v_j$ exists because $\begin{bmatrix} u_j \\ v_j \end{bmatrix} \in B$. Moreover, $u_1 \, \ldots \, u_n \; u_j \models_{\mathcal{V}_{\mathit{flow}}} I'_1 \, \ldots \, I'_n \; A$ and $v_1 \, \ldots \, v_n \; v_j \not\models_{\mathcal{V}_{\mathit{flow}}} I'_1 \, \ldots \, I'_n \; A$ holds since $u_j \notin A$ and

- either $v_j \in A$ which implies $|\Lbag v_1 \, \ldots \, v_n \; v_j \Rbag \cap \Lbag I'_1 \, \ldots \, I'_n \; A \Rbag| \geq 2$,
- or $v_j \notin A$ which implies

$$|\Lbag v_1 \, \ldots \, v_n \; v_j \Rbag \cap \Lbag I'_1 \, \ldots \, I'_n \; A \Rbag| = \underbrace{|\Lbag v_1 \, \ldots \, v_n \Rbag \cap \Lbag I'_1 \, \ldots \, I'_n \Rbag|}_{\neq 1} .$$

Another application of Lemma 2.33 gives $I \notin \text{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$. $\square$

With this result, we obtain, as before, a characterization of $\text{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$ via a finite count of letters.

Corollary 2.10. *For all k-crowds* $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ *the set* $\text{Inductive}_{\mathcal{V}_{\mathit{flow}}}$ *coincides with*

$$\left\{ I \in \left(2^{\Sigma}\right)^* \mid \exists I' \in \text{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R}) \;.\; \text{occ}(I') = \text{occ}^{\leq k+3}(I') = \text{occ}^{\leq k+3}(I) \right\} .$$

With the same idea as before we can translate this characterization into an automaton.

Corollary 2.11. *Let* $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ *be a k-crowd. One can effectively construct a DFA with* $\mathcal{O}((k+4)^{2^{\Sigma}})$ *states for* $\text{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$.

3 Learning inductive invariants

Until now, we have always used all inductive statements to over-approximate the reachability relation. For some questions, however, not all inductive statements are necessary. We want to illustrate this with an example first.

Example 3.1: *Explanation for safety conditions in Example 2.2.*

Recall the token passing algorithm from Example 2.2. We showed that one can prove two safety conditions:

- there always exists at least one token, and
- there never is more than one token.

In particular, Example 2.5 illustrated that

- $\text{“}0 < \text{''} = \{t\}^+ \subseteq \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$ proves that no configuration in n^+ can be reached in $\mathcal{R}$ from any configuration that contains at least one t. In other words, “there is at least one t” is an inductive statement.
- Similarly, $\text{“} < 2\text{''} = \emptyset^* \; \{n\} \; \emptyset^* \; \{n\} \; \emptyset^* \subseteq \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$ proves that no configuration in $\Sigma^* \; t \; \Sigma^* \; t \; \Sigma^*$ can be reached in $\mathcal{R}$ from any configuration which has at most one t. Again, in other words, “there is at most one t” is an inductive statement.

Consequently, $Id(\mathcal{I})\circ \Rightarrow_{\mathcal{V}_{trap}}$ is exactly $Id(\mathcal{I})\circ \rightsquigarrow_{\mathcal{T}}^*$ in this case. But the same is already true for $Id(\mathcal{I})\circ \xRightarrow{\text{“}0<\text{''}\cup\text{“}<2\text{''}}_{\mathcal{V}_{trap}}$. This abstraction relies on “easier” sets of inductive statements. For this, see Figure 3.1 where we give a DFA for $\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$ and Figure 3.2 where we give a DFA for $\{t\}^+ \cup \emptyset^* \; \{n\} \; \emptyset^* \; \{n\} \; \emptyset^* \subseteq \text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$.

Figure 3.1: *An automaton for* $\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$.

This is the minimal DFA that recognizes $\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$ for $\mathcal{R}$ from Example 2.2.

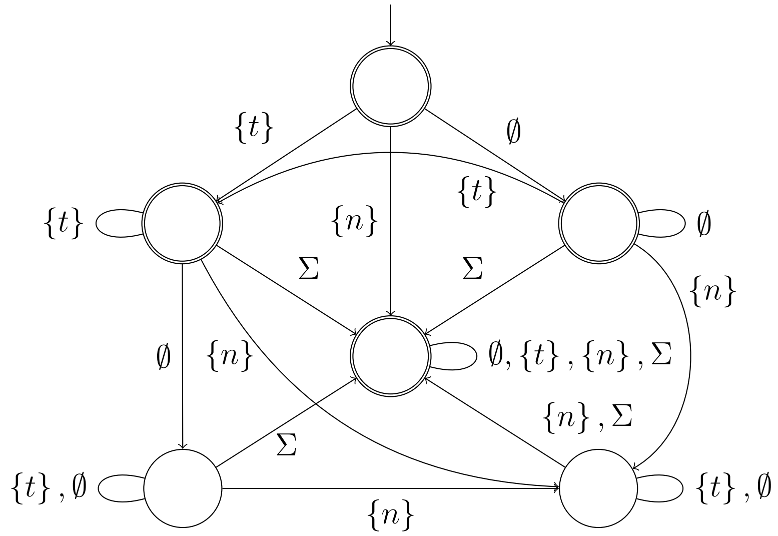
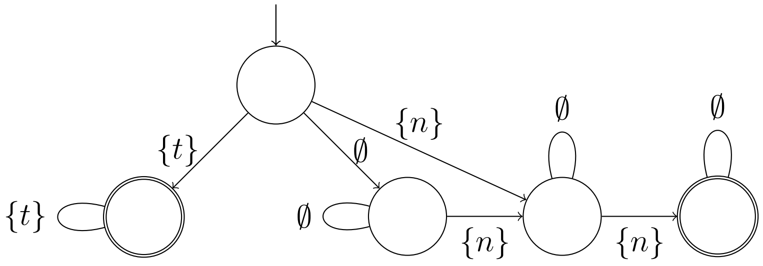


Figure 3.2: *Automata for useful subsets of* $\text{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$.

This DFA recognizes $\{t\}^+ \cup \emptyset^* \ \{n\} \ \emptyset^* \ \{n\} \ \emptyset^*$ which is a sufficient subset of inductive statements to capture the reachability relation from $\mathcal{R}$ from Example 2.2. All omitted transitions lead to a non-accepting sink.



Motivated by this example, we explore in this section how to compute a sufficient set of inductive statements that already establish a safety property. We use *automata learning* for this – a formalism to compute a regular language if one is only allowed to ask two kinds of questions:

Membership: Should the word w be part of the language?

Equivalence: Does this DFA $\mathcal{H}$ already accept the language?

The first question is answered with “yes” or “no” and the second question with either “yes” or with a word v that is accepted by $\mathcal{H}$ but should not, or that should be accepted by $\mathcal{H}$ but is not. In other words, v is picked from the *symmetric difference* of the language of $\mathcal{H}$ and the language that we try to learn. With access to only these two questions, it is possible to learn every regular language [Ang87].

The concept of automata learning has been applied to RMC before [Nei14; Var06; Che+17; Var+04; NJ13]. Specifically, these approaches formulate a learning scenario to obtain a regular set $R \subseteq \Sigma^*$ that is *inductive*; i. e. $\text{target}(\mathit{Id}(R) \circ \rightsquigarrow_{\mathcal{T}}) \subseteq R$. If, additionally, $\mathcal{L}(\mathcal{I}) \subseteq R$ and $\mathcal{L}(\mathcal{B}) \cap R \neq \emptyset$ then one can give a positive answer for this instance of Problem 2.1 since R contains all reachable configurations. Since many examples have such a regular over-approximation R, these approaches perform well on classical benchmarks of parametrized verification. If, on the other hand, no such regular over-approximation exists, then these approaches either run indefinitely or they stop due to some computational limit to the learning process. These limits are, for example, the size of the automaton that should recognize R [Nei14] or the running time of the learning algorithm [Che+17].

The approach that we propose is to learn, for any interpretation $\mathcal{V}$, a sufficient subset S of inductive statements such that the abstraction induced by $\overset{S}{\Rightarrow}_{\mathcal{V}}$ already proves the safety condition. Since the learning procedure for S has a target $\text{Inductive}_{\mathcal{V}}(\mathcal{R})$ that is a regular set itself, we can guarantee that it halts at some point[1]. Moreover, if no S exists to establish the safety condition, this approach terminates with the guarantee that the safety condition cannot be established with inductive statements of $\mathcal{V}$. For instance, for the variant of Example 2.2 in which the transitions do not enforce a unique token, this leads to the definite statement:

> “In the RTS $\left\langle t\, n^*, \left(\begin{bmatrix} n \\ n \end{bmatrix} \middle| \begin{bmatrix} t \\ t \end{bmatrix}\right)^* \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \left(\begin{bmatrix} n \\ n \end{bmatrix} \middle| \begin{bmatrix} t \\ t \end{bmatrix}\right)^* \right\rangle$ there exists *no* trap that shows that $t\ n\ t$ cannot be reached from $t\ n\ n$.”

---

[1] In particular, because we design oracles to answer the two possible questions in such a way that they form a “minimally adequate teacher” [Ang87].

3.1 Learning inductive statements

Conceptionally, we try to solve Problem 2.2 by answering the following question:

Does a DFA $\mathcal{H}$ exist such that

- $\mathcal{L}(\mathcal{H}) \subseteq \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ and
- $Id(\mathcal{I}) \circ \xRightarrow{\mathcal{L}(\mathcal{H})}_{\mathcal{V}} \circ Id(\mathcal{B}) = \emptyset$?

The design of our learning algorithm is as follows.

Context: A fixed interpretation $\langle \Gamma, \mathcal{V} \rangle$.

Input: An RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$, and a NFA $\mathcal{B}$.

Target Concept: DFA $\mathcal{H}$ such that $\mathcal{L}(\mathcal{H}) \subseteq \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ and $\mathcal{L}(\mathcal{I}) \circ \xRightarrow{\mathcal{L}(\mathcal{H})}_{\mathcal{V}} \circ \mathcal{L}(\mathcal{B}) = \emptyset$.

Membership Oracle $\mathcal{O}_{\in}$:

$$\mathcal{O}_{\in}(w) = \begin{cases} \times & \text{if } w \notin \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R}) \\ \checkmark & \text{if } w \in \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R}) \end{cases}$$

Equivalence Oracle $\mathcal{O}_{=}$: First, the oracle checks $\mathcal{L}(\mathcal{H}) \subseteq \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ by returning $I \in \mathcal{L}(\mathcal{H}) \setminus \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ if it exists. The other cases are:

$$\mathcal{O}_{=}(\mathcal{H}) = \begin{cases} I & \text{if } \exists u \in \mathcal{L}(\mathcal{I}), w \in \mathcal{L}(\mathcal{B}) \;.\; u \xRightarrow{\mathcal{L}(\mathcal{H})} w \text{ but } u \cancel{\xRightarrow{\mathcal{L}(\mathcal{H}) \cup \{I\}}} w \\ \times & \begin{array}{l} \text{if } \exists u \in \mathcal{L}(\mathcal{I}), w \in \mathcal{L}(\mathcal{B}) \;.\; u \xRightarrow{\mathcal{L}(\mathcal{H})} w \\ \qquad \text{and } u \xRightarrow{\mathcal{L}(\mathcal{H}) \cup \{I\}} w \text{ for all } I \in \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R}) \end{array} \\ \checkmark & \text{if } \mathcal{L}(\mathcal{I}) \circ \xRightarrow{\mathcal{L}(\mathcal{H})} \cap \mathcal{L}(\mathcal{B}) = \emptyset \end{cases}$$

That means, four cases are distinguished:

- The hypothesis includes a non-inductive statement.
- The hypothesis is not yet strong enough, but there is some inductive statement that can disprove one of the current faults.
- The hypothesis is not yet strong enough, but there exists a counterexample that cannot be removed with any inductive statement.

- The hypothesis is strong enough and proves the desired safety condition.

Note here, that these cases are not mutually exclusive. For example, the second and third cases might be true at the same time. In this case, the answer of the oracle depends on the counterexample to the current hypothesis which is considered. However, eventually, the third case occurs – for instance, if all the fixable cases are exhausted. Regardless, if the oracle returns either × or ✓ we are absolutely sure that we failed or succeeded, respectively.

Remark 3.1. *Essentially, these two oracles form, in the terminology of [Ang87], a "minimally adequate teacher" for* $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$*, because membership queries return exactly whether a statement is a member of* $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ *and equivalence queries always return a statement from the symmetric difference of the language of the hypothesis and* $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$*. Because of [Ang87, Theorem 6] the hypothesis is a DFA for* $\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ *after finitely many steps.*

Implementing the Oracles

Let us first look at the implementation of a *Membership Oracle*. One can utilize Lemma 2.2 to obtain a NFA $\mathcal{M}$ for $\overline{\mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})}$ which is roughly of the size of the transducer $\mathcal{T}$ of RTS $\mathcal{R}$ (if we assume the interpretation to be of constant size). Therefore, one can simply implement the *Membership Oracle* by checking acceptance of $\mathcal{M}$ and negate the answer. Thus, this operation can be implemented in polynomial time for the input $\mathcal{R}$.

For the implementation of the *Equivalence Oracle*, we can use previous results. First, we need to check that the hypothesis $\mathcal{H}$ does not accept any non-inductive statement. To this end, one can compute whether there is a word that is accepted by $\mathcal{H}$ and $\mathcal{M}$ at the same time. This is possible by checking the product construction of these two automata for emptiness. Again, this can be implemented in polynomial time for the inputs $\mathcal{H}$ and $\mathcal{M}$.

At this point, we are assured that the language of the hypothesis only contains inductive statements. We need to check whether these inductive statements are sufficient to establish the safety condition. Recall that, due to Lemma 2.6, one can compute, from the automaton $\mathcal{H}$, the potential reachability relation $\xRightarrow{\mathcal{L}(\mathcal{H})}_{\mathcal{V}}$ which is induced by the inductive statements $\mathcal{L}(\mathcal{H})$. With this and the fact that the composition of relations of transducers can be obtained by a product construction (Lemma 2.1), one can check whether $Id(\mathcal{I})\circ \overset{\mathcal{H}}{\Longrightarrow}_{\mathcal{V}} \circ Id(\mathcal{B}) = \emptyset$ using $\mathcal{O}(\log(|\mathcal{I}|) \cdot |\mathcal{H}| \cdot \log(|\mathcal{B}|))$ space. If this is true the oracle returns ✓. Otherwise we obtain a counterexample $\langle u_1 \ldots u_n, v_1 \ldots v_n\rangle \in Id(\mathcal{I})\circ \overset{\mathcal{H}}{\Longrightarrow}_{\mathcal{V}} \circ Id(\mathcal{B})$. This leads to the question whether $I \in \text{Inductive}_{\mathcal{V}}(\mathcal{R})$ exists such that $u_1 \ldots u_n \models_{\mathcal{V}} I$ and $v_1 \ldots v_n \not\models_{\mathcal{V}} I$.

Problem 3.1 (Word problem)**.** *For a given interpretation $\mathcal{V}$:*

Given: $u_1 \ldots u_n$, $v_1 \ldots v_n$ *and RTS* $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T}\rangle$

Compute: *Does* $I \in \text{Inductive}_{\mathcal{V}}(\mathcal{R})$ *exist such that* $u_1 \ldots u_n \models_{\mathcal{V}} I$ *and* $v_1 \ldots v_n \not\models_{\mathcal{V}} I$*?*

This problem is in the complexity class NP. This observation is immediate if one considers the statement I as a certificate. Since I is a word of length n it is of polynomial length of the instance of Problem 3.1; namely, of the same length as $u_1 \ldots u_n$ and $v_1 \ldots v_n$. This certificate must *not* accepted by $\mathcal{M}$, the automaton that recognizes all non-inductive statements that can be constructed in polynomial time from $\mathcal{T}$. Moreover, whether $u_1 \ldots u_n \models_{\mathcal{V}} I$ and $v_1 \ldots v_n \not\models_{\mathcal{V}} I$ can be checked with two membership queries to $\mathcal{V}$.

Lemma 3.1. *Problem 3.1 is in* NP*.*

Later, we present a polynomial time reduction to SAT – the problem of whether a given propositional formula in conjunctive normal form has a satisfying assignment. This reduction also proves this result but, additionally, has practical application: Because there are heavily optimized solvers for SAT, we can compute separating inductive statements by reducing the instance to SAT, finding a satisfying assignment for the propositional formula, and extracting, from the assignment, the separating statement.

First, though, we prove that, for the interpretation $\mathcal{V}_{\mathit{flow}}$, Problem 3.1 is NP-hard. We establish this result via a polynomial time reduction from a variant of SAT (3-SAT) which restricts each clause to only have three literals:

Problem 3.2 (3-SAT)**.**

Given: $\{C_1, \ldots, C_n\}$ *with* $C_i = \left\{L_{3\cdot(i-1)+1}, L_{3\cdot(i-1)+2}, L_{3\cdot(i-1)+3}\right\}$ *for each* $1 \leq i \leq n$ *where each* $L \in C_i$ *is an element from* $\{x, \overline{x} \colon x \in \mathcal{X}\}$ *for some* $\mathcal{X}$*.*

Compute: *Exists* $J : \mathcal{X} \to \{0, 1\}$ *such that for each* $1 \leq i \leq n$ *there is at least one literal in* C_i *satisfied?*

Roughly speaking, the proof goes as follows: We use an alphabet of three letters $s, h, \#$. Then, we encode a given instance of the 3-SAT problem into Problem 3.1 such that every literal of the formula corresponds to one position of words $u_1 \ \dots \ u_{3 \cdot n}$ and $v_1 \ \dots \ v_{3 \cdot n}$. More specifically, the literals of the first clause are represented by the first three positions, the literals of the second clause by the next three positions, and so on. In other words, every clause is represented by one triplet of positions. We choose the transitions of the RTS such that any flow $I_1 \ \dots \ I_{3 \cdot n}$ that separates $u_1 \ \dots \ u_{3 \cdot n}$ and $v_1 \ \dots \ v_{3 \cdot n}$ satisfies the following properties:

- For every clause C, exactly one of I_i, I_{i+1}, and I_{i+2} contains the letter h where $I_i \ I_{i+1} \ I_{i+2}$ is the triplet that represents C.
- For any two distinct positions i and j which represent a literal and its negation, at most one of I_i and I_j contains h.

In this way, any separating flow encodes a satisfying assignment for the propositional formula by satisfying those literals for which one position exists that contains h. Similarly, any satisfying assignment for the propositional formula can be used to obtain a separating flow. Roughly speaking, one can add the letter h to the position of exactly one literal of every clause that is satisfied by the assignment.

Lemma 3.2. *For $\mathcal{V}_{\mathit{flow}}$, Problem 3.1 is* NP*-hard.*

Proof. Reduce from Problem 3.2. In the following, any assignment $J \colon \mathcal{X} \to \{0, 1\}$ is implicitly expanded to the domain $\{x, \overline{x} \colon x \in \mathcal{X}\}$ with $J(\overline{x}) = 1 - J(x)$ for all $x \in \mathcal{X}$. For technical reasons, which do not restrict the problem, assume that there are always more than two clauses in the instances of Problem 3.2.

For the reduction, fix an alphabet of three elements $\{s, h, \#\}$. Set the word $s^{3 \cdot n}$ as the initial word u and $\#^{3 \cdot n}$ as the final word v. For the definition of the transitions, consider first

$$X_{i,j} = \begin{cases} \begin{bmatrix} \# \\ \# \end{bmatrix}^{i-1} \begin{bmatrix} h \\ h \end{bmatrix} \begin{bmatrix} \# \\ \# \end{bmatrix}^{j-i-1} \begin{bmatrix} \# \\ h \end{bmatrix} \begin{bmatrix} \# \\ \# \end{bmatrix}^{3 \cdot n-j} & \text{if } i < j \\ \begin{bmatrix} \# \\ \# \end{bmatrix}^{j-1} \begin{bmatrix} \# \\ h \end{bmatrix} \begin{bmatrix} \# \\ \# \end{bmatrix}^{i-j-1} \begin{bmatrix} h \\ h \end{bmatrix} \begin{bmatrix} \# \\ \# \end{bmatrix}^{3 \cdot n-i} & \text{if } j < i \end{cases}$$

In other words, the transition $X_{i,j}$ has the letter $\begin{bmatrix} h \\ h \end{bmatrix}$ at the i-th position and the letter $\begin{bmatrix} \# \\ h \end{bmatrix}$ at the j-th position. Everywhere else the word of this transition has the letter $\begin{bmatrix} \# \\ \# \end{bmatrix}$. The set of all $X_{i,j}$ for all $i \neq j$ such that $L_i = \overline{L_j}$ can be recognized with a DFA with

$(3 \cdot n + 1) \cdot ((3 \cdot n + 1) \cdot 2 + 2)$ states (not counting a non-accepting sink state $\bot$). The automaton moves through three phases while reading a transition:

1. No letter in $\left\{\begin{bmatrix}\#\\h\end{bmatrix}, \begin{bmatrix}h\\h\end{bmatrix}\right\}$ has occurred.
2. One letter in $\left\{\begin{bmatrix}\#\\h\end{bmatrix}, \begin{bmatrix}h\\h\end{bmatrix}\right\}$ has occurred at position i.
3. Two letters in $\left\{\begin{bmatrix}\#\\h\end{bmatrix}, \begin{bmatrix}h\\h\end{bmatrix}\right\}$ at matching positions i and j have occurred.

For all these phases the automaton keeps a running count (between 0 and $3 \cdot n$) of the steps it has already taken. For the second phase the automaton stores in its state the index i and which of the three letters occurred. Consequently, the automaton can be constructed with the states

$$\begin{aligned}
&\{1\} \times \{0, \dots, 3 \cdot n\} \\
\cup\, &\{2\} \times \{0, \dots, 3 \cdot n\} \times \{0, \dots, 3 \cdot n\} \times \left\{\begin{bmatrix}\#\\h\end{bmatrix}, \begin{bmatrix}h\\h\end{bmatrix}\right\} \\
\cup\, &\{3\} \times \{0, \dots, 3 \cdot n\}\,.
\end{aligned}$$

The transition function δ follows

$$\begin{aligned}
&\delta(\langle 1, p\rangle, \begin{bmatrix}\#\\\#\end{bmatrix}) = \langle 1, p+1\rangle && \text{for all } 0 \le p < n \\
&\delta(\langle 1, p\rangle, \begin{bmatrix}\#\\h\end{bmatrix}) = \left\langle 2, p+1, p, \begin{bmatrix}\#\\h\end{bmatrix}\right\rangle && \text{for all } 0 \le p < n \\
&\delta(\langle 1, p\rangle, \begin{bmatrix}h\\h\end{bmatrix}) = \left\langle 2, p+1, p, \begin{bmatrix}h\\h\end{bmatrix}\right\rangle && \text{for all } 0 \le p < n \\
&\delta(\langle 2, p, i, \ell\rangle, \begin{bmatrix}\#\\\#\end{bmatrix}) = \langle 2, p+1, i, \ell\rangle && \text{for all } 0 \le p < n \\
&\delta(\left\langle 2, p, i, \begin{bmatrix}h\\h\end{bmatrix}\right\rangle, \begin{bmatrix}\#\\h\end{bmatrix}) = \langle 3, p+1\rangle && \text{for all } 0 \le p < n \text{ s. t. } L_i = \overline{L_{p+1}} \\
&\delta(\left\langle 2, p, i, \begin{bmatrix}\#\\h\end{bmatrix}\right\rangle, \begin{bmatrix}h\\h\end{bmatrix}) = \langle 3, p+1\rangle && \text{for all } 0 \le p < n \text{ s. t. } L_i = \overline{L_{p+1}} \\
&\delta(\langle 3, p\rangle, \begin{bmatrix}\#\\\#\end{bmatrix}) = \langle 3, p+1\rangle && \text{for all } 0 \le p < n
\end{aligned}$$

while all other transitions lead to $\bot$. The final unique final state is $\langle 3, n\rangle$.

Additionally, we add the transitions

$$\left(\begin{bmatrix}s\\\#\end{bmatrix} \begin{bmatrix}s\\\#\end{bmatrix} \begin{bmatrix}s\\\#\end{bmatrix}\right)^* \begin{bmatrix}s\\h\end{bmatrix} \begin{bmatrix}s\\h\end{bmatrix} \begin{bmatrix}s\\h\end{bmatrix} \left(\begin{bmatrix}s\\\#\end{bmatrix} \begin{bmatrix}s\\\#\end{bmatrix} \begin{bmatrix}s\\\#\end{bmatrix}\right)^* . \tag{3.1}$$

This set of transitions can be recognized by a DFA with 9 states (not counting a non-accepting sink state). Because the first letter of every accepted transition uniquely determines whether the transition is some $X_{i,j}$ or part of (3.1), a DFA for all transitions can be constructed with $\mathcal{O}(9 + (3 \cdot n + 1) \cdot ((3 \cdot n + 1) \cdot 2 + 2))$ states. This concludes the reduction.

For the correctness proof of the reduction, assume, first, that an assignment $J : \mathcal{X} \to \{0, 1\}$ exists which makes at least one literal in every clause true. For every clause C_i let $p_i \in \{1, \ldots, 3 \cdot n\}$ be an index such that J satisfies the literal $L_{p_i} \in C_i$. Choose $I_1 \ \ldots \ I_{3 \cdot n}$ such that $I_{p_i} = \{h\}$ for all $1 \leq i \leq n$ and $I_k = \emptyset$ for all $1 \leq k \leq 3 \cdot n$ with $k \notin \{p_1, \ldots, p_n\}$. The statement $I = (I_1 \cup \{s\}) \ I_2 \ \ldots \ I_{3 \cdot n}$

- is satisfied by u because only the first letter of this statement contains s,
- is not satisfied by v because no letter of this statement contains #, and
- is inductive because:
  - Pick any $1 \leq i \leq n$. Consider any position j of the triplet that corresponds to the i-th clause; that is, $j \in \{3 \cdot (i - 1) + 1, 3 \cdot (i - 1) + 2, 3 \cdot (i - 1) + 3\}$. Then, $h \in I_j$ if and only if $j = p_i$. Thus, $w \models_{\mathcal{V}_{\mathit{flow}}} I$ for all transitions $u \rightsquigarrow_{\mathcal{T}} w$ from (3.1) because no letter in I contains #.
  - $J(L_i) = 1$ holds for all $\langle x, y \rangle = X_{i,j}$ with $x \models I$. Therefore, the letter I_j does not contain h since $L_i = \overline{L_j}$ and, thus, $J(L_j) = 0$.

This concludes the first direction of the correctness proof.

On the other hand, assume there is an inductive statement $I_1 \ \ldots \ I_{3 \cdot n}$ such that $u \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_{3 \cdot n}$ and $v \not\models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_{3 \cdot n}$. Because $u \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_{3 \cdot n}$ also $w \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_{3 \cdot n}$ for all $u \rightsquigarrow_{\mathcal{T}} w$ from (3.1). Consequently, there are no $i \neq j$ such that $\# \in I_i$ and $\# \in I_j$ because, otherwise, there is $u \rightsquigarrow_{\mathcal{T}} w$ from (3.1) such that the i-th and j-th letters of w are # since $n > 2$. Moreover, there is no unique $1 \leq i \leq n$ such that $\# \in I_i$ because $v \not\models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_{3 \cdot n}$. Therefore, $\# \notin I_i$ for all $1 \leq i \leq n$. From this and $w \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_{3 \cdot n}$ for all $u \rightsquigarrow_{\mathcal{T}} w$ from (3.1), it follows immediately that there are $p_i \in \{(i - 1) \cdot 3 + 1, (i - 1) \cdot 3 + 2, (i - 1) \cdot 3 + 3\}$ such that $h \in I_{p_i}$ for all $1 \leq i \leq n$.

Secondly, pick $i \neq j$ such that $L_{p_i} = \overline{L_{p_j}}$ and $h \in I_{p_i}$. Establish now that $h \notin I_{p_j}$. For this, consider $\langle x, y \rangle = X_{p_i, p_j}$. Then, $x \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_{3 \cdot n}$ because $h \in I_{p_i}$ but $\# \notin I_k$ for all $1 \leq k \leq n$. Since the p_i-th and p_j-th letters of y are h and $h \in I_{p_i}$, $h \notin I_{p_j}$ follows from $y \models_{\mathcal{V}_{\mathit{flow}}} I_1 \ \ldots \ I_{3 \cdot n}$.

In conclusion, one can construct $J : \mathcal{X} \to \{0,1\}$ such that $J(L_{p_i}) = 1$ for all $1 \leq i \leq n$. Moreover, by choice of $p_1, \dots, p_n$, the assignment J is a model of the propositional formula. ☐

Consequently, this problem is, in general and for the case of $\mathcal{V}_{\mathit{flow}}$, NP-complete.

Remark 3.2. *Since Problem 3.1 is* NP*-hard for* $\mathcal{V}_{\mathit{flow}}$*, the variant of Problem 3.1 in which the interpretation is part of the input also is* NP*-hard. Moreover, the argument of using a separating inductive statement as a certificate also applies to this variant. Therefore, Problem 3.1 in the variant where the interpretation is part of the input is* NP*-complete.*

Because Problem 3.1 is in NP, it can be reduced, in polynomial time, to SAT (since SAT is NP-hard). As we demonstrate now, this reduction is straightforward. Moreover, one can extract separating inductive statements from satisfying assignments for the constructed propositional formula. This allows us to leverage solvers for SAT to solve Problem 3.1 and compute separating inductive statements. We separate the design of the propositional formula into four parts:

Form of the Certificate: Here we will introduce and restrict propositional variables such that they eventually encode the word I.

Compatibility with $u_1 \, \dots \, u_n$: Here we will encode the run of $\mathcal{V}$ on $u_1 \, \dots \, u_n$ and I and make sure it is accepting.

Non-Compatibility with $v_1 \, \dots \, v_n$: Here we will encode the run of $\mathcal{V}$ on $v_1 \, \dots \, v_n$ and I and make sure it is *not* accepting.

Inductivity of I: This is the most complex part of the formula. Here we have to make sure that I is not accepted by *any* run of $\mathcal{M}$, the NFA that is roughly of the size of the transducer and recognizes this set of all non-inductive statements.

Form of the Certificate

Initially, let us consider how to encode the word I. This word is chosen from the set Γ^n. We fix $\Gamma = \{\gamma_1, \dots, \gamma_m\}$ and use the variables

$$\{\gamma_1(1), \gamma_2(1), \dots, \gamma_m(1), \gamma_1(2), \dots, \gamma_m(n)\} = \Gamma \times \{1, \dots, n\}\,.$$

The intended semantics of these variables is that $\gamma_j(i)$ represents whether the i-th letter of I is γ_j. To ensure that any solution to our propositional formula corresponds to exactly one word $I \in \Gamma^n$ any model of the formula must not satisfy $\gamma(i)$ and $\gamma'(i)$ for any $1 \leq i \leq n$ and two *distinct* $\gamma, \gamma' \in \Gamma$ at the same time. Thus, we introduce the macro

$$\mathit{ExactlyOne}(V) = \bigvee_{v \in V} v \wedge \bigwedge_{v,v' \in V : v \neq v'} \neg (v \wedge v')$$

and add

$$\bigwedge_{1 \leq i \leq n} \mathit{ExactlyOne}(\Gamma \times \{i\}) \tag{3.2}$$

to the formula. One can now verify that any model of the formula in (3.2) satisfies, for every $1 \leq i \leq n$, exactly one propositional variable in $\Gamma \times \{i\}$. Consequently, we can identify with any model the *unique* word $I_1 \ \ldots \ I_n$ such that the model satisfies $I_i(i)$ for every $1 \leq i \leq n$.

Compatibility with $u_1 \ \ldots \ u_n$

Since we assume that $\mathcal{V}$ is a DFA there is exactly one sequence $q_0 \ \ldots \ q_n$ of states from $\mathcal{V}$ that is compatible with reading $\langle u_1, I_1 \rangle \ \ldots \ \langle u_n, I_n \rangle$. We capture this sequence with propositional variables $Q_{\mathcal{V}} \times \{0, \ldots, n\}$ where $Q_{\mathcal{V}}$ is the set of states of $\mathcal{V}$. Again, we want to ensure that every model represents this unique sequence accurately. For this we need to encode the transition function of $\mathcal{V}$, make sure that the first letter is the initial state $q_0^{\mathcal{V}}$ of $\mathcal{V}$, and, finally, that the last state is an accepting one; that is, in $F_{\mathcal{V}}$. We capture this via the following formula:

$$\begin{gathered}\bigwedge_{0 \leq i \leq n} \mathit{ExactlyOne}(Q_{\mathcal{V}} \times \{i\}) \\ \wedge q_0^{\mathcal{V}}(0) \wedge \bigwedge_{\substack{0 \leq i < n \\ q \in Q_{\mathcal{V}}, \gamma \in \Gamma}} q(i) \wedge \gamma(i+1) \rightarrow \delta_{\mathcal{V}}(q, \gamma)(i+1) \wedge \bigvee_{q \in F_{\mathcal{V}}} q(n)\end{gathered} \tag{3.3}$$

Non-Compatibility with $v_1 \dots v_n$

This formula is very similar to (3.3). However, this time we make sure that the run does not end in an accepting state but a rejecting one.

$$\begin{aligned}&\bigwedge_{0 \leq i \leq n} \mathit{ExactlyOne}(Q_{\mathcal{V}} \times \{i\})\\ &\wedge q_0^{\mathcal{V}}(0) \wedge \bigwedge_{\substack{0 \leq i < n\\ q \in Q_{\mathcal{V}}, \gamma \in \Gamma}} q(i) \wedge \gamma(i+1) \rightarrow \delta_{\mathcal{V}}(q,\gamma)(i+1) \wedge \neg \bigvee_{q \in F_{\mathcal{V}}} q(n)\end{aligned} \tag{3.4}$$

Inductivity of I

Finally, we have to make sure that I is rejected by $\mathcal{M}$. To this end, we axiomatize a reachability analysis on the graph of $\mathcal{M}$ for I. More precisely, we try to find all states that are reachable in $\mathcal{M}$ while reading I. This is similar to the two parts before. This time, however, there is not one unique run but we need to compute all states that are reachable while reading I to make sure that there is not any accepting run. Let us fix the automaton $\mathcal{M}$ as $\langle Q_{\mathcal{M}}, Q_0^{\mathcal{M}}, \Gamma, \Delta_{\mathcal{M}}, F_{\mathcal{M}} \rangle$, first. As before, we use the atomic propositions $Q_{\mathcal{M}} \times \{0, \dots, n\}$. This time, the intended meaning of any proposition $q(i)$ is that one can reach state q in $\mathcal{M}$ while reading $I_1 \dots I_i$. The structure of the following formula mirrors the structure of the previous ones. One of the differences is that we do not enforce a single state per position. Also, we now have multiple initial states. We obtain:

$$\bigwedge_{q \in Q_0^{\mathcal{M}}} q(0) \wedge \bigwedge_{\substack{0 \leq i < n\\ \langle q,\gamma,p\rangle \in \Delta_{\mathcal{M}}}} q(i) \wedge \gamma(i+1) \rightarrow p(i+1) \wedge \neg \bigvee_{q \in F_{\mathcal{M}}} q(n) \tag{3.5}$$

Assembling the final formula

Because of the way we have created these formulas we obtained for the conjunction φ of (3.2), (3.3), (3.4), and (3.5) the crucial result:

Lemma 3.3. *There is a model J for φ if and only if Problem 3.1 can be answered with yes. Moreover, one can effectively obtain a separating inductive statement for Prob-*

lem 3.1 from any model J for φ.

Proof. $I_1 \;\dots\; I_n$ where $J(I_i) = 1$ for all $1 \leq i \leq n$ is a separating inductive statement for Problem 3.1 for the interpretation $\mathcal{V}$ with $u_1 \;\dots u_n$, $v_1 \;\dots v_n$ and $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$:

- The unique run of $\mathcal{V}$ on $\langle u_1, I_1 \rangle \;\dots\; \langle u_n, I_n \rangle$ is accepting because J is a model of (3.3).
- The unique run of $\mathcal{V}$ on $\langle v_1, I_1 \rangle \;\dots\; \langle v_n, I_n \rangle$ is not accepting because J is a model of (3.4).
- There is no accepting run of $\mathcal{M}$ on $I_1 \;\dots\; I_n$ because J is a model of (3.5).

□

With this, we construct the equivalence oracle in Algorithm 1.

```
Data: RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ and NFA $\mathcal{B}$
Input: Hypothesis $\mathcal{H}$
Output: $\checkmark, \times,$ or $I \in \Gamma^*$
begin
    $\mathcal{M} \leftarrow$ getAutomatonFor($\overline{\text{Inductive}_{\mathcal{V}}(\mathcal{R})}$);
    if $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{M}) \neq \emptyset$ then
        return $I \in \mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{M})$;
    end
    $\mathcal{D} \leftarrow$ getAutomatonFor($\mathcal{L}(\mathcal{I}) \circ \xRightarrow{\mathcal{L}(H)} \circ \mathcal{L}(\mathcal{B})$);
    if $\mathcal{L}(\mathcal{D}) = \emptyset$ then
        return $\checkmark$;
    end
    $\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \;\dots\; \begin{bmatrix} u_n \\ v_n \end{bmatrix} \leftarrow$ getWordFrom($\mathcal{L}(\mathcal{D})$);
    $I \leftarrow$ disprove($\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \;\dots\; \begin{bmatrix} u_n \\ v_n \end{bmatrix}$);
    if $I$ = null then
        return $\times$;
    end
    return $I$;
end
```

Algorithm 1: The implementation of an equivalence oracle for an interpretation $\mathcal{V}$.

3.2 The word problem for concrete interpretations

In the following, we consider Problem 3.1 for the three interpretations $\mathcal{V}_{trap}$, $\mathcal{V}_{siphon}$, and $\mathcal{V}_{flow}$. In particular, we discuss a reduction of Problem 3.1 for the interpretation $\mathcal{V}_{flow}$ that yields a simpler propositional formula. Additionally, we show that Problem 3.1 can be solved in PTime for the interpretations $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$.

The interpretation $\mathcal{V}_{flow}$

For the interpretation $\mathcal{V}_{flow}$, statements are encoded in the alphabet $\Gamma = 2^{\Sigma}$. Moreover, the steps of $\mathcal{V}_{flow}$ distinguish pairs $\langle v, I\rangle$ by the fact whether $v \in I$ or $v \notin I$. This allows us to slightly change the encoding of the separating statement $I_1 \, \ldots \, I_n$. In particular, we can now fix for the encoding of $I_1 \, \ldots \, I_n$ the propositional variables $\Sigma \times \{1, \ldots, n\}$. Then, we can relate any assignment $J : (\Sigma \times \{1, \ldots, n\}) \to \{0, 1\}$ to $I_1 \, \ldots \, I_n$ by setting $I_i = \{\sigma \in \Sigma \mid J(\langle \sigma, i\rangle) = 1\}$ and vice versa. One can observe here that *any* assignment encodes a statement and we do not need to restrict it further. Therefore, the part of the propositional formula for the generic case that encoded that we necessarily obtain a word from Γ^n (that is, Equation (3.2)) can be removed entirely. Additionally, recall that $u_1 \, \ldots \, u_n \models_{\mathcal{V}_{flow}} I_1 \, \ldots \, I_n$ if and only if there is exactly one $1 \leq i \leq n$ such that $u_i \in I_i$. That, however, means that we can replace the formulas from Equation (3.3) and Equation (3.4) with

$$\mathit{ExactlyOne}\Big(\bigcup_{1 \leq i \leq n} \{\langle u_i, i\rangle\}\Big) \text{ and } \neg\,\mathit{ExactlyOne}\Big(\bigcup_{1 \leq i \leq n} \{\langle v_i, i\rangle\}\Big),$$

respectively. Equation (3.5) remains mostly the same. However, we can construct a propositional formula to test that the considered statement $I_1 \, \ldots \, I_n$ is inductive directly from the transducer $\mathcal{T}$[2] of the RTS. To this end, recall that a statement is *not* inductive if there exists one transition $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \ldots \begin{bmatrix} x_n \\ y_n \end{bmatrix}$ that is accepted by $\mathcal{T}$ for which holds that $x_1 \, \ldots \, x_n \models_{\mathcal{V}_{flow}} I_1 \, \ldots \, I_n$ and $y_1 \, \ldots \, y_n \not\models_{\mathcal{V}_{flow}} I_1 \, \ldots \, I_n$. Effectively, it suffices to guess this transition and verify, first, that there is exactly one index i exists such that $x_i \in I_i$ and, second, that either there is no index j such that $y_j \in I_j$ or more than one. An NFA that performs this test can be constructed with the states $\{0, 1\} \times Q_{\mathcal{T}} \times \{0, 1, 2\}$. Semantically speaking, a state $\langle k, q, \ell\rangle$ corresponds to the observation that one can reach

---

[2] To construct this formula it is sufficient to iterate once over all steps in $\Delta_{\mathcal{T}}$. Additionally, this removes the need to compute all letters from 2^{Σ} for the formula which has practical benefits.

the state q of $\mathcal{T}$ with a word $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \dots \begin{bmatrix} x_m \\ y_m \end{bmatrix}$ such that there are k many indices i where $x_i \in I_i$[3] while, on the other hand, there are ℓ many indices (or more than 2 if $\ell = 2$) j where $y_j \in I_j$. Consequently, the initial and accepting states are $\{0\} \times Q_0^{\mathcal{T}} \times \{0\}$ and $\{1\} \times F_{\mathcal{T}} \times \{0, 2\}$, respectively. With this in mind, one can translate Equation (3.5) for this specific use case to

$$
\begin{aligned}
&\bigvee_{q_0 \in Q_0^{\mathcal{T}}} \langle \langle 0, q_0, 0 \rangle, 0 \rangle \wedge \neg \bigvee_{f \in F_{\mathcal{T}}} \langle \langle 1, f, 0 \rangle, n \rangle \vee \langle \langle 1, f, 2 \rangle, n \rangle \\
&\wedge \bigwedge_{\substack{0 \leq i < n \\ \left\langle q, \begin{bmatrix} x \\ y \end{bmatrix}, p \right\rangle \in \Delta_{\mathcal{T}}}}
\left(
\begin{array}{l}
\langle \langle 0, q, 0 \rangle, i \rangle \wedge \langle x, i+1 \rangle \wedge \langle y, i+1 \rangle \rightarrow \langle \langle 1, p, 1 \rangle, i+1 \rangle \\
\wedge \langle \langle 0, q, 1 \rangle, i \rangle \wedge \langle x, i+1 \rangle \wedge \langle y, i+1 \rangle \rightarrow \langle \langle 1, p, 2 \rangle, i+1 \rangle \\
\wedge \langle \langle 0, q, 2 \rangle, i \rangle \wedge \langle x, i+1 \rangle \wedge \langle y, i+1 \rangle \rightarrow \langle \langle 1, p, 2 \rangle, i+1 \rangle \\
\wedge \displaystyle\bigwedge_{k \in \{0,1\}} \left(
\begin{array}{l}
\langle \langle k, q, 0 \rangle, i \rangle \wedge \neg \langle x, i+1 \rangle \wedge \langle y, i+1 \rangle \rightarrow \langle \langle k, p, 1 \rangle, i+1 \rangle \\
\wedge \langle \langle k, q, 1 \rangle, i \rangle \wedge \neg \langle x, i+1 \rangle \wedge \langle y, i+1 \rangle \rightarrow \langle \langle k, p, 2 \rangle, i+1 \rangle \\
\wedge \langle \langle k, q, 2 \rangle, i \rangle \wedge \neg \langle x, i+1 \rangle \wedge \langle y, i+1 \rangle \rightarrow \langle \langle k, p, 2 \rangle, i+1 \rangle
\end{array}
\right) \\
\wedge \displaystyle\bigwedge_{\ell \in \{0,1,2\}} \left(
\begin{array}{l}
\langle \langle 0, q, \ell \rangle, i \rangle \wedge \langle x, i+1 \rangle \wedge \neg \langle y, i+1 \rangle \rightarrow \langle \langle 1, p, \ell \rangle, i+1 \rangle \\
\wedge \langle \langle 0, q, \ell \rangle, i \rangle \wedge \langle x, i+1 \rangle \wedge \neg \langle y, i+1 \rangle \rightarrow \langle \langle 1, p, \ell \rangle, i+1 \rangle
\end{array}
\right) \\
\wedge \displaystyle\bigwedge_{\substack{k \in \{0,1\} \\ \ell \in \{0,1,2\}}} \langle \langle k, q, \ell \rangle, i \rangle \wedge \neg \langle x, i+1 \rangle \wedge \neg \langle y, i+1 \rangle \rightarrow \langle \langle k, q, \ell \rangle, i+1 \rangle
\end{array}
\right)
\end{aligned}
$$

Naturally, a similar approach works for the interpretations $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$. However, we show in the following that for these interpretations Problem 3.1 can be solved in polynomial time.

A polynomial time algorithm for the word problem for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$

Again, we focus on $\mathcal{V}_{trap}$ since the arguments for $\mathcal{V}_{siphon}$ are analogous. Central to our argument is Corollary 2.4. We use this observation (and the idea of computing separators in general) to obtain an algorithm for Problem 3.1 that runs in polynomial time. Recall that one can compute the separator for a word $v_1 \dots v_n$ by starting with the statement $\Sigma \setminus \{v_1\} \dots \Sigma \setminus \{v_n\}$. This statement is repeatedly refined. Specifically, if a transition $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \dots \begin{bmatrix} x_n \\ y_n \end{bmatrix}$ exists such that $x_1 \dots x_n$ satisfies the current statement and $y_1 \dots y_n$ does not, then one removes x_i from the i-th letter of the statement for all $1 \leq i \leq n$. In

---

[3] The case that $k > 1$ is not considered because the origin of any transition where there is more than one index i such that $x_i \in I_i$ does not satisfy the statement.

this way, there is a separator sequence for two words of length n with at most $n \cdot (|\Sigma| - 1)$ steps since we remove at least one letter from one index at every step. We show now that finding a refining transition in $\mathcal{T}$ for every step is possible in polynomial time. Thus, we can compute the separator for $v_1 \ \dots \ v_n$ in polynomial time because one can do so by running a polynomial time algorithm at most $n \cdot (|\Sigma| - 1)$ times.

We find a refining transition by constructing a graph (of size polynomial in n and $\mathcal{T}$) such that any path from a set of initial states to a set of final states is annotated with one. Essentially we are looking for a transition accepted by $\mathcal{T}$ where the origin of the transition satisfies the current element $I = I_1 \ \dots \ I_n$ of the separator sequence (in the sense of $\mathcal{V}_{trap}$); that is, there is at least one index where the first element of the letter of the transition is part of the letter of I at the same index, while the same is not true for the target of the transition. To this end, we explore the graph of $\mathcal{T}$ for n steps. While doing this, we make sure to use in the i-th step only a step $\left\langle q, \begin{bmatrix} x \\ y \end{bmatrix}, p \right\rangle$ of $\mathcal{T}$ such that $y \notin I_i$. This guarantees that the target of the transition does not satisfy I. If after n steps we can reach an accepting state of $\mathcal{T}$ and in at least one of these steps, say i, we used a step $\left\langle q, \begin{bmatrix} x \\ y \end{bmatrix}, p \right\rangle$ such that $x \in I_i$, then the path to this accepting state is annotated with a refining transition. Formally, we annotate the states of $\mathcal{T}$ with a number from $\{0, \dots, n\}$ to indicate in which step we are, and a bit to indicate whether we already used a step $\left\langle q, \begin{bmatrix} x \\ y \end{bmatrix}, p \right\rangle$ such that $x \in I_i$. This leads us to the graph $\langle V, E \rangle$ with

- $V = \{0, \dots, n\} \times Q_{\mathcal{T}} \times \{0, 1\}$, and
- $E = \left\{ \left\langle \langle i, q, b \rangle, \begin{bmatrix} x \\ y \end{bmatrix}, \langle i+1, p, b' \rangle \right\rangle : \begin{array}{l} \left\langle q, \begin{bmatrix} x \\ y \end{bmatrix}, p \right\rangle \in \Delta_{\mathcal{T}}, 0 \le i < n, y \notin I_{i+1}, \\ b' = 0 \text{ iff } b = 0 \text{ and } x \notin I_{i+1} \end{array} \right\}.$

We sketch this graph in Figure 3.3. In this graph, we are computing now whether there is a path from any element in $\{0\} \times Q_0^{\mathcal{T}} \times \{0\}$ to any element in $\{n\} \times F_{\mathcal{T}} \times \{1\}$; e. g. with a *depth-first search*. From the construction of the graph the following observation is immediate:

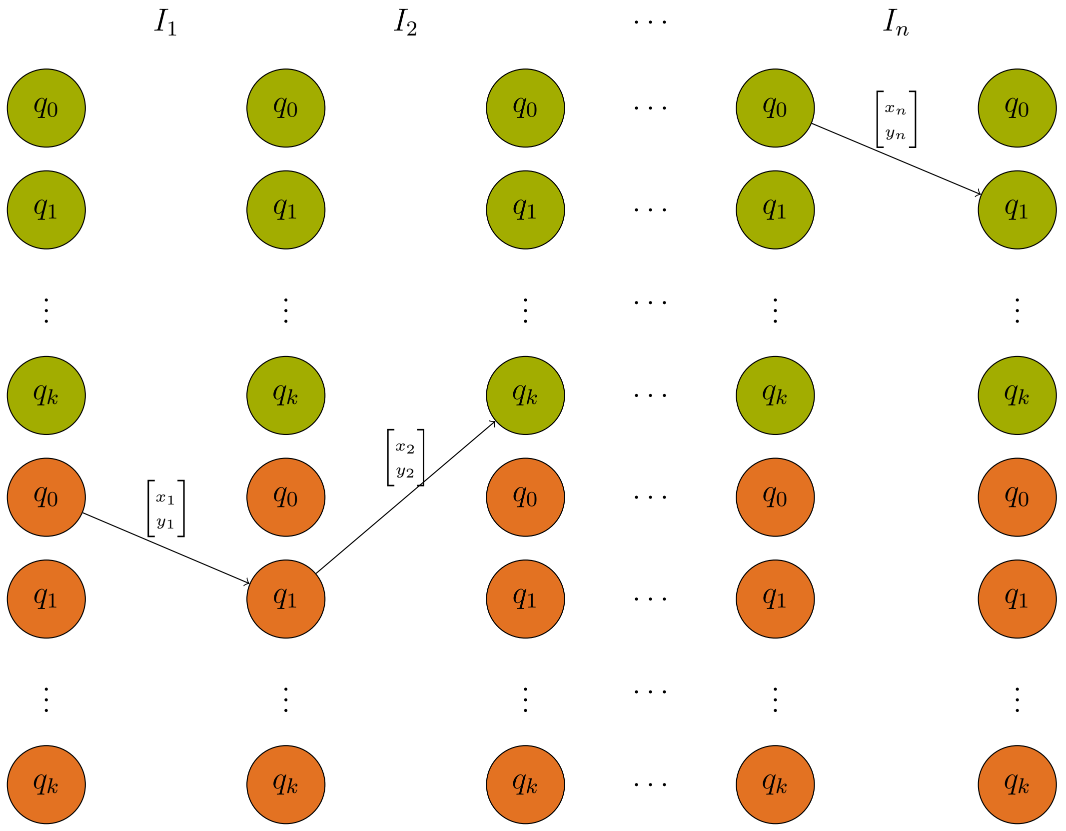
Lemma 3.4. *The annotations of any path which starts in* $\{0\} \times Q_0^{\mathcal{T}} \times \{0\}$ *and ends in* $\{n\} \times F_{\mathcal{T}} \times \{1\}$ *form a word* $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \ \dots \ \begin{bmatrix} x_n \\ y_n \end{bmatrix}$ *which strictly refines* I *and is accepted by* $\mathcal{T}$.

Since computing reachability in graphs is possible in polynomial time and the graphs are of polynomial size with respect to n and $|\mathcal{T}|$ we obtain the following corollary.

Corollary 3.1. *[Ras22; Kra23] Problem 3.1 for* $\mathcal{V}_{trap}$ *can be solved in polynomial time.*

Figure 3.3: *A graph to find a refining transition.*

Here we sketch the graph which we can use to find a refining transition for $I_1 \;\ldots\; I_n$. The columns of states correspond to the $n+1$ copies of the states of the automaton $\mathcal{T}$. The lower half of states (in orange) are those in which the bit is not set, while the upper half of states (in green) illustrate those copies in which the bit is set. Conceptionally, in each half, every column is connected with the next via the steps of $\mathcal{T}$. However, a step is not present if its target is part of the corresponding letter of the statement that is currently refined; e. g. $y_1 \notin I_1$, $y_2 \notin I_2$, and $y_n \notin I_n$. Moreover, if the origin of a step is part of the corresponding letter of the statement that is currently refined, then the step leads from the lower half to the upper half. Thus, $x_2 \in I_2$. Otherwise, steps only relate states from their respective halves. Therefore, $x_1 \notin I_1$. Whether x_n occurs in I_n is immaterial for the transition because it already is in the upper half.



3.3 Accelerate learning via topologies

Having established a general methodology to learn inductive statements for a regular transition system, we consider now how one can exploit the fact that we know the topology of the system to improve the learning procedure. Let us illustrate our idea with the ring topology first: Imagine the teacher provides one statement $I_1 \; \dots \; I_n \in \mathrm{Inductive}_{\mathcal{V}_{\mathit{trap}}}(\mathcal{R})$ to disprove a counterexample for some regular transition system which follows a ring topology. Consider the situation where $n > 1$. Because this statement must not be satisfied by the bad word of the counterexample, it must not contain the letter Σ. Because Lemma 2.26 shows that $\overline{\mathrm{Inductive}_{\mathcal{V}_{\mathit{trap}}}(\mathcal{R})}$ coincides with

$$\left\{ I \in \left(2^{\Sigma} \setminus \{\Sigma\}\right)^{*} \,\middle|\, \begin{array}{l} \text{there is } \langle A, B\rangle \in \mathrm{adj}(I) \\ \text{that is non-inductive for } \mathcal{V}_{\mathit{trap}} \end{array} \right\}$$

there is no $\langle A, B\rangle \in \mathrm{adj}(I_1 \; \dots \; I_n)$ that is non-inductive for $\mathcal{V}_{\mathit{trap}}$. Moreover, the set $\left\{ I' \in \left(2^{\Sigma}\right)^{*} \mid \mathrm{adj}(I') \subseteq \mathrm{adj}(I_1 \; \dots \; I_n) \right\}$ only contains inductive statements because we do not introduce any new pairs and, thus, no non-inductive ones. Consequently, we can generalize a single inductive statement to a language of inductive statements. Let us illustrate this in a concrete example next:

Example 3.2: *A generalization example for a flow statement.*

Recall the bow topology formulation of the running example from Example 2.2; that is, the regular transition system with initial language $t \; n^{*}$, and the transition language $\left(\begin{bmatrix} t \\ t \end{bmatrix} \mid \begin{bmatrix} n \\ n \end{bmatrix} \right)^{*} \begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \left(\begin{bmatrix} t \\ t \end{bmatrix} \mid \begin{bmatrix} n \\ n \end{bmatrix} \right)^{*}$. Imagine we try to disprove that there can be more than one token; i.e., we must not reach any word in $\Sigma^{*} \; t \; \Sigma^{*} \; t \; \Sigma^{*}$. Consider that, during the learning process, the counterexample $\begin{bmatrix} t \\ n \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix} \begin{bmatrix} n \\ t \end{bmatrix}$ is disproven with the flow $\{t\} \; \{t\} \; \{t\}$. Consequently, we know that $\{t\} \; \{t\} \; \{t\} \in \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$ and, by virtue of Lemma 2.32 we see that, with respect to $\mathcal{V}_{\mathit{flow}}$, $\langle \{t\}, \{t\} \rangle$ is compatible with P_R, P_L, and P_M for this bow. Invoking Lemma 2.32 once again, we may conclude that $\{t\}^{+} \subseteq \mathrm{Inductive}_{\mathcal{V}_{\mathit{flow}}}(\mathcal{R})$. In this fashion, we generalize a language of useful inductive statements from a single statement.

With the help of the characterizations of the language of all inductive statements from Section 2.7, we can formulate generalization procedures for all these topologies.

We will now formulate how to obtain from a single inductive statement in some topology a language of inductive statements. To ease presentation, we only generalize inductive statements that do not contain Σ as a letter. Lifting this restriction is straightforward.

Lemma 3.5. *Let $\mathcal{R}$ be a regular transition system and $I = I_1 \ \ldots \ I_n \in \mathrm{Inductive}_{\mathcal{V}}(\mathcal{R})$ such that $I_i \neq \Sigma$ for all $1 \leq i \leq n$. The following table contains, for a combination of a topology and an interpretation, languages that only contain inductive statements.*

Topology	$\mathcal{V}$	*Language*
Ring	$\mathcal{V}_{\mathit{trap}}$	$\left\{I' \in \left(2^{\Sigma}\right)^* \mid \mathrm{adj}(I') \subseteq \mathrm{adj}(I)\right\}$
	$\mathcal{V}_{\mathit{siphon}}$	$\left\{I' \in \left(2^{\Sigma}\right)^* \mid \mathrm{adj}(I') \subseteq \mathrm{adj}(I)\right\}$
	$\mathcal{V}_{\mathit{flow}}$	$\left\{I' \in \left(2^{\Sigma}\right)^* \mid \mathrm{adj}(I') \subseteq \mathrm{adj}(I)\right\}$
Bow	$\mathcal{V}_{\mathit{trap}}$	$\left\{ I' \in \left(2^{\Sigma}\right)^* \,\middle\vert\, \begin{array}{l} \mathrm{adj}_L(I') = \mathrm{adj}_L(I), \\ \mathrm{adj}_R(I') = \mathrm{adj}_R(I), \\ \mathrm{adj}_M(I') \subseteq \mathrm{adj}_M(I) \end{array} \right\}$
	$\mathcal{V}_{\mathit{siphon}}$	$\left\{ I' \in \left(2^{\Sigma}\right)^* \,\middle\vert\, \begin{array}{l} \mathrm{adj}_L(I') = \mathrm{adj}_L(I), \\ \mathrm{adj}_R(I') = \mathrm{adj}_R(I), \\ \mathrm{adj}_M(I') \subseteq \mathrm{adj}_M(I) \end{array} \right\}$
	$\mathcal{V}_{\mathit{flow}}$	$\left\{ I' \in \left(2^{\Sigma}\right)^* \,\middle\vert\, \begin{array}{l} \mathrm{adj}_L(I') = \mathrm{adj}_L(I), \\ \mathrm{adj}_R(I') = \mathrm{adj}_R(I), \\ \mathrm{adj}_M(I') \subseteq \mathrm{adj}_M(I) \end{array} \right\}$
k-Crowd	$\mathcal{V}_{\mathit{trap}}$	$\left\{I' \in \left(2^{\Sigma}\right)^* \mid \mathrm{occ}^{\leq k+1}(I') = \mathrm{occ}^{\leq k+1}(I)\right\}$
	$\mathcal{V}_{\mathit{siphon}}$	$\left\{I' \in \left(2^{\Sigma}\right)^* \mid \mathrm{occ}^{\leq k+1}(I') = \mathrm{occ}^{\leq k+1}(I)\right\}$
	$\mathcal{V}_{\mathit{flow}}$	$\left\{I' \in \left(2^{\Sigma}\right)^* \mid \mathrm{occ}^{\leq k+3}(I') = \mathrm{occ}^{\leq k+3}(I)\right\}$

Proof. These results follow immediately from the characterizations of (non-)inductive statements presented in Section 2.7. □

Moreover, we present in Section 2.7 how to construct automata for these languages of inductive statements. Consequently, one can immediately refine the abstraction with all inductive statements of these languages after having encountered a single inductive statement.

4 Implementation & Experiments

We have implemented the previously described verification methods in a prototype tool, called `dodo` [Wel23a; Wel23b]. In the following, we describe, first, which examples we consider, second, which verification procedures `dodo` supports and, third, the results of the procedures on the examples.

4.1 Case studies

To evaluate our prototype we consider a collection of 22 systems.

Dijkstra's algorithm for mutual exclusion

This example is based on a very early solution [Dij02] to the problem of mutual exclusion. Roughly speaking, we consider a group of agents which compete for a critical section. Every agent maintains a bit variable. With the help of one global pointer, this algorithm ensures mutual exclusion and a progress guarantee. We only check for the mutual exclusion property and whether the protocol can deadlock. Notably, the latter property is not equivalent to the progress guarantee. Also, the version we consider performs one atomic check over all other participants. This is a simplification of the original algorithm which includes an iterative check. Due to the restrictions of modeling the protocol as a regular transition system, this simplification is necessary.

Dijkstra's algorithm for mutual exclusion with a token

This example is based on [FO97] and models a mutual exclusion algorithm for agents that form a ring and pass around a single token as a semaphore for a critical region.

Other mutual exclusion algorithms

Additionally, we also consider the mutual exclusion algorithms of Burns [JL98] and Szymanski [AHH16]. Also, we consider the standard bakery algorithm (as formalized in [Che+17]).

Dining philosophers

We consider three variants of the dining philosophers. First, the atomic version that we already illustrated in Example 2.3. Second, the version in which one philosopher grabs their forks in a different order than all the others. And, lastly, a version that allows philosophers to put down their fork again if they only grabbed the first one [LR81]. For all these versions we only prove that they cannot deadlock.

Cache coherence protocols

The central property to check for cache coherence protocols is that there are no two different versions of the same data point present in the cache. We consider the protocols MESI, MOESI, Illinois, Berkeley, Synapse, FutureBus+, Dragon, and Firefly. For all these protocols we consider various custom safety properties. The models are based on [Del00a].

Termination detection

Based on [DS80], we consider a linear host of agents where a token moves down and up the line again to check whether all the agents have finished some computation. Here we check whether at most one token moves up or down the line.

Dining cryptographers

In this model (which follows [Che+17]), we consider a group of cryptographers sitting around a circular table. They just shared a meal that was paid for by an anonymous person. This person might be one of the cryptographers or not. Now, they try to figure out whether one of them paid without revealing the actual person. To this end, they run the following protocol: each of them throws a coin, compares their coin with the coin of their right-hand neighbor, and announces whether both coins show the same side or a difference. However, a cryptographer who paid for the meal will lie in their announcement. It turns out that an even number of announcements that state that the coins show different sites imply that none of the cryptographers paid while an odd number of these announcements give away that one of them paid the meal but does not reveal who. We verify that this protocol cannot yield an even number of disagreeing announcements with a paying cryptography and, on the other hand, we also check that there is not an odd number of disagreeing announcements without a paying cryptographer.

Leader election

We also include two leader election algorithms, attributed to Herman, and Israeli and Jafon. The formalizations are taken from [Che+17].

Token passing

Finally, we include the running examples of this thesis that model token passing algorithms. That is, Example 2.2 and, its variant from Example 2.9.

4.2 Verification procedures

`dodo` supports the three concrete interpretations we have considered throughout the thesis; namely, $\mathcal{V}_{trap}$, $\mathcal{V}_{siphon}$, and $\mathcal{V}_{flow}$. Because all approaches are compositional; that is, one can intersect the over-approximations of different interpretations to obtain one refined over-approximation, `dodo` allows to specify any subset $\boldsymbol{V} \subseteq \{\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}\}$ of interpretations to use. Based on the chosen interpretations `dodo` can be operated in three different modes: `oneshot`, `learn`, and `adaptive`. Roughly speaking, `oneshot` constructs the abstraction of all inductive statements for the given RTS and checks, on that basis, whether any undesired state can be reached. For $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$, the construction of Section 2.6 is used while for $\mathcal{V}_{flow}$ the generic construction (cp. Section 2.3) is used. On the other hand, `learn` and `adaptive` employ the methodology described in Chapter 3. The difference between the two modes is that the latter generalizes statements based on the topology of the system while the former does not. Therefore, one would expect better performance from the latter because it identifies more inductive statements from the same information.

The experiments are run on `openjdk 19.0.2` with a maximum heap size of 10 `GiB`.

The central processing unit identifies itself as `Intel(R) Core(TM) i5-9500TE CPU @ 2.20GHz`. For every combination of RTS and property, we call `dodo` (in the considered mode) with all possible $\boldsymbol{V} \subseteq \{\mathcal{V}_{\mathit{trap}}, \mathcal{V}_{\mathit{siphon}}, \mathcal{V}_{\mathit{flow}}\}$. However, we do so gradually. That is, we start with $\boldsymbol{V} = \{\mathcal{V}_{\mathit{trap}}\}$, $\boldsymbol{V} = \{\mathcal{V}_{\mathit{siphon}}\}$, and, then, $\boldsymbol{V} = \{\mathcal{V}_{\mathit{flow}}\}$. Afterward, we proceed to all possible sets with two elements and, finally, to the whole set $\boldsymbol{V} = \{\mathcal{V}_{\mathit{trap}}, \mathcal{V}_{\mathit{siphon}}, \mathcal{V}_{\mathit{flow}}\}$. If `dodo` already succeeded in establishing the property with some subset $\boldsymbol{V}$ (or failed to do so because it exhausted computational limits) we do not consider supersets of $\boldsymbol{V}$ anymore. For every call to `dodo` we set a timeout of 20 minutes.

`oneshot`

In this mode, `dodo` constructs a transducer for the relation

$$\mathit{Id}(\mathcal{I}) \circ \left(\bigcap_{\mathcal{V} \in \boldsymbol{V}} \Rrightarrow_{\mathcal{V}} \right) \circ \mathit{Id}(\mathcal{B})$$

and checks whether this transducer accepts any word. This transducer is explored lazily. Specifically, the elements of the step relation of the transducer are only computed if necessary. `dodo` uses the automata library `AutomataLib` [IHS15] to represent the automata it constructs.

If the interpretations $\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$ are chosen, then `dodo` relies on the step game to compute the step relation. In this game, when systematically exploring the winning strategies[1], it is possible to consider the same game state $\langle \ell, S, r \rangle$ via two different histories because the order in which the elements of S are removed might differ. Because, in both cases, the same strategies can be used for the game, we implemented a caching mechanism for these situations. The idea of this caching mechanism is to trade memory for time. We report results for experiments using it and results for not using it.

The complete data can be found in Appendix A. In the following, we only present statistics on this dataset.

Effectiveness: 178 calls to `dodo` using the mode `oneshot` failed either to timeouts (122) or memory issues (56). Of the remaining 165 calls 58 were successful while 107 could not establish the property but provided counterexamples which witness that

---

[1] `dodo` does not solve the step game for all possible steps but computes, for any given base column b and letter $\begin{bmatrix} u \\ v \end{bmatrix}$, all base columns b' for which the step game can be won. The argument, however, translates to the actual implementation.

the abstraction is not sufficient to establish the property at all. Overall, 31 properties out of 62 could be established. Unfortunately, for 11 properties, no answer could be given because all calls either timed out or ran into memory issues.

Interpretations: In the successful 58 instances, 12 were using $\mathcal{V}_{trap}$ without the caching mechanism and 26 with the caching mechanism. Only 1 successful instance was obtained using $\mathcal{V}_{siphon}$ without the caching mechanism and 4 with caching. The interpretation $\mathcal{V}_{flow}$ was used in 15 successful calls. In general, using the caching mechanism made 34 calls return an answer which did not return one without it. In only one case it was the other way around. Moreover, in 16 cases using the caching mechanism sped up the process by more than half a second. There are no cases where not using the caching mechanism shortened the time until a result was found by more than half a second.

Efficiency: Using the `oneshot` mode, a counterexample or the definite absence of one was reported, on average, in 20 seconds. The longest `dodo` took in this mode was 7 minutes and 13 seconds. This call returns a negative result using the interpretations $\mathcal{V}_{siphon}$ and $\mathcal{V}_{flow}$. Anecdotally, the same call can be executed in 4.4 seconds if one activates the caching mechanism for the abstraction of inductive statements for $\mathcal{V}_{siphon}$.

Overall, the `oneshot` mode is not competitive. Only half the properties can be established and more than half of the executions run out of resources. However, the caching mechanism seems to have an overall positive effect. Since the learning approaches are specifically designed to use less memory, we hope to achieve better results with them.

`learn`

Based on the library `LearnLib` [IHS15] we employ an off-the-shelf learning algorithm (specifically L^* [Ang87]) using the oracles from Chapter 3. The alphabet of the language that we learn is considerably large; i. e. exponentially larger than the alphabet of the RTS. `LearnLib` supports starting a learning process with some alphabet which can be expanded later if necessary. Therefore, we start the learning process with an empty alphabet and gradually add those letters from 2^{Σ} which occur in solutions for Problem 3.1. We illustrate in Algorithm 2 how `dodo` uses all interpretations in $\boldsymbol{V}$ simultaneously. Roughly speaking, in this mode `dodo` maintains a transducer $\mathcal{A}$ for the current over-approximation of the reachability relation. Every time this over-approximation proves not sufficient, `dodo` iterates over all interpretations until it finds $\mathcal{V}$ for which an inductive statement I exists that disproves the counterexample. The learner $\mathcal{P}$ for $\mathcal{V}$ is refined by teaching it that I should be part of its language. This prompts $\mathcal{P}$ to update its hypothesis. Then, $\mathcal{P}$ is presented with all non-inductive statements its new hypothesis contains until it only accepts inductive statements. For this updated hypothesis the induced over-approximation $\mathcal{H}$ is computed and $\mathcal{A}$ is updated to accept the intersection of the languages of $\mathcal{H}$ and itself.

```
Input: RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$, NFA $\mathcal{B}$ and interpretations $\boldsymbol{V}$.
Output: ✓ or ×
begin
    $\mathcal{A} \leftarrow$ getAutomatonFor($Id(\mathcal{I}) \circ Id(\mathcal{B})$);
    while $\mathcal{L}(\mathcal{A}) \neq \emptyset$ do
        counterExample $\leftarrow$ getWordFrom($\mathcal{L}(\mathcal{A})$);
        foreach $\mathcal{V} \in \boldsymbol{V}$ do
            $I \leftarrow$ disproveWithInterpretation(counterExample, $\mathcal{V}$);
            if $I \neq$ null then
                $\mathcal{P} \leftarrow$getLearnerForInterpretation($\mathcal{V}$);
                teach($\mathcal{P}$, $I$);
                removeNonInductive($\mathcal{P}$);
                $\mathcal{A} \leftarrow$ getAutomatonFor($\mathcal{L}(\mathcal{A}) \cap$getAbstraction($\mathcal{P}$));
                continue outer loop;
            end
        end
        return ×;
    end
    return ✓;
end
```

Algorithm 2: `dodo`'s learning algorithm for multiple interpretations.

We use `SAT4j` [BP10] as SAT solver to solve Problem 3.1 for $\mathcal{V}_{flow}$. For $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$, we implemented the algorithm for Problem 3.1 that runs in polynomial time. `dodo` can also solve Problem 3.1 for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ via `SAT4j` using a similar encoding as for $\mathcal{V}_{flow}$.

Again, the complete data is reported in Appendix B and we only report the statistics of it here.

Effectiveness: In this mode we did not encounter any memory issues and only 4 calls timed out. All of the remaining 461 instances return definite answers of which 137 were positive ones. In this way, 50 of 62 properties could be established.

Interpretations: From the successful 137 calls, 48 used $\mathcal{V}_{trap}$ (both in the version where Problem 3.1 is solved with a polynomial time algorithm and by formulating it as a propositional formula), 11 used $\mathcal{V}_{siphon}$ (again, in both versions), and 21 used $\mathcal{V}_{flow}$. Surprisingly, in exactly one case formulating Problem 3.1 as a propositional formula led to a (negative) result where the version that uses the polynomial time algorithm timed out. This happened for the system with the largest alphabet of 50 symbols and the interpretation $\mathcal{V}_{trap}$. In the polynomial time algorithm for Problem 3.1 a depth-first search in a graph is executed, in the worst case, $n \cdot (|\Sigma| - 1)$ times where n is the length of the words that need to be separated. Moreover, the number of edges in the graph is, potentially, quadratic in $|\Sigma|$. Therefore, the polynomial time algorithm for Problem 3.1 is, in the worst case, bounded by $|\Sigma|^3$. Also, by its definition, it always computes the weakest inductive statement for $\mathcal{V}_{trap}$[2]. For this particular case, it is checked whether one can reach a state in which no transition can be executed anymore. The NFA which captures these bad configurations has 281 states.

We believe the reason that `dodo` times out here is a combination of these factors: The individual letters that are used for the learned hypotheses are insufficient to formulate inductive statements that dismiss many bad configurations at once because only the weakest inductive statements are considered as solutions for Problem 3.1. Hence, Problem 3.1 is solved many times which takes, due to the large alphabet size, too long.

On the other hand, formulating Problem 3.1 as a propositional formula leads to any (not necessarily the weakest) inductive statement that separates the counterexample. The letters of this inductive statement potentially allow for more expressive inductive statements in the hypotheses which leads to the counterexample which cannot be dismissed faster.

The opposite case; that is, using the polynomial time algorithm to solve Problem 3.1 allowed to compute a result while using the reduction to SAT timed out, did not occur. However, for the cases where both methods compute a result, using

---

[2] In particular, the definition of a separator renders it the weakest inductive statement for the interpretation $\mathcal{V}_{trap}$ because it is the union of all inductive statements that the bad configuration does not satisfy.

the polynomial time algorithm sped up the computation time by more than half a second in 14 cases. On the other hand, in 15 cases it was the other way around.

Efficiency: The average computing time in this mode was 12.9 seconds. The longest computation took 19 minutes and 10 seconds – just shy of the timeout of 20 minutes.

Comparison of `oneshot` and `learn`

For this comparison, we only consider, for `dodo`'s mode `oneshot`, calls that use the caching mechanism because it performs better. Similarly, we restrict ourselves to calls that solve Problem 3.1 with polynomial-time algorithms (if possible) for the mode `learn`.

This leaves 175 cases for the same instance of RMC and used interpretations that differ only in the mode. For 94 of these cases, the mode `oneshot` did not return a result but the mode `learn` did. There is no case where `learn` did not return a result but `oneshot` did.

There are 77 cases where both modes returned a result. For these, there are 46 cases where the explored states of the transducer that capture the potential reachability relation were fewer in the mode `oneshot` than in the mode `learn`. On average, in these cases, the number of states of the transducer that are explored in the mode `oneshot` to determine a result is only half of the number of states of the transducer that is constructed in the mode `learn`. It was the other way around in 30 cases. Here, the number of explored states in the transducer that is constructed in `learn` is around a third of the number of states that are explored in `oneshot`. If we consider one mode to be faster than the other if it improves the running time by more than half a second, then `learn` outperforms `oneshot` in 47 cases. There is no case where it is the other way around.

One significant difference between the two modes is that `oneshot` constructs its (potentially nondeterministic) transducer lazily while `learn` not only constructs the whole transducer but also constructs a deterministic one. Moreover, `learn` does not look for a minimal transducer to capture its over-approximation. In fact, every time this over-approximation is refined `dodo` does so by intersecting its language with another over-approximation – an operation that requires a product construction. Therefore, we expect the abstractions of `oneshot` to be smaller in cases where a negative result is returned because these cases only require constructing a part of the transducer. For positive results, both modes construct complete transducers for the respective over-approximations. Although `learn` does not optimize for a small transducer of its over-approximation, it, generally, performs well. Thus, we are interested in whether its abstraction is smaller than for the mode `oneshot` in these cases.

There are 51 cases where both modes returned a negative result. For these, the explored states of the over-approximation are fewer in `oneshot` than in `learn` in 40 cases. On average, in these 40 cases, the number of explored states is halved in the mode `oneshot`. `learn` explored fewer states in 11 cases; that is, on average, only 42% the number of states as `oneshot`. Regardless, the mode `learn` produces a result faster than the mode `oneshot` in 33 cases. Recall from before that `oneshot` never outperforms `learn`.

There are 26 cases where both modes returned a positive result. For these, the states of the transducer that captures the over-approximation are fewer in `oneshot` than in `learn` in 6 cases. On average, in these 6 cases, the number of states is only 69% of the states in `learn`. `learn` explored fewer states in 19 cases; that is, on average, only 31% the number of states as `oneshot`. The mode `learn` produces a result faster than the mode `oneshot` in 14 cases.

This data suggests that the size of the transducer of the over-approximation does not seem to be the crucial factor for the runtime of the tool. In particular, `oneshot` can half the number of states that it explores of its over-approximation (for negative results) but is not significantly faster in any of these cases. However, `learn` requires only, on average, 8% of the possible alphabet for its inductive statements across these 175 cases. The alphabets that encode all possible inductive statements grow exponentially in the size of the alphabet for the considered RTS. `oneshot` uses these alphabets either explicitly (for $\mathcal{V}_{\mathit{flow}}$) or implicitly (for $\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$). Thus, significantly reducing the sizes of the alphabets might also contribute to the good performance of `learn`.

In conclusion, learning only some sufficient part of all inductive statements mitigates most of the memory issues and, also, improves the computation time. Moreover, in this mode `dodo` can be considered a useful tool for RMC because the provided generic interpretations already suffice to establish many properties. Although the run time in this mode is not prohibitive, we still strive for improvements with the remaining mode `adaptive`.

`adaptive`

This final mode is essentially the same as `learn`. The only difference is, that statements which are used to discharge a counterexample are also generalized, if possible. These generalizations are regular languages of inductive statements. These languages induce a potential reachability relation on their own. Therefore, instead of teaching these languages to the learner, one can immediately refine the over-approximation with this induced potential reachability relation and not update the learner. In this way, the learner is only responsible for those inductive statements that cannot be generalized.

The data for `adaptive` is also given in Appendix B.

Effectiveness: Complementing the learning algorithm with generalizations led to 1 call (out of 388) which runs out of memory and 15 which run out of time. The remaining 72 instances can be separated into 95 successful ones and 277 failing ones. Because this mode is only applicable to RTSs with specific topologies, we consider 50 properties. Of these, 36 can be established. For 2 properties no call returned any answer.

Interpretations: In the 95 successful calls, 33 used $\mathcal{V}_{trap}$ and solved Problem 3.1 with the polynomial time algorithm, 33 used $\mathcal{V}_{trap}$ and solved Problem 3.1 with the reduction to SAT, 8 used $\mathcal{V}_{siphon}$ and solved Problem 3.1 with the polynomial time algorithm, 8 used $\mathcal{V}_{siphon}$ and solved Problem 3.1 with the reduction to SAT, and 13 used $\mathcal{V}_{flow}$. In 1 case using the polynomial time algorithm for Problem 3.1 made computing an answer possible over the embedding into a propositional formula. However, in 1 case it was the other way around. Embedding into a propositional formula sped up the computation by more than half a second in 2 cases. The polynomial time algorithm led to a speedup of more than half a second in 3 cases.

Efficiency: The average computing time in this mode was 1.5 seconds. The longest computation took 2 minutes and 3 seconds.

Comparison of `adaptive` and `learn`

Overall, `adaptive` speeds up the computation of answers in some cases. However, the effect is less than expected. In fact, in 26 cases this mode performs significantly[3] better

---

[3] Again, that means a speedup of more than half a second.

than `learn`. On average, in these 26 cases, using `adaptive` sped up the computation by 73 seconds. On the other hand, it is the other way around in 5 cases where `adaptive` was, on average, 50 seconds slower and both modes show roughly the same behavior in 343 cases. Moreover, there are 4 properties that can be established using `learn` but not with `adaptive`. Notably, the systems in these 4 cases are all rings or bows. Thus, an adaptive learning approach does not seem particularly suited for these topologies.

In conclusion, `learn` seems to be the best approach, because `oneshot` struggles with memory and time issues and `adaptive` needs, for every combination of interpretation and topology, a generalization result – which also not necessarily aides the performance. In particular, `learn` is agnostic to the used interpretation but offers the most robust behavior of all three modes with only 4 failed calls overall.

4.3 Qualitative comparison with other approaches

In the following, we compare our approach with others from the literature. In particular, we are interested in its expressiveness; that is, whether the over-approximation induced by inductive statements for the generic interpretations $\mathcal{V}_{trap}$, $\mathcal{V}_{siphon}$, and $\mathcal{V}_{flow}$ suffices to establish interesting properties. Some tools from the literature are not publicly available (anymore). This poses the problem that one can consider various safety properties for some RTS and it is not precisely reported which properties are checked. Therefore, we say an RTS is "positive" for our approach if we can establish at least half of the properties that we consider for it[4].

In the following, we report, for every approach, how many RTSs are considered there which we also consider in this thesis and for how many of those we report positive results. Since there are no negative results reported in the literature, all approaches yield a positive result for all RTSs that are checked with them.

[Nei14] For the 4 RTSs both approaches consider, we can report positive results in 4 cases.

[Che+17] For the 10 RTSs both approaches consider, we report positive results in 9 cases.

[Var06] For the 8 RTSs both approaches consider, we report positive results in 6 cases.

---

[4] This is only relevant for the cache coherence protocols *Dragon*, *FutureBus*, and *Berkeley*.

[Abd+07] For the 11 RTSs both approaches consider, we report positive results in 8 cases.

We only report data on the three generic interpretations that we introduced. Therefore, if we do not report a positive result for some RTS, then it is not impossible to obtain a positive result with our methodology because one can consider other interpretations. Regardless, since these generic interpretations ($\mathcal{V}_{trap}$ in particular) already seem to perform reasonably well in comparison, we are confident to state that our approach is a competitive paradigm for RMC.

5 Conclusion

In this thesis, we have introduced a new paradigm for regular model checking in the form of logical statements that are encoded using interpretations. This paradigm streamlines (and extends) previous work on parameterized systems [Boz+20; ERW21b; ERW22b]. Moreover, it is the basis for a useful analysis tool for regular model checking.

Additionally, it raises many interesting theoretical questions. We provide answers to some of the questions that are relevant to the immediate application of this paradigm. In particular, we show that Problem 2.2 is PSpace-complete for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$. For the interpretation $\mathcal{V}_{flow}$, we show that the problem is PSpace-hard and in ExpSpace.

In the context of learning a sufficient set of inductive statements, we have considered the naturally occurring word problem (Problem 3.1). Here, the problem is in PTime for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ and NP-complete for $\mathcal{V}_{flow}$.

In Remark 2.3, we note that Problem 2.2 is, if the interpretation is part of the input, ExpSpace-complete. However, Chapter 3 formulates a learning approach with constructions that do not rely on a specific interpretation. Moreover, the implementation of this learning approach which only uses generic constructions performs well, experimentally (cp. Chapter 4). Therefore, we do not believe the (theoretically) high complexity to be prohibitive for further research on the application of this approach. As is often the case, pathological cases might be rare in practice.

5.1 Future work

We want to conclude this thesis with a list of open problems.

Complexity gap for $\mathcal{V}_{flow}$ Problem 2.2 is PSpace-hard for $\mathcal{V}_{flow}$ and solvable in ExpSpace. The exact complexity, however, is still an open question. Solving Problem 2.2 for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ in PSpace crucially depends on two factors:

- the inductive statements for these abstractions can be merged to produce a canonical weakest statement which is used to compute the separator for any configuration, and
- how to compute the letters of the separator can be formulated as a transformation on permutations of the states of the transducer.

For the interpretation $\mathcal{V}_{\mathit{flow}}$, the first factor already is not true anymore (and, consequently, the second one is immaterial). Thus, solving Problem 2.2 for $\mathcal{V}_{\mathit{flow}}$ in PSPACE, if possible at all, probably requires a different approach than for $\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$.

Complexity jump for $\mathcal{V}_{\mathit{flow}}$ The word problem (Problem 3.1) for $\mathcal{V}_{\mathit{flow}}$ has a higher complexity than for $\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$. The interpretations $\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$ are reachability and safety automata, respectively. $\mathcal{V}_{\mathit{flow}}$, on the other hand, is neither. Thus, we ask whether there are structural properties of the interpretation that determine the complexity of the word problem. Moreover, solving the word problem for $\mathcal{V}_{\mathit{trap}}$ and $\mathcal{V}_{\mathit{siphon}}$ in PTIME relies on the construction for the parameterized case. Can we establish a general connection between the complexity of the parameterized case and the word case (or the other way around)?

Improving tooling In [Boz+20] a tool `ostrich` is built on top of the WS1S solver `MONA` [Hen+95] which is effectively equivalent with `dodo`'s mode `oneshot`. Although the specification language for the parameterized systems does not match the whole expressiveness of RMC, the performance is significantly better. This raises the question of whether the difference in performance is due to the restricted specification language or the refined engineering of `MONA`. In particular, [Hen+95] argues that encoding the step function of `MONA`'s automata with binary decision diagrams is a crucial part of the design since it allows to deal with large alphabets for the automata.

The defining factor for the size of the alphabet of the automaton that recognizes the inductive statements is the size of the alphabet Σ of the analyzed RTS $\mathcal{R}$. On the other hand, the number of states of the automaton that recognizes the inductive statements (or, respectively, the transducer of the over-approximation for $\mathcal{V}_{\mathit{trap}}$ or $\mathcal{V}_{\mathit{siphon}}$) is determined by the size of the transducer $\mathcal{T}$ of $\mathcal{R}$. Previously, we hypothesized that the size of the alphabets that encode the inductive statements is a limiting factor for the mode `oneshot` (rather than the size of the transducer that captures the over-approximation of the reachability relation). In Figure 5.1, we plot calls to the mode `oneshot` that are successful, unsuccessful, or exceed the time limit, against the size of Σ and $\mathcal{T}$. Based on this rough visualization, neither factor can be identified as the primary restriction.

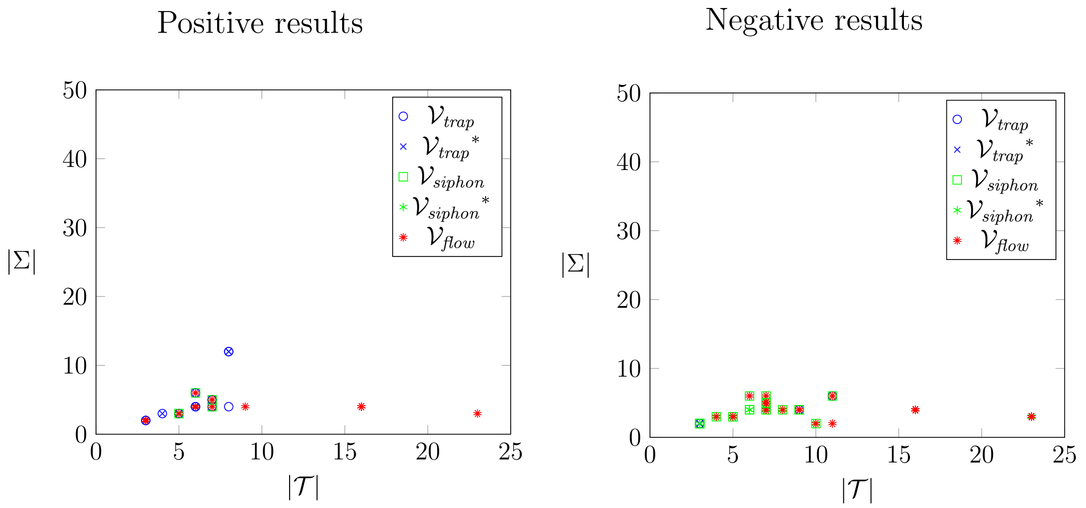
Therefore, an in-depth comparison of both tools should be considered in the future to identify and (possibly) remove limiting factors for the mode `oneshot`. Additionally, there are more avenues to consider for the further development of `dodo`:

- For instance, one should evaluate whether regularly minimizing $\mathcal{A}$, the over-approximation induced by all learned inductive statements, in Algorithm 2 improves the performance of the approach (and, if so, in what frequency one should minimize $\mathcal{A}$).
- The mode `oneshot` can be run by only considering inductive statements that can be encoded by some $\Gamma' \subseteq \Gamma$ instead of using all of Γ. For the case, $\Gamma = 2^\Sigma$ one could gradually increase a value $k = 1, 2, \ldots$ and only consider the corresponding alphabets $\left\{I \in 2^\Sigma \mid |I| \leq k\right\}$. On the other hand, one could separate some randomly selected initial and bad configurations via inductive statements of the interpretation and take the letters from these statements to perform the mode `oneshot`.
- Experimentally, the interpretation $\mathcal{V}_{\mathit{flow}}$ seems to be primarily useful in systems that operate with some sort of unique access token and the interpretation $\mathcal{V}_{\mathit{siphon}}$ does not perform well at all. If one wanted to translate this paradigm into a tool that goes beyond a prototype, then one should evaluate more interpretations to identify the top-performing ones. There might even be a notion of interpretations that "fit" the transducer of the interpretations which are then selected by the tool to analyze the given RTS.

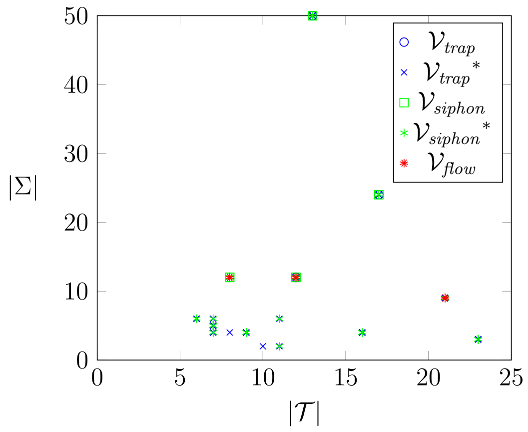
Learn more There are various other automata learning algorithms than L^* [HS18]. This means one can consider other algorithms for the mode `learn` of `dodo`. In particular, the learned language of inductive statements is a certificate of the correctness of the property. Keeping this certificate as small as possible allows us to present it to a user. If this certificate is small enough, a user might be able to verify it by hand which vastly improves confidence in the result. Moreover, it may contain non-inductive statements, which hint at an incorrect formalization of the system.

Figure 5.1: *Qualitative analysis of results in* `oneshot`.

In these scatterplots, we consider calls to the mode `oneshot` for a single interpretation. In particular, we plot the number of states of the transducer and the size of the alphabet for the considered RTS. Here, $\mathcal{V}_{trap}{}^*$ and $\mathcal{V}_{siphon}{}^*$ denote using the interpretations $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$, respectively, with the caching mechanism.



Calls that exceeded the time limit



Bibliography

[Abd+02] Parosh Aziz Abdulla et al. “Regular Model Checking Made Simple and Efficient”. In: *CONCUR*. Vol. 2421. Lecture Notes in Computer Science. Springer, 2002, pp. 116–130 (cit. on p. 1).

[Abd+04] Parosh Aziz Abdulla et al. “A Survey of Regular Model Checking”. In: *CONCUR*. Vol. 3170. Lecture Notes in Computer Science. Springer, 2004, pp. 35–48 (cit. on pp. 1, 2, 15, 16, 18).

[Abd+07] Parosh Aziz Abdulla et al. “Regular Model Checking Without Transducers (On Efficient Verification of Parameterized Systems)”. In: *TACAS*. Vol. 4424. Lecture Notes in Computer Science. Springer, 2007, pp. 721–736 (cit. on p. 146).

[Abd+12] Parosh Aziz Abdulla et al. “Regular model checking for LTL(MSO)”. In: *Int. J. Softw. Tools Technol. Transf.* 14.2 (2012), pp. 223–241 (cit. on p. 1).

[Abd12] Parosh Aziz Abdulla. “Regular model checking”. In: *Int. J. Softw. Tools Technol. Transf.* 14.2 (2012), pp. 109–118 (cit. on pp. 1, 2, 15, 16, 18).

[AHH16] Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. “Parameterized verification through view abstraction”. In: *Int. J. Softw. Tools Technol. Transf.* 18.5 (2016), pp. 495–516 (cit. on p. 136).

[Ang87] Dana Angluin. “Learning Regular Sets from Queries and Counterexamples”. In: *Inf. Comput.* 75.2 (1987), pp. 87–106 (cit. on pp. 117, 119, 139).

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008 (cit. on p. 1).

[Blo+16] Roderick Bloem et al. “Decidability in Parameterized Verification”. In: *SIGACT News* 47.2 (2016), pp. 53–64 (cit. on pp. 6, 24).

[Boi12] Bernard Boigelot. “Domain-specific regular acceleration”. In: *Int. J. Softw. Tools Technol. Transf.* 14.2 (2012), pp. 193–206 (cit. on p. 1).

[Bou+12] Ahmed Bouajjani et al. “Abstract regular (tree) model checking”. In: *Int. J. Softw. Tools Technol. Transf.* 14.2 (2012), pp. 167–191 (cit. on p. 1).

[Boz+20] Marius Bozga et al. “Structural Invariants for the Verification of Systems with Parameterized Architectures”. In: *TACAS (1).* Vol. 12078. Lecture Notes in Computer Science. Springer, 2020, pp. 228–246 (cit. on pp. 9, 10, 37, 147, 148).

[BP10] Daniel Le Berre and Anne Parrain. “The Sat4j library, release 2.2”. In: *J. Satisf. Boolean Model. Comput.* 7.2-3 (2010), pp. 59–6 (cit. on p. 140).

[BT12] Ahmed Bouajjani and Tayssir Touili. “Widening techniques for regular tree model checking”. In: *Int. J. Softw. Tools Technol. Transf.* 14.2 (2012), pp. 145–165 (cit. on p. 1).

[CES09] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. “Model checking: algorithmic verification and debugging”. In: *Commun. ACM* 52.11 (2009), pp. 74–84 (cit. on p. 1).

[Che+17] Yu-Fang Chen et al. “Learning to prove safety over parameterised concurrent systems”. In: *FMCAD.* IEEE, 2017, pp. 76–83 (cit. on pp. 4, 117, 136, 137, 145).

[Del00a] Giorgio Delzanno. “Automatic Verification of Parameterized Cache Coherence Protocols”. In: *CAV.* Vol. 1855. Lecture Notes in Computer Science. Springer, 2000, pp. 53–68 (cit. on pp. 85, 136).

[Del00b] Giorgio Delzanno. “Automatic Verification of Parameterized Cache Coherence Protocols”. In: *CAV.* Vol. 1855. Lecture Notes in Computer Science. Springer, 2000, pp. 53–68 (cit. on p. 107).

[Dij02] Edsger W. Dijkstra. “Cooperating Sequential Processes”. In: *The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls.* Ed. by Per Brinch Hansen. New York, NY: Springer New York, 2002, pp. 65–138. ISBN: 978-1-4757-3472-0. DOI: `10.1007/978-1-4757-3472-0_2`. URL: `https://doi.org/10.1007/978-1-4757-3472-0_2` (cit. on p. 135).

[DR12] Giorgio Delzanno and Ahmed Rezine. “A lightweight regular model checking approach for parameterized systems”. In: *Int. J. Softw. Tools Technol. Transf.* 14.2 (2012), pp. 207–222 (cit. on p. 1).

[DS80] Edsger W. Dijkstra and Carel S. Scholten. “Termination Detection for Diffusing Computations”. In: *Inf. Process. Lett.* 11.1 (1980), pp. 1–4 (cit. on p. 136).

[EGK12] Javier Esparza, Andreas Gaiser, and Stefan Kiefer. “Proving Termination of Probabilistic Programs Using Patterns”. In: *CAV*. Vol. 7358. Lecture Notes in Computer Science. Springer, 2012, pp. 123–138 (cit. on p. 20).

[EM00] Javier Esparza and Stephan Melzer. “Verification of Safety Properties Using Integer Programming: Beyond the State Equation”. In: *Formal Methods Syst. Des.* 16.2 (2000), pp. 159–189 (cit. on p. 37).

[ERW21a] Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. “Abduction of trap invariants in parameterized systems”. In: *GandALF*. Vol. 346. EPTCS. 2021, pp. 1–17 (cit. on pp. 9, 10).

[ERW21b] Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. “Computing Parameterized Invariants of Parameterized Petri Nets”. In: *Petri Nets*. Vol. 12734. Lecture Notes in Computer Science. Springer, 2021, pp. 141–163 (cit. on pp. 9–11, 37, 147).

[ERW22a] Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. “Computing Parameterized Invariants of Parameterized Petri Nets”. In: *Fundam. Informaticae* 187.2-4 (2022), pp. 197–243 (cit. on pp. 10, 85).

[ERW22b] Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. “Regular Model Checking Upside-Down: An Invariant-Based Approach”. In: *CONCUR*. Vol. 243. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 23:1–23:19 (cit. on pp. 10, 37, 67, 85, 147).

[ERW22c] Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. “Regular Model Checking Upside-Down: An Invariant-Based Approach”. In: *CoRR* abs/2205.03060 (2022) (cit. on pp. 10, 11).

[Esp+14] Javier Esparza et al. “An SMT-Based Approach to Coverability Analysis”. In: *CAV*. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 603–619 (cit. on p. 37).

[FO97] Laurent Fribourg and Hans Olsén. “Reachability sets of parameterized rings as regular languages”. In: *INFINITY*. Vol. 9. Electronic Notes in Theoretical Computer Science. Elsevier, 1997, p. 40 (cit. on pp. 85, 135).

[Grä20] Erich Grädel. “Automatic Structures: Twenty Years Later”. In: *LICS*. ACM, 2020, pp. 21–34 (cit. on p. 5).

[Hen+95] J.G. Henriksen et al. “Mona: Monadic Second-order logic in practice”. In: *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS ’95, LNCS 1019*. 1995 (cit. on p. 148).

[HMU07] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007 (cit. on p. 14).

[Hon+19] Chih-Duo Hong et al. “Probabilistic Bisimulation for Parameterized Systems - (with Applications to Verifying Anonymous Protocols)”. In: *CAV (1)*. Vol. 11561. Lecture Notes in Computer Science. Springer, 2019, pp. 455–474 (cit. on p. 1).

[HS18] Falk Howar and Bernhard Steffen. “Active Automata Learning in Practice - An Annotated Bibliography of the Years 2011 to 2016”. In: *Machine Learning for Dynamic Software Analysis*. Vol. 11026. Lecture Notes in Computer Science. Springer, 2018, pp. 123–148 (cit. on p. 149).

[IHS15] Malte Isberner, Falk Howar, and Bernhard Steffen. “The Open-Source LearnLib - A Framework for Active Automata Learning”. In: *CAV (1)*. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 487–495 (cit. on pp. 138, 139).

[JL98] Henrik Ejersbo Jensen and Nancy A. Lynch. “A Proof of Burns N-Process Mutual Exclusion Algorithm Using Abstraction”. In: *TACAS*. Vol. 1384. Lecture Notes in Computer Science. Springer, 1998, pp. 409–423 (cit. on p. 136).

[Kes+01] Yonit Kesten et al. “Symbolic model checking with rich assertional languages”. In: *Theor. Comput. Sci.* 256.1-2 (2001), pp. 93–112 (cit. on pp. 1, 15).

[KM95] Robert P. Kurshan and Kenneth L. McMillan. “A Structural Induction Theorem for Processes”. In: *Inf. Comput.* 117.1 (1995), pp. 1–11 (cit. on p. 85).

[Kra23] Valentin Krasotin. “An invariant-based approach to Regular Model Checking”. MA thesis. Technical University of Munich, 2023 (cit. on pp. 11, 59, 130).

[Leg12] Axel Legay. “Extrapolating (omega-)regular model checking”. In: *Int. J. Softw. Tools Technol. Transf.* 14.2 (2012), pp. 119–143 (cit. on p. 1).

[Len+17] Ondrej Lengál et al. “Fair Termination for Parameterized Probabilistic Concurrent Systems”. In: *TACAS (1).* Vol. 10205. Lecture Notes in Computer Science. 2017, pp. 499–517 (cit. on p. 1).

[Lin+16] Anthony W. Lin et al. “Regular Symmetry Patterns”. In: *VMCAI.* Vol. 9583. Lecture Notes in Computer Science. Springer, 2016, pp. 455–475 (cit. on p. 85).

[LR16] Anthony W. Lin and Philipp Rümmer. “Liveness of Randomised Parameterised Systems under Arbitrary Schedulers”. In: *CAV (2).* Vol. 9780. Lecture Notes in Computer Science. Springer, 2016, pp. 112–133 (cit. on p. 1).

[LR21] Anthony W. Lin and Philipp Rümmer. “Regular Model Checking Revisited”. In: *Model Checking, Synthesis, and Learning.* Vol. 13030. Lecture Notes in Computer Science. Springer, 2021, pp. 97–114 (cit. on pp. 1, 4–6).

[LR81] Daniel Lehmann and Michael O. Rabin. “On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem”. In: *POPL.* ACM Press, 1981, pp. 133–138 (cit. on p. 136).

[Nei14] Daniel Neider. “Applications of automata learning in verification and synthesis”. PhD thesis. RWTH Aachen University, 2014 (cit. on pp. 4, 117, 145).

[NJ13] Daniel Neider and Nils Jansen. “Regular Model Checking Using Solver Technologies and Automata Learning”. In: *NASA Formal Methods.* Vol. 7871. Lecture Notes in Computer Science. Springer, 2013, pp. 16–31 (cit. on pp. 4, 117).

[Pap94] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994 (cit. on p. 48).

[Ras22] Mikhail Raskin. personal communication. 2022 (cit. on pp. 11, 130).

[Var+04] Abhay Vardhan et al. "Learning to Verify Safety Properties". In: *ICFEM*. Vol. 3308. Lecture Notes in Computer Science. Springer, 2004, pp. 274–289 (cit. on pp. 4, 117).

[Var06] Abhay Vardhan. "Learning To Verify Systems". PhD thesis. University of Illinois, 2006 (cit. on pp. 4, 117, 145).

[WB98] Pierre Wolper and Bernard Boigelot. "Verifying Systems with Infinite but Regular State Spaces". In: *CAV*. Vol. 1427. Lecture Notes in Computer Science. Springer, 1998, pp. 88–97 (cit. on pp. 1, 15).

[Wel23a] Christoph Welzel-Mohr. *dodo*. Sept. 2023. DOI: `10.5281/zenodo.8354894`. URL: `https://doi.org/10.5281/zenodo.8354894` (cit. on p. 135).

[Wel23b] Christoph Welzel-Mohr. *`dodo`*. `https://gitlab.lrz.de/i7/dodo`. 2023 (cit. on p. 135).

[WL89] Pierre Wolper and Vinciane Lovinfosse. "Verifying Properties of Large Sets of Processes with Network Invariants". In: *Automatic Verification Methods for Finite State Systems*. Vol. 407. Lecture Notes in Computer Science. Springer, 1989, pp. 68–80 (cit. on p. 85).

A Experimental results for `oneshot`

All tables in this appendix consist of nine columns which contain the following information:

Name: This is the name of the example.

$|I|$: This is the number of states for the automaton that describes the initial language of the system.

$|T|$: This is the number of states for the automaton that describes the language of the transitions of the system.

$|\Sigma|$: This is the number of elements in the alphabet of the system.

Property: This is a description of the property that all undesired configurations have.

$|B|$: This is the number of states for the automaton that describes the language of undesired configurations of the system.

Interpretations: Here we report which interpretations are used. $\mathcal{V}_{trap}{}^*$ and $\mathcal{V}_{siphon}{}^*$ are used if the cashing mechanism for the steps of the transducer for $\Rightarrow_{\mathcal{V}_{trap}}$ and $\Rightarrow_{\mathcal{V}_{siphon}}$ is activated.

Result: This column indicates with either ✓ or × whether the property could be established or not. Alternatively, we indicate here with `oom` and `oot` that `dodo` ran out of memory or time while computing the result. The timeout is set to 20 minutes.

Time: Here we report the time it took to establish the result.

 # expl. abs.: This column reports how many states of the abstraction were explored to establish the result. Note that this coincides with the number of reachable states of the complete automaton of the abstraction if the result is positive.

We do not report on executions with more than one interpretation if any of the interpretations already suffice to establish the property, or `dodo` ran out of space or time for any of the interpretations before.

Contents

A.1 Dijkstra's algorithm for mutual exclusion

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
Dijkstra	2	17	24	Two agents are in the mutual exclusive region simultaneously	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{flow}$	`oom`	×	×
				No transition can be executed	142	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}$	`oot`	2 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{flow}$	`oom`	×	×

A.2 Dijkstra's algorithm for mutual exclusion with a token

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
Dijkstra ring	2	12	12	Two agents are in the mutual exclusive region simultaneously	3	$\mathcal{V}_{trap}$	`oot`	20 (*min*)	×
						${\mathcal{V}_{trap}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{flow}$	`oot`	20 (min)	×
				No transition can be executed	24	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{flow}$	`oot`	20 (min)	×

A.3 Other mutual exclusion algorithms

Name	$\|I\|$	$\|T\|$	$\|\Sigma\|$	Property	$\|B\|$	Interpretations	Result	Time	# expl. abs.
Burns	1	6	6	Two agents are in the mutual exclusive region simultaneously	3	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{trap}}{}^*$	✓	6.2 (s)	316
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{siphon}}{}^*$	×	2.9 (s)	105
						$\mathcal{V}_{\mathit{flow}}$	×	726 (ms)	4
						$\mathcal{V}_{\mathit{siphon}}{}^*, \mathcal{V}_{\mathit{flow}}$	×	2.3 (s)	123
				No transition can be executed	6	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{trap}}{}^*$	✓	6.1 (s)	542
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{siphon}}{}^*$	✓	4.5 (s)	638
						$\mathcal{V}_{\mathit{flow}}$	✓	728 (ms)	13
Szymanski	1	13	50	Two agents are in the mutual exclusive region simultaneously	3	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{trap}}{}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{siphon}}{}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{flow}}$	`oom`	×	×
				No transition can be executed	281	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{trap}}{}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{siphon}}{}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{flow}}$	`oom`	×	×
bakery	2	4	3	Two agents are in the mutual exclusive region simultaneously	3	$\mathcal{V}_{\mathit{trap}}$	✓	145 (ms)	66
						$\mathcal{V}_{\mathit{trap}}{}^*$	✓	109 (ms)	66
						$\mathcal{V}_{\mathit{siphon}}$	×	84 (ms)	15
						$\mathcal{V}_{\mathit{siphon}}{}^*$	×	87 (ms)	15
						$\mathcal{V}_{\mathit{flow}}$	×	188 (ms)	3
						$\mathcal{V}_{\mathit{siphon}}, \mathcal{V}_{\mathit{flow}}$	×	222 (ms)	29
						$\mathcal{V}_{\mathit{siphon}}{}^*, \mathcal{V}_{\mathit{flow}}$	×	214 (ms)	29

A.4 Dining philosophers

Name	$\|I\|$	$\|T\|$	$\|\Sigma\|$	Property	$\|B\|$	Interpretations	Result	Time	# expl. abs.
Atomic	1	8	4	No transition can be executed	17	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{trap}}{}^*$	✓	137 (s)	47293
						$\mathcal{V}_{\mathit{siphon}}$	×	128 (ms)	4
						$\mathcal{V}_{\mathit{siphon}}{}^*$	×	115 (ms)	4
						$\mathcal{V}_{\mathit{flow}}$	×	415 (ms)	4
						$\mathcal{V}_{\mathit{siphon}}, \mathcal{V}_{\mathit{flow}}$	×	443 (s)	107
						$\mathcal{V}_{\mathit{siphon}}{}^*, \mathcal{V}_{\mathit{flow}}$	×	4.4 (s)	107
Lefty	1	11	6	No transition can be executed	20	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{trap}}{}^*$	×	15 (s)	73
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{siphon}}{}^*$	×	3.0 (s)	4
						$\mathcal{V}_{\mathit{flow}}$	×	7.0 (s)	4
						$\mathcal{V}_{\mathit{trap}}{}^*, \mathcal{V}_{\mathit{siphon}}{}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{trap}}, \mathcal{V}_{\mathit{flow}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{trap}}{}^*, \mathcal{V}_{\mathit{flow}}$	`oom`	×	×
						$\mathcal{V}_{\mathit{siphon}}{}^*, \mathcal{V}_{\mathit{flow}}$	×	8.9 (s)	5
Return	1	7	6	No transition can be executed	20	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{trap}}{}^*$	`oom`	×	×
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						$\mathcal{V}_{\mathit{siphon}}{}^*$	×	31 (s)	12
						$\mathcal{V}_{\mathit{flow}}$	×	2.8 (s)	3
						$\mathcal{V}_{\mathit{siphon}}{}^*, \mathcal{V}_{\mathit{flow}}$	×	70 (s)	35

A.5 Cache coherence protocols

MESI

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
MESI	1	7	4	Two cells are modified at the same time	3	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{trap}}}^*$	✓	16 (s)	1743
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{siphon}}}^*$	×	2.4 (s)	180
						$\mathcal{V}_{\mathit{flow}}$	×	284 (ms)	4
						${\mathcal{V}_{\mathit{siphon}}}^*, \mathcal{V}_{\mathit{flow}}$	×	495 (ms)	22
				One cell falsely claims ownership	4	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{trap}}}^*$	✓	16 (s)	2460
						$\mathcal{V}_{\mathit{siphon}}$	×	109 (s)	10
						${\mathcal{V}_{\mathit{siphon}}}^*$	×	316 (ms)	10
						$\mathcal{V}_{\mathit{flow}}$	×	286 (ms)	2
						$\mathcal{V}_{\mathit{siphon}}, \mathcal{V}_{\mathit{flow}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{siphon}}}^*, \mathcal{V}_{\mathit{flow}}$	×	533 (ms)	5
				No transition can be executed	6	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{trap}}}^*$	✓	17 (s)	2963
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{siphon}}}^*$	✓	17 (s)	2963
						$\mathcal{V}_{\mathit{flow}}$	✓	470 (ms)	11

Illinois

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
Illinois	1	16	4	Two cells are dirty at the same time	3	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{trap}}}^*$	`oom`	×	×
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{siphon}}}^*$	`oom`	×	×
						$\mathcal{V}_{\mathit{flow}}$	×	4.6 (s)	16
				One cell is dirty and another is shared	4	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{trap}}}^*$	`oom`	×	×
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{siphon}}}^*$	`oom`	×	×
						$\mathcal{V}_{\mathit{flow}}$	×	4.5 (s)	4
				No transition can be executed	12	$\mathcal{V}_{\mathit{trap}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{trap}}}^*$	`oom`	×	×
						$\mathcal{V}_{\mathit{siphon}}$	`oot`	20 (min)	×
						${\mathcal{V}_{\mathit{siphon}}}^*$	`oom`	×	×
						$\mathcal{V}_{\mathit{flow}}$	✓	4.6 (s)	4

MOESI

Name	$\lvert I \rvert$	$\lvert T \rvert$	$\lvert \Sigma \rvert$	Property	$\lvert B \rvert$	Interpretations	Result	Time	# expl. abs.
MOESI	1	7	5	Two cells are modified at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}^*$	✓	73 (s)	2055
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}^*$	×	9.8 (s)	163
						$\mathcal{V}_{flow}$	×	708 (ms)	4
						$\mathcal{V}_{siphon}^*, \mathcal{V}_{flow}$	×	2.2 (s)	67
				Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}^*$	✓	71 (s)	2446
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}^*$	×	1.1 (s)	17
						$\mathcal{V}_{flow}$	×	668 (ms)	4
						$\mathcal{V}_{siphon}^*, \mathcal{V}_{flow}$	×	975 (ms)	3
				One cell falsely claims exclusive access (other cell shared)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}^*$	✓	70 (s)	3252
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}^*$	×	1.1 (s)	17
						$\mathcal{V}_{flow}$	×	683 (ms)	4
						$\mathcal{V}_{siphon}^*, \mathcal{V}_{flow}$	×	1 (s)	3
				One cell falsely claims exclusive access (other cell claims ownership)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}^*$	✓	71 (s)	2648
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}^*$	×	1.1 (s)	19
						$\mathcal{V}_{flow}$	×	685 (ms)	4
						$\mathcal{V}_{siphon}^*, \mathcal{V}_{flow}$	×	1.6 (s)	27
				One cell falsely claims exclusive access (other cell modified)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}^*$	✓	70 (s)	2543
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}^*$	×	1.1 (s)	19
						$\mathcal{V}_{flow}$	×	682 (ms)	4
						$\mathcal{V}_{siphon}^*, \mathcal{V}_{flow}$	×	2.1 (s)	35
				One cell falsely claims ownership (other cell modified)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}^*$	✓	71 (s)	2257
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}^*$	×	8 (s)	138
						$\mathcal{V}_{flow}$	×	706 (ms)	4
						$\mathcal{V}_{siphon}^*, \mathcal{V}_{flow}$	×	1.4 (s)	13
				One cell falsely claims modified content (other cell shared)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}^*$	✓	71 (s)	2861
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}^*$	×	6.7 (s)	113
						$\mathcal{V}_{flow}$	×	695 (ms)	4
						$\mathcal{V}_{siphon}^*, \mathcal{V}_{flow}$	×	2 (s)	20
				No transition can be executed	5	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}^*$	✓	72 (s)	3633
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}^*$	✓	70 (s)	5009
						$\mathcal{V}_{flow}$	✓	702 (ms)	10

Berkeley

Name	$\|I\|$	$\|T\|$	$\|\Sigma\|$	Property	$\|B\|$	Interpretations	Result	Time	# expl. abs.
Berkeley	1	9	4	Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	×	11 (s)	177
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	×	3.7 (s)	10
						$\mathcal{V}_{flow}$	×	768 (ms)	3
						${\mathcal{V}_{trap}}^*, \mathcal{V}_{flow}$	×	5.2 (s)	57
						${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	×	5.7 (s)	9
						${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*$	`oom`	×	×
				One cell falsely claims exclusive access (other cell claims shared ownership)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	×	4 (s)	7
						$\mathcal{V}_{flow}$	×	685 (ms)	4
						${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	×	4.7 (s)	6
				One cell falsely claims exclusive access (other cell claims unexclusive access)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	×	4 (s)	7
						$\mathcal{V}_{flow}$	×	705 (ms)	4
						${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	×	5 (s)	3
				No transition can be executed	10	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	✓	660 (ms)	17

Synapse

Name	$\|I\|$	$\|T\|$	$\|\Sigma\|$	Property	$\|B\|$	Interpretations	Result	Time	# expl. abs.
Synapse	1	5	3	Two cells are dirty at the same time	3	$\mathcal{V}_{trap}$	✓	12 (s)	96
						${\mathcal{V}_{trap}}^*$	✓	219 (ms)	96
						$\mathcal{V}_{siphon}$	×	3.1 (s)	18
						${\mathcal{V}_{siphon}}^*$	×	156 (ms)	18
						$\mathcal{V}_{flow}$	×	132 (ms)	4
						$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	×	3.4 (s)	8
						${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	×	171 (ms)	8
				One cell falsely claims exclusive access (other cell claims unexclusive access)	4	$\mathcal{V}_{trap}$	✓	15 (s)	126
						${\mathcal{V}_{trap}}^*$	✓	219 (ms)	126
						$\mathcal{V}_{siphon}$	×	4 (s)	21
						${\mathcal{V}_{siphon}}^*$	×	165 (ms)	21
						$\mathcal{V}_{flow}$	×	130 (ms)	2
						$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	×	347 (ms)	1
						${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	×	163 (ms)	1
				No transition can be executed	4	$\mathcal{V}_{trap}$	✓	11 (s)	106
						${\mathcal{V}_{trap}}^*$	✓	202 (ms)	106
						$\mathcal{V}_{siphon}$	✓	11 (s)	97
						${\mathcal{V}_{siphon}}^*$	✓	202 (ms)	97
						$\mathcal{V}_{flow}$	✓	16 (ms)	8

FutureBus+

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
FutureBus+	1	21	9	Two cells have pending (right) changes at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	`oot`	20 (min)	×
				One cell falsely claims exclusive access (other cell claims shared ownership)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	`oot`	20 (min)	×
				Two cells have pending changes at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	`oot`	20 (min)	×
				Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	`oot`	20 (min)	×
				No transition can be executed	144	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	`oot`	20 (min)	×

Firefly

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
Firefly	1	16	4	Two cells are dirty at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	1.7 (s)	4
				Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	1.7 (s)	4
				One cell falsely claims exclusive access (other cell is shared)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	1.6 (s)	4
				One cell falsely claims exclusive access (other cell claims dirty access)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	1.7 (s)	4
				No transition can be executed	16	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						$\mathcal{V}_{trap}{}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						$\mathcal{V}_{siphon}{}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	✓	1.7 (s)	23

Dragon

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
Dragon	1	23	3	Two cells are dirty at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	102 (s)	4
				Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	103 (s)	5
				One cell falsely claims exclusive access (other cell claims dirty and shared access)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	103 (s)	3
				One cell falsely claims exclusive access (other cell claims shared access)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	111 (s)	5
				One cell falsely claims exclusive access (other cell claims dirty access)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	102 (s)	5
				One cell falsely claims dirty access (other cell claims shared access)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	104 (s)	4
				One cell falsely claims dirty access (other cell claims dirty and shared access)	4	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	×	109 (s)	10
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	105 (s)	3
				No transition can be executed	16	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	✓	103 (s)	25

A.6 Termination detection

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
Termination detection	1	6	4	Two tokens moving down	3	$\mathcal{V}_{trap}$	✓	1 (s)	244
						${\mathcal{V}_{trap}}^*$	✓	234 (ms)	244
						$\mathcal{V}_{siphon}$	×	272 (ms)	77
						${\mathcal{V}_{siphon}}^*$	×	159 (ms)	77
						$\mathcal{V}_{flow}$	✓	389 (ms)	498
				Two tokens moving up	3	$\mathcal{V}_{trap}$	✓	1.4 (s)	331
						${\mathcal{V}_{trap}}^*$	✓	241 (ms)	331
						$\mathcal{V}_{siphon}$	×	372 (ms)	83
						${\mathcal{V}_{siphon}}^*$	×	162 (ms)	83
						$\mathcal{V}_{flow}$	✓	377 (ms)	393
				No transition can be executed	7	$\mathcal{V}_{trap}$	✓	1 (s)	282
						${\mathcal{V}_{trap}}^*$	✓	239 (ms)	282
						$\mathcal{V}_{siphon}$	×	142 (ms)	36
						${\mathcal{V}_{siphon}}^*$	×	205 (ms)	36
						$\mathcal{V}_{flow}$	✓	350 (ms)	327

A.7 Dining cryptographers

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
Dining cryptographers	2	8	12	Paying cryptographer	4	$\mathcal{V}_{trap}$	✓	295 (s)	1226
						${\mathcal{V}_{trap}}^*$	✓	75 (s)	1226
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{flow}$	`oot`	20 (min)	×
				No paying cryptographer	2	$\mathcal{V}_{trap}$	✓	102 (s)	614
						${\mathcal{V}_{trap}}^*$	✓	36 (s)	614
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oot`	20 (min)	×
						$\mathcal{V}_{flow}$	`oot`	20 (min)	×

A.8 Leader election

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
Herman	2	11	2	Only followers	1	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	`oot`	20 (min)	×
						${\mathcal{V}_{siphon}}^*$	`oom`	×	×
						$\mathcal{V}_{flow}$	×	145 (ms)	2
Israeli-Jafon	2	10	2	Only followers	1	$\mathcal{V}_{trap}$	`oot`	20 (min)	×
						${\mathcal{V}_{trap}}^*$	`oom`	×	×
						$\mathcal{V}_{siphon}$	×	111 (s)	4
						${\mathcal{V}_{siphon}}^*$	×	1 (s)	4
						$\mathcal{V}_{flow}$	×	130 (ms)	2
						$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	×	5.5 (s)	4
						${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	×	543 (ms)	4

A.9 Token passing

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Property	$\lvert B\rvert$	Interpretations	Result	Time	# expl. abs.
With invariant	2	3	2	There is no token	2	$\mathcal{V}_{trap}$	✓	77 (ms)	15
						$\mathcal{V}_{trap}^{*}$	✓	81 (ms)	15
						$\mathcal{V}_{siphon}$	×	73 (ms)	6
						$\mathcal{V}_{siphon}^{*}$	×	77 (ms)	6
						$\mathcal{V}_{flow}$	✓	130 (ms)	35
				There are many tokens	3	$\mathcal{V}_{trap}$	✓	76 (ms)	15
						$\mathcal{V}_{trap}^{*}$	✓	82 (ms)	15
						$\mathcal{V}_{siphon}$	×	69 (ms)	3
						$\mathcal{V}_{siphon}^{*}$	×	74 (ms)	3
						$\mathcal{V}_{flow}$	✓	133 (ms)	35
Without invariant	2	3	2	There is no token	2	$\mathcal{V}_{trap}$	✓	85 (ms)	25
						$\mathcal{V}_{trap}^{*}$	✓	88 (ms)	25
						$\mathcal{V}_{siphon}$	×	75 (ms)	8
						$\mathcal{V}_{siphon}^{*}$	×	78 (ms)	8
						$\mathcal{V}_{flow}$	✓	130 (ms)	23
				There are many tokens	3	$\mathcal{V}_{trap}$	×	75 (ms)	10
						$\mathcal{V}_{trap}^{*}$	×	79 (ms)	10
						$\mathcal{V}_{siphon}$	×	71 (ms)	3
						$\mathcal{V}_{siphon}^{*}$	×	75 (ms)	3
						$\mathcal{V}_{flow}$	✓	127 (ms)	23
						$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	×	93 (ms)	28
						$\mathcal{V}_{trap}^{*}, \mathcal{V}_{siphon}^{*}$	×	94 (ms)	28

B Experimental results for `learn` and `adaptive`

All tables in this appendix consist of fourteen columns which contain the following information:

Name: This is the name of the example.

$|I|$: This is the number of states for the automaton that describes the initial language of the system.

$|T|$: This is the number of states for the automaton that describes the language of the transitions of the system.

$|\Sigma|$: This is the number of elements in the alphabet of the system.

Topology: This is the name of the topology of this example (or × if this example does not follow any of the considered topologies).

Property: This is a description of the property that all undesired configurations have.

$|B|$: This is the number of states for the automaton that describes the language of undesired configurations of the system.

Interpretations: Here we report which interpretations are used. ${\mathcal{V}_{trap}}^*$ and ${\mathcal{V}_{siphon}}^*$ are used if instances of Problem 3.1 for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ are solved with an embedding into a propositional formula.

Mode: This is either `learn` or `adaptive` depending on which mode is used.

Result: This column indicates with either ✓ or × whether the property could be established or not. Alternatively, we indicate here with `oom` and `oot` that `dodo` ran out of memory or time while computing the result. The timeout is set to 20 minutes.

Time: Here we report the time it took to establish the result.

 # expl. abs.: This column reports two values x and y in the form (x/y). Here x is the number of states of the automata that recognize all inductive statements that are learned. y, on the other hand, is the number of states of the transducer which captures the abstraction of the learned inductive statements until the result is established.

 # 2^Σ: In this column the number of letters for the learned inductive statements of all interpretations is given.

 # instances: For the mode `learn`, this column reports how often Problem 3.1 is solved until the result is established. For the mode `adaptive`, this column reports, again, two values x and y in the form (x/y). Here, x is how often Problem 3.1 is solved until the result is established and y reports how many of the solutions for Problem 3.1 could be generalized with the help of Lemma 3.5.

We do not report on executions with more than one interpretation if any of the interpretations already suffice to establish the property, or `dodo` ran out of space or time for any of the interpretations before.

Contents

B.1 Dijkstra's algorithm for mutual exclusion

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
Dijkstra	2	17	24	`crowd`	Two agents are in the mutually exclusive region simultaneously	3	$\mathcal{V}_{trap}$	`learn`	✓	385 (s)	(12/157)	12	8
								`adaptive`	✓	123 (s)	(128/988)	18	(26/26)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	750 (s)	(11/278)	12	7
								`adaptive`	✓	27 (s)	(32/507)	12	(7/7)
							$\mathcal{V}_{siphon}$	`learn`	×	844 (ms)	(1/16)	1	1
								`adaptive`	×	826 (ms)	(3/16)	1	(2/2)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	665 (ms)	(1/13)	2	1
								`adaptive`	×	841 (ms)	(5/17)	2	(2/1)
							$\mathcal{V}_{flow}$	`learn`	×	20 (s)	(5/100)	3	3
								`adaptive`	×	6 (s)	(21/49)	3	(3/2)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	1.9 (s)	(2/82)	2	5
								`adaptive`	×	1.8 (s)	(8/63)	2	(5/2)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	1.5 (s)	(2/61)	3	5
								`adaptive`	×	1.8 (s)	(10/49)	3	(5/2)
					No transition can be executed	142	$\mathcal{V}_{trap}$	`learn`	✓	5 (s)	(4/42)	6	6
								`adaptive`	✓	45 (s)	(101/111)	6	(20/20)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	58 (s)	(4/58)	6	6
								`adaptive`	✓	98 (s)	(50/1510)	11	(13/13)
							$\mathcal{V}_{siphon}$	`learn`	×	2.6 (s)	(4/37)	2	3
								`adaptive`	×	1.3 (s)	(5/39)	2	(3/2)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	4.4 (s)	(4/39)	4	4
								`adaptive`	×	1.2 (s)	(5/46)	2	(3/2)
							$\mathcal{V}_{flow}$	`learn`	✓	20 (s)	(7/471)	7	6
								`adaptive`	✓	97 (s)	(50/524)	8	(9/7)

B.2 Dijkstra's algorithm for mutual exclusion with a token

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
Dijkstra ring	2	12	12	`ring`	Two agents are in the mutually exclusive region simultaneously	3	$\mathcal{V}_{trap}$	`learn`	×	327 (s)	(14/750)	8	7
								`adaptive`	×	1.5 (s)	(58/394)	8	(10/9)
							${\mathcal{V}_{trap}}^*$	`learn`	×	70 (s)	(13/1230)	8	6
								`adaptive`	×	2.4 (s)	(38/4742)	8	(6/5)
							$\mathcal{V}_{siphon}$	`learn`	×	95 (ms)	(1/7)	0	1
								`adaptive`	×	92 (ms)	(1/7)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	106 (ms)	(1/7)	0	1
								`adaptive`	×	102 (ms)	(1/7)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	✓	2.4 (s)	(9/235)	5	3
								`adaptive`	✓	6.9 (s)	(26/7345)	7	(4/3)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	329 (s)	(15/750)	8	8
								`adaptive`	×	879 (ms)	(59/394)	8	(11/9)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*$	`learn`	×	100 (s)	(14/1230)	8	7
								`adaptive`	×	3.3 (s)	(39/4742)	8	(7/5)
					No transition can be executed	24	$\mathcal{V}_{trap}$	`learn`	✓	1.4 (s)	(7/4055)	5	4
								`adaptive`	✓	700 (ms)	(15/139)	5	(5/2)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	2.8 (s)	(9/81)	7	5
								`adaptive`	✓	561 (ms)	(20/167)	7	(6/3)
							$\mathcal{V}_{siphon}$	`learn`	×	125 (ms)	(1/18)	1	2
								`adaptive`	×	123 (ms)	(1/18)	1	(2/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	141 (ms)	(1/18)	1	2
								`adaptive`	×	137 (ms)	(1/18)	1	(2/0)
							$\mathcal{V}_{flow}$	`learn`	✓	2.1 (s)	(8/95)	5	4
								`adaptive`	✓	910 (ms)	(18/357)	5	(5/3)

B.3 Other mutual exclusion algorithms

Name	$\|I\|$	$\|T\|$	$\|\Sigma\|$	Topology	Property	$\|B\|$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
Burns'	1	6	6	×	Two agents are in the mutually exclusive region simultaneously	3	$\mathcal{V}_{trap}$	`learn`	✓	552 (ms)	(6/74)	7	6
							$\mathcal{V}_{trap}{}^{*}$	`learn`	✓	335 (ms)	(5/38)	4	3
							$\mathcal{V}_{siphon}$	`learn`	×	77 (ms)	(1/6)	0	1
							$\mathcal{V}_{siphon}{}^{*}$	`learn`	×	86 (ms)	(1/6)	0	1
							$\mathcal{V}_{flow}$	`learn`	×	101 (ms)	(1/6)	0	1
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	111 (ms)	(2/6)	0	2
							$\mathcal{V}_{siphon}{}^{*}, \mathcal{V}_{flow}$	`learn`	×	106 (ms)	(2/6)	0	2
					No transition can be executed	5	$\mathcal{V}_{trap}$	`learn`	✓	67 (ms)	(1/2)	0	0
							$\mathcal{V}_{trap}{}^{*}$	`learn`	✓	68 (ms)	(1/2)	0	0
							$\mathcal{V}_{siphon}$	`learn`	✓	68 (ms)	(1/2)	0	0
							$\mathcal{V}_{siphon}{}^{*}$	`learn`	✓	69 (ms)	(1/0)	0	0
							$\mathcal{V}_{flow}$	`learn`	✓	69 (ms)	(1/2)	0	0
Szymanski	1	13	50	×	Two agents are in the mutually exclusive region simultaneously	3	$\mathcal{V}_{trap}$	`learn`	×	115 (s)	(7/95)	6	6
							$\mathcal{V}_{trap}{}^{*}$	`learn`	×	81 (s)	(3/30)	2	2
							$\mathcal{V}_{siphon}$	`learn`	×	10 (s)	(1/15)	1	2
							$\mathcal{V}_{siphon}{}^{*}$	`learn`	×	28 (s)	(1/33)	5	5
							$\mathcal{V}_{flow}$	`learn`	×	93 (s)	(2/158)	7	6
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	112 (s)	(8/95)	6	7
							$\mathcal{V}_{trap}{}^{*}, \mathcal{V}_{siphon}{}^{*}$	`learn`	×	79 (s)	(4/30)	2	3
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	114 (s)	(8/95)	6	7
							$\mathcal{V}_{trap}{}^{*}, \mathcal{V}_{flow}$	`learn`	×	81 (s)	(4/30)	2	3
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	12 (s)	(2/15)	1	3
							$\mathcal{V}_{siphon}{}^{*}, \mathcal{V}_{flow}$	`learn`	×	30 (s)	(2/33)	5	6
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	1150 (s)	(9/95)	6	8
							$\mathcal{V}_{trap}{}^{*}, \mathcal{V}_{siphon}{}^{*}, \mathcal{V}_{flow}$	`learn`	×	80 (s)	(5/30)	2	4
					No transition can be executed	281	$\mathcal{V}_{trap}$	`learn`	`oot`	20 (min)	(×/×)	×	×
							$\mathcal{V}_{trap}{}^{*}$	`learn`	×	441 (s)	(4/317)	6	5
							$\mathcal{V}_{siphon}$	`learn`	×	63 (s)	(4/127)	2	3
							$\mathcal{V}_{siphon}{}^{*}$	`learn`	×	204 (s)	(4/159)	7	7
							$\mathcal{V}_{flow}$	`learn`	`oot`	20 (min)	(×/×)	×	×
							$\mathcal{V}_{trap}{}^{*}, \mathcal{V}_{siphon}{}^{*}$	`learn`	×	443 (s)	(5/317)	6	6
bakery	2	4	3	×	Two agents are in the mutually exclusive region simultaneously	3	$\mathcal{V}_{trap}$	`learn`	✓	157 (ms)	(6/36)	3	2
							$\mathcal{V}_{trap}{}^{*}$	`learn`	✓	170 (ms)	(6/36)	3	2
							$\mathcal{V}_{siphon}$	`learn`	×	68 (ms)	(1/7)	0	1
							$\mathcal{V}_{siphon}{}^{*}$	`learn`	×	77 (ms)	(1/7)	0	1
							$\mathcal{V}_{flow}$	`learn`	×	85 (ms)	(1/7)	0	1
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	94 (ms)	(2/7)	0	2
							$\mathcal{V}_{siphon}{}^{*}, \mathcal{V}_{flow}$	`learn`	×	92 (ms)	(2/7)	0	2

B.4 Dining philosophers

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
Atomic	1	8	4	`ring`	No transition can be executed	17	$\mathcal{V}_{trap}$	`learn`	✓	166 (ms)	(10/59)	4	5
								`adaptive`	✓	92 (s)	(9004/37397)	4	(457/457)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	784 (ms)	(13/156)	6	6
								`adaptive`	`oom`	×	(×/×)	×	(×/×)
							$\mathcal{V}_{siphon}$	`learn`	×	74 (ms)	(1/19)	0	1
								`adaptive`	×	71 (ms)	(1/19)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	84 (ms)	(1/19)	0	1
								`adaptive`	×	83 (ms)	(1/19)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	97 (ms)	(1/19)	0	1
								`adaptive`	×	95 (ms)	(1/19)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	107 (ms)	(2/19)	0	2
								`adaptive`	×	103 (ms)	(2/19)	2	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/19)	0	2
								`adaptive`	×	101 (ms)	(2/19)	0	(2/0)
Lefty	1	11	6	`bow`	No transition can be executed	20	$\mathcal{V}_{trap}$	`learn`	×	5.5 (s)	(20/244)	9	8
								`adaptive`	×	143 (ms)	(29/64)	9	(10/9)
							$\mathcal{V}_{trap}{}^*$	`learn`	×	2.7 (s)	(13/185)	9	6
								`adaptive`	×	149 (ms)	(17/69)	9	(6/5)
							$\mathcal{V}_{siphon}$	`learn`	×	81 (ms)	(1/19)	0	1
								`adaptive`	×	79 (ms)	(1/19)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	92 (ms)	(1/19)	0	1
								`adaptive`	×	90 (ms)	(1/19)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	106 (ms)	(1/19)	0	1
								`adaptive`	×	104 (ms)	(1/19)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	120 (ms)	(2/19)	0	2
								`adaptive`	×	116 (ms)	(2/19)	0	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	115 (ms)	(2/19)	0	2
								`adaptive`	×	113 (ms)	(2/19)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	✓	148 (s)	(43/9461)	23	18
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							$\mathcal{V}_{trap}{}^*, \mathcal{V}_{flow}$	`learn`	✓	94 (s)	(34/7352)	18	13
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	5.6 (s)	(21/244)	9	9
								`adaptive`	×	151 (ms)	(30/64)	9	(11/9)
							$\mathcal{V}_{trap}{}^*, \mathcal{V}_{siphon}{}^*$	`learn`	×	2.8 (s)	(14/185)	9	7
								`adaptive`	×	154 (ms)	(18/69)	9	(7/5)
Return	1	7	6	`ring`	No transition can be executed	20	$\mathcal{V}_{trap}$	`learn`	✓	193 (ms)	(5/45)	4	6
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	721 (ms)	(5/43)	3	4
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							$\mathcal{V}_{siphon}$	`learn`	×	79 (ms)	(1/15)	0	1
								`adaptive`	×	78 (ms)	(1/15)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	90 (ms)	(1/15)	0	1
								`adaptive`	×	88 (ms)	(1/15)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	103 (ms)	(1/15)	0	1
								`adaptive`	×	99 (ms)	(1/15)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	116 (ms)	(2/15)	0	2
								`adaptive`	×	114 (ms)	(2/15)	0	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	111 (ms)	(2/15)	0	2
								`adaptive`	×	110 (ms)	(2/15)	0	(2/0)

B.5 Cache coherence protocols

MESI

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^{Σ}	# instances
MESI	1	7	4	`crowd`	Two cells are modified at the same time	3	$\mathcal{V}_{trap}$	`learn`	✓	140 (ms)	(4/53)	3	3
								`adaptive`	✓	102 (ms)	(12/39)	3	(3/3)
							$\mathcal{V}_{trap}{}^{*}$	`learn`	✓	121 (ms)	(2/43)	3	2
								`adaptive`	✓	148 (ms)	(14/45)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	68 (ms)	(1/6)	0	1
								`adaptive`	×	68 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}{}^{*}$	`learn`	×	80 (ms)	(1/6)	0	1
								`adaptive`	×	77 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	95 (ms)	(1/6)	0	1
								`adaptive`	×	91 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/6)	0	2
								`adaptive`	×	101 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{siphon}{}^{*}, \mathcal{V}_{flow}$	`learn`	×	100 (ms)	(2/6)	0	2
								`adaptive`	×	97 (ms)	(2/6)	0	(2/0)
					One cell falsely claims ownership	4	$\mathcal{V}_{trap}$	`learn`	✓	149 (ms)	(4/53)	4	3
								`adaptive`	✓	134 (ms)	(14/46)	3	(2/2)
							$\mathcal{V}_{trap}{}^{*}$	`learn`	✓	149 (ms)	(4/53)	4	3
								`adaptive`	✓	151 (ms)	(14/46)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	71 (ms)	(1/8)	0	1
								`adaptive`	×	68 (ms)	(1/1)	0	(1/0)
							$\mathcal{V}_{siphon}{}^{*}$	`learn`	×	80 (ms)	(1/8)	0	1
								`adaptive`	×	79 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	94 (ms)	(1/8)	0	1
								`adaptive`	×	92 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/8)	0	2
								`adaptive`	×	101 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}{}^{*}, \mathcal{V}_{flow}$	`learn`	×	101 (ms)	(2/8)	0	2
								`adaptive`	×	99 (ms)	(2/8)	0	(2/0)
					No transition can be executed	6	$\mathcal{V}_{trap}$	`learn`	✓	622 (ms)	(1/2)	0	0
								`adaptive`	✓	59 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{trap}{}^{*}$	`learn`	✓	62 (ms)	(1/2)	0	0
								`adaptive`	✓	61 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}$	`learn`	✓	62 (ms)	(1/2)	0	0
								`adaptive`	✓	61 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}{}^{*}$	`learn`	✓	63 (ms)	(1/2)	0	0
								`adaptive`	✓	62 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{flow}$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	62 (ms)	(1/2)	0	(0/0)

Illinois

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
Illinois	1	16	4	`crowd`	Two cells are dirty at the same time	3	$\mathcal{V}_{trap}$	`learn`	×	70 (ms)	(1/6)	0	1
								`adaptive`	×	66 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{trap}}^*$	`learn`	×	83 (ms)	(1/6)	0	1
								`adaptive`	×	79 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}$	`learn`	×	70 (ms)	(1/6)	0	1
								`adaptive`	×	67 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	82 (ms)	(1/6)	0	1
								`adaptive`	×	80 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	101 (ms)	(1/6)	0	1
								`adaptive`	×	99 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	109 (ms)	(2/6)	0	2
								`adaptive`	×	108 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{trap}}^*, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/6)	0	2
								`adaptive`	×	103 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	108 (ms)	(2/6)	0	2
								`adaptive`	×	107 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	106 (ms)	(2/6)	0	2
								`adaptive`	×	103 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	76 (ms)	(2/6)	0	2
								`adaptive`	×	74 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*$	`learn`	×	89 (ms)	(2/6)	0	2
								`adaptive`	×	86 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	115 (ms)	(3/6)	0	3
								`adaptive`	×	110 (ms)	(3/6)	0	(3/0)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	110 (ms)	(3/6)	0	3
								`adaptive`	×	106 (ms)	(3/6)	0	(3/0)
					One cell is dirty and another is shared	4	$\mathcal{V}_{trap}$	`learn`	×	98 (ms)	(4/21)	4	2
								`adaptive`	×	90 (ms)	(6/24)	2	(2/1)
							${\mathcal{V}_{trap}}^*$	`learn`	×	116 (ms)	(4/21)	2	2
								`adaptive`	×	108 (ms)	(6/24)	2	(2/1)
							$\mathcal{V}_{siphon}$	`learn`	×	71 (ms)	(1/8)	0	1
								`adaptive`	×	68 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	81 (ms)	(1/8)	0	1
								`adaptive`	×	82 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	101 (ms)	(1/8)	0	1
								`adaptive`	×	99 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	143 (ms)	(5/21)	2	2
								`adaptive`	×	135 (ms)	(7/24)	2	(3/1)
							${\mathcal{V}_{trap}}^*, \mathcal{V}_{flow}$	`learn`	×	142 (ms)	(5/21)	2	2
								`adaptive`	×	133 (ms)	(7/24)	2	(3/1)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	109 (ms)	(2/8)	0	2
								`adaptive`	×	108 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	106 (ms)	(2/8)	0	2
								`adaptive`	×	102 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	103 (ms)	(5/21)	2	3
								`adaptive`	×	97 (ms)	(7/24)	2	(3/1)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*$	`learn`	×	121 (ms)	(5/21)	2	3
								`adaptive`	×	115 (ms)	(7/24)	2	(3/1)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	149 (ms)	(6/21)	2	4
								`adaptive`	×	137 (ms)	(8/24)	2	(4/1)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	146 (ms)	(6/21)	2	4
								`adaptive`	×	135 (ms)	(8/24)	2	(4/1)
					No transition can be executed	12	$\mathcal{V}_{trap}$	`learn`	✓	65 (ms)	(1/2)	0	0
								`adaptive`	✓	63 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	61 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}$	`learn`	✓	65 (ms)	(1/2)	0	0
								`adaptive`	✓	62 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	✓	65 (ms)	(1/2)	0	0
								`adaptive`	✓	62 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{flow}$	`learn`	✓	66 (ms)	(1/2)	0	0
								`adaptive`	✓	65 (ms)	(1/2)	0	(0/0)

MOESI

Name	$\lvert I \rvert$	$\lvert T \rvert$	$\lvert \Sigma \rvert$	Topology	Property	$\lvert B \rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^{Σ}	# instances
MOESI	1	7	5	`crowd`	Two cells are modified at the same time	3	$\mathcal{V}_{trap}$	`learn`	✓	159 (ms)	(4/53)	3	3
								`adaptive`	✓	110 (ms)	(12/39)	3	(3/3)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	170 (ms)	(4/47)	3	2
								`adaptive`	✓	165 (ms)	(14/45)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	72 (ms)	(1/6)	0	1
								`adaptive`	×	68 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	83 (ms)	(1/6)	0	1
								`adaptive`	×	80 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	97 (ms)	(1/6)	0	1
								`adaptive`	×	94 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	106 (ms)	(2/6)	0	2
								`adaptive`	×	102 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	103 (ms)	(2/6)	0	2
								`adaptive`	×	100 (ms)	(2/6)	0	(2/0)
					Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`learn`	✓	155 (ms)	(4/32)	3	3
								`adaptive`	✓	109 (ms)	(12/26)	3	(3/3)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	133 (ms)	(2/43)	2	2
								`adaptive`	✓	164 (ms)	(14/45)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	72 (ms)	(1/6)	0	1
								`adaptive`	×	68 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	81 (ms)	(1/6)	0	1
								`adaptive`	×	178 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	97 (ms)	(1/6)	0	1
								`adaptive`	×	94 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/6)	0	2
								`adaptive`	×	104 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	102 (ms)	(2/6)	0	2
								`adaptive`	×	100 (ms)	(2/6)	0	(2/0)
					One cell falsely claims exclusive access (other cell shared)	4	$\mathcal{V}_{trap}$	`learn`	✓	158 (ms)	(4/31)	3	3
								`adaptive`	✓	153 (ms)	(14/29)	3	(2/2)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	165 (ms)	(4/59)	4	3
								`adaptive`	✓	169 (ms)	(14/71)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	72 (ms)	(1/8)	0	1
								`adaptive`	×	71 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	84 (ms)	(1/8)	0	1
								`adaptive`	×	80 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	98 (ms)	(1/8)	0	1
								`adaptive`	×	96 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	108 (ms)	(2/8)	0	2
								`adaptive`	×	103 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/8)	0	2
								`adaptive`	×	103 (ms)	(2/8)	0	(2/0)
					One cell falsely claims exclusive access (other cell claims ownership)	4	$\mathcal{V}_{trap}$	`learn`	✓	180 (ms)	(6/46)	4	4
								`adaptive`	✓	180 (ms)	(22/44)	4	(3/3)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	170 (ms)	(4/47)	3	2
								`adaptive`	✓	166 (ms)	(14/61)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	71 (ms)	(1/8)	0	1
								`adaptive`	×	68 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	83 (ms)	(1/8)	0	1
								`adaptive`	×	81 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	97 (ms)	(1/8)	0	1
								`adaptive`	×	94 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	109 (ms)	(2/8)	0	2
								`adaptive`	×	105 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	106 (ms)	(2/8)	0	2
								`adaptive`	×	101 (ms)	(2/8)	0	(2/0)
					One cell falsely claims exclusive access (other cell modified)	4	$\mathcal{V}_{trap}$	`learn`	✓	171 (ms)	(4/53)	4	3
								`adaptive`	✓	180 (ms)	(22/52)	4	(3/3)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	132 (ms)	(2/49)	3	2
								`adaptive`	✓	166 (ms)	(14/45)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	70 (ms)	(1/8)	0	1
								`adaptive`	×	68 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	82 (ms)	(1/8)	0	1
								`adaptive`	×	81 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	98 (ms)	(1/8)	0	1
								`adaptive`	×	94 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	106 (ms)	(2/8)	0	2
								`adaptive`	×	104 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	104 (ms)	(2/8)	0	2
								`adaptive`	×	101 (ms)	(2/8)	0	(2/0)

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
MOESI contd.	1	7	5	`crowd`	One cell falsely claims ownership (other cell modified)	4	$\mathcal{V}_{trap}$	`learn`	✓	181 (ms)	(6/74)	4	4
								`adaptive`	✓	178 (ms)	(22/69)	4	(3/3)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	169 (ms)	(4/47)	3	2
								`adaptive`	✓	165 (ms)	(14/61)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	71 (ms)	(1/8)	0	1
								`adaptive`	×	69 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	83 (ms)	(1/8)	0	1
								`adaptive`	×	80 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	97 (ms)	(1/8)	0	1
								`adaptive`	×	96 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	107 (ms)	(2/8)	0	2
								`adaptive`	×	104 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/8)	0	2
								`adaptive`	×	103 (ms)	(2/8)	0	(2/0)
					One cell falsely claims modified content (other cell shared)	4	$\mathcal{V}_{trap}$	`learn`	✓	156 (ms)	(4/53)	3	3
								`adaptive`	✓	152 (ms)	(14/46)	3	(2/2)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	167 (ms)	(4/59)	4	3
								`adaptive`	✓	169 (ms)	(14/71)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	71 (ms)	(1/8)	0	1
								`adaptive`	×	69 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	82 (ms)	(1/8)	0	1
								`adaptive`	×	80 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	97 (ms)	(1/8)	0	1
								`adaptive`	×	95 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	108 (ms)	(2/8)	0	2
								`adaptive`	×	105 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/8)	0	2
								`adaptive`	×	103 (ms)	(2/8)	0	(2/0)
					No transition can be executed	5	$\mathcal{V}_{trap}$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	62 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	61 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	61 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	✓	63 (ms)	(1/2)	0	0
								`adaptive`	✓	62 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{flow}$	`learn`	✓	66 (ms)	(1/2)	0	0
								`adaptive`	✓	63 (ms)	(1/2)	0	(0/0)

Berkeley

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^{Σ}	# instances
Berkeley	1	9	4	`crowd`	Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`learn`	×	340 (ms)	(1/6)	0	1
								`adaptive`	×	95 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{trap}}^{*}$	`learn`	×	85 (ms)	(1/6)	0	1
								`adaptive`	×	77 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}$	`learn`	×	69 (ms)	(1/6)	0	1
								`adaptive`	×	67 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	81 (ms)	(1/6)	0	1
								`adaptive`	×	77 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	106 (ms)	(1/6)	0	1
								`adaptive`	×	94 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	107 (ms)	(2/6)	0	2
								`adaptive`	×	100 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{trap}}^{*}, \mathcal{V}_{flow}$	`learn`	×	104 (ms)	(2/6)	0	2
								`adaptive`	×	98 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	104 (ms)	(2/6)	0	2
								`adaptive`	×	100 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	103 (ms)	(2/6)	0	2
								`adaptive`	×	98 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	75 (ms)	(2/6)	0	2
								`adaptive`	×	73 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}$	`learn`	×	88 (ms)	(2/6)	0	2
								`adaptive`	×	82 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	110 (ms)	(3/6)	0	3
								`adaptive`	×	105 (ms)	(3/6)	0	(3/0)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(3/6)	0	3
								`adaptive`	×	102 (ms)	(3/6)	0	(3/0)
					One cell falsely claims exclusive access (other cell claims shared ownership)	4	$\mathcal{V}_{trap}$	`learn`	✓	190 (ms)	(4/21)	3	2
								`adaptive`	✓	266 (ms)	(14/29)	3	(2/2)
							${\mathcal{V}_{trap}}^{*}$	`learn`	✓	144 (ms)	(4/21)	3	2
								`adaptive`	✓	150 (ms)	(14/29)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	85 (ms)	(1/8)	0	1
								`adaptive`	×	67 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	86 (ms)	(1/8)	0	1
								`adaptive`	×	77 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	96 (ms)	(1/8)	0	1
								`adaptive`	×	93 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/8)	0	2
								`adaptive`	×	100 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	102 (ms)	(2/8)	0	2
								`adaptive`	×	100 (ms)	(2/8)	0	(2/0)
					One cell falsely claims exclusive access (other cell claims unexclusive access)	4	$\mathcal{V}_{trap}$	`learn`	✓	150 (ms)	(6/41)	4	4
								`adaptive`	✓	150 (ms)	(22/44)	4	(3/3)
							${\mathcal{V}_{trap}}^{*}$	`learn`	✓	145 (ms)	(4/33)	3	2
								`adaptive`	✓	151 (ms)	(14/46)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	69 (ms)	(1/8)	0	1
								`adaptive`	×	67 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	82 (ms)	(1/8)	0	1
								`adaptive`	×	78 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	96 (ms)	(1/8)	0	1
								`adaptive`	×	93 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/8)	0	2
								`adaptive`	×	103 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	101 (ms)	(2/8)	0	2
								`adaptive`	×	99 (ms)	(2/8)	0	(2/0)
					No transition can be executed	10	$\mathcal{V}_{trap}$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	61 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{trap}}^{*}$	`learn`	✓	62 (ms)	(1/2)	0	0
								`adaptive`	✓	60 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	62 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	✓	65 (ms)	(1/2)	0	0
								`adaptive`	✓	61 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{flow}$	`learn`	✓	65 (ms)	(1/2)	0	0
								`adaptive`	✓	63 (ms)	(1/2)	0	(0/0)

Synapse

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
Synapse	1	5	3	`crowd`	Two cells are dirty at the same time	3	$\mathcal{V}_{trap}$	`learn`	✓	96 (ms)	(3/36)	3	4
								`adaptive`	✓	92 (ms)	(12/26)	3	(3/3)
							$\mathcal{V}_{trap^*}$	`learn`	✓	104 (ms)	(2/37)	3	2
								`adaptive`	✓	127 (ms)	(14/32)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	67 (ms)	(1/6)	0	1
								`adaptive`	×	65 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon^*}$	`learn`	×	77 (ms)	(1/6)	0	1
								`adaptive`	×	78 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	89 (ms)	(1/6)	0	1
								`adaptive`	×	88 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	96 (ms)	(2/6)	0	2
								`adaptive`	×	96 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{siphon^*}, \mathcal{V}_{flow}$	`learn`	×	94 (ms)	(2/6)	0	2
								`adaptive`	×	93 (ms)	(2/6)	0	(2/0)
					One cell falsely claims exclusive access (other cell claims unexclusive access)	4	$\mathcal{V}_{trap}$	`learn`	✓	94 (ms)	(3/38)	3	3
								`adaptive`	✓	116 (ms)	(14/29)	3	(2/2)
							$\mathcal{V}_{trap^*}$	`learn`	✓	105 (ms)	(2/37)	3	2
								`adaptive`	✓	126 (ms)	(14/32)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	67 (ms)	(1/8)	0	1
								`adaptive`	×	65 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon^*}$	`learn`	×	77 (ms)	(1/8)	0	1
								`adaptive`	×	77 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	89 (ms)	(1/8)	0	1
								`adaptive`	×	89 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	98 (ms)	(2/8)	0	2
								`adaptive`	×	96 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon^*}, \mathcal{V}_{flow}$	`learn`	×	97 (ms)	(2/8)	0	2
								`adaptive`	×	94 (ms)	(2/8)	0	(2/0)
					No transition can be executed	4	$\mathcal{V}_{trap}$	`learn`	✓	61 (ms)	(1/2)	0	0
								`adaptive`	✓	59 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{trap^*}$	`learn`	✓	62 (ms)	(1/2)	0	0
								`adaptive`	✓	59 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}$	`learn`	✓	61 (ms)	(1/2)	0	0
								`adaptive`	✓	58 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon^*}$	`learn`	✓	61 (ms)	(1/2)	0	0
								`adaptive`	✓	58 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{flow}$	`learn`	✓	63 (ms)	(1/2)	0	0
								`adaptive`	✓	60 (ms)	(1/2)	0	(0/0)

FutureBus+

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
FutureBus+	1	22	9	`crowd`	Two cells have pending (right) changes at the same time	3	$\mathcal{V}_{trap}$	`learn`	✓	12 (s)	(17/268)	7	7
								`adaptive`	✓	4 (s)	(2/349)	7	(16/16)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	1 (s)	(5/61)	5	3
								`adaptive`	✓	731 (ms)	(22/108)	5	(3/3)
							$\mathcal{V}_{siphon}$	`learn`	×	82 (ms)	(1/6)	0	1
								`adaptive`	×	83 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	96 (ms)	(1/6)	0	1
								`adaptive`	×	97 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	128 (ms)	(1/6)	0	1
								`adaptive`	×	125 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	142 (ms)	(2/6)	0	2
								`adaptive`	×	136 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	140 (ms)	(2/6)	0	2
								`adaptive`	×	135 (ms)	(2/6)	0	(2/0)
					One cell falsely claims exclusive access (other cell claims shared ownership)	4	$\mathcal{V}_{trap}$	`learn`	×	85 (ms)	(1/8)	0	1
								`adaptive`	×	82 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{trap}}^*$	`learn`	×	102 (ms)	(1/8)	0	1
								`adaptive`	×	100 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}$	`learn`	×	91 (ms)	(1/8)	0	1
								`adaptive`	×	87 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	107 (ms)	(1/8)	0	1
								`adaptive`	×	96 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	123 (ms)	(1/8)	0	1
								`adaptive`	×	119 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	143 (ms)	(2/8)	0	2
								`adaptive`	×	136 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{trap}}^*, \mathcal{V}_{flow}$	`learn`	×	145 (ms)	(2/8)	0	2
								`adaptive`	×	113 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	148 (ms)	(2/8)	0	2
								`adaptive`	×	143 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	139 (ms)	(2/8)	0	2
								`adaptive`	×	140 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	103 (ms)	(2/8)	0	2
								`adaptive`	×	98 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*$	`learn`	×	109 (ms)	(2/8)	0	2
								`adaptive`	×	108 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	157 (ms)	(3/8)	0	3
								`adaptive`	×	148 (ms)	(3/8)	0	(3/0)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*, \mathcal{V}_{flow}$	`learn`	×	145 (ms)	(3/8)	0	3
								`adaptive`	×	153 (ms)	(3/8)	0	(3/0)

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
FutureBus+ contd.	1	22	9	`crowd`	Two cells have pending changes at the same time	3	$\mathcal{V}_{trap}$	`learn`	✓	136 (s)	(53/402)	10	10
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	17 (s)	(16/100)	6	4
								`adaptive`	✓	451 (ms)	(22/75)	4	(3/3)
							$\mathcal{V}_{siphon}$	`learn`	×	89 (ms)	(1/8)	0	1
								`adaptive`	×	82 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	102 (ms)	(1/8)	0	1
								`adaptive`	×	94 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{\mathit{flow}}$	`learn`	×	123 (ms)	(1/8)	0	1
								`adaptive`	×	121 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{\mathit{flow}}$	`learn`	×	137 (ms)	(2/8)	0	2
								`adaptive`	×	137 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{\mathit{flow}}$	`learn`	×	143 (ms)	(2/8)	0	2
								`adaptive`	×	134 (ms)	(2/8)	0	(2/0)
					Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`learn`	×	89 (ms)	(1/8)	0	1
								`adaptive`	×	86 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{trap}}^*$	`learn`	×	97 (ms)	(1/8)	0	1
								`adaptive`	×	97 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}$	`learn`	×	87 (ms)	(1/8)	0	1
								`adaptive`	×	89 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	×	97 (ms)	(1/8)	0	1
								`adaptive`	×	99 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{\mathit{flow}}$	`learn`	×	127 (ms)	(1/8)	0	1
								`adaptive`	×	119 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{\mathit{flow}}$	`learn`	×	143 (ms)	(2/8)	0	2
								`adaptive`	×	144 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{trap}}^*, \mathcal{V}_{\mathit{flow}}$	`learn`	×	142 (ms)	(2/8)	0	2
								`adaptive`	×	136 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{\mathit{flow}}$	`learn`	×	148 (ms)	(2/8)	0	2
								`adaptive`	×	144 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^*, \mathcal{V}_{\mathit{flow}}$	`learn`	×	141 (ms)	(2/8)	0	2
								`adaptive`	×	131 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	100 (ms)	(2/8)	0	2
								`adaptive`	×	95 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*$	`learn`	×	110 (ms)	(2/8)	0	2
								`adaptive`	×	107 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{\mathit{flow}}$	`learn`	×	149 (ms)	(3/8)	0	3
								`adaptive`	×	146 (ms)	(3/8)	0	(3/0)
							${\mathcal{V}_{trap}}^*, {\mathcal{V}_{siphon}}^*, \mathcal{V}_{\mathit{flow}}$	`learn`	×	154 (ms)	(3/8)	0	3
								`adaptive`	×	148 (ms)	(3/8)	0	(3/0)
					No transition can be executed	108	$\mathcal{V}_{trap}$	`learn`	✓	87 (ms)	(1/2)	0	0
								`adaptive`	✓	88 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{trap}}^*$	`learn`	✓	89 (ms)	(1/2)	0	0
								`adaptive`	✓	82 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}$	`learn`	✓	85 (ms)	(1/2)	0	0
								`adaptive`	✓	83 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{siphon}}^*$	`learn`	✓	88 (ms)	(1/2)	0	0
								`adaptive`	✓	86 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{\mathit{flow}}$	`learn`	✓	92 (ms)	(1/2)	0	0
								`adaptive`	✓	87 (ms)	(1/2)	0	(0/0)

Firefly

Name	$\lvert I \rvert$	$\lvert T \rvert$	$\lvert \Sigma \rvert$	Topology	Property	$\lvert B \rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^{Σ}	# instances
Firefly	1	16	4	`crowd`	Two cells are dirty at the same time	3	$\mathcal{V}_{trap}$	`learn`	×	69 (ms)	(1/6)	0	1
								`adaptive`	×	67 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{trap}}^{*}$	`learn`	×	82 (ms)	(1/6)	0	1
								`adaptive`	×	79 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}$	`learn`	×	69 (ms)	(1/6)	0	1
								`adaptive`	×	67 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	81 (ms)	(1/6)	0	1
								`adaptive`	×	79 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	98 (ms)	(1/6)	0	1
								`adaptive`	×	96 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	107 (ms)	(2/6)	0	2
								`adaptive`	×	104 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{trap}}^{*}, \mathcal{V}_{flow}$	`learn`	×	104 (ms)	(2/6)	0	2
								`adaptive`	×	100 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	106 (ms)	(2/6)	0	2
								`adaptive`	×	102 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/6)	0	2
								`adaptive`	×	98 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	76 (ms)	(2/6)	0	2
								`adaptive`	×	72 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}$	`learn`	×	87 (ms)	(2/6)	0	2
								`adaptive`	×	82 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	110 (ms)	(3/6)	0	3
								`adaptive`	×	108 (ms)	(3/6)	0	(3/0)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	107 (ms)	(3/6)	0	3
								`adaptive`	×	104 (ms)	(3/6)	0	(3/0)
					Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`learn`	×	69 (ms)	(1/6)	0	1
								`adaptive`	×	66 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{trap}}^{*}$	`learn`	×	80 (ms)	(1/6)	0	1
								`adaptive`	×	79 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}$	`learn`	×	69 (ms)	(1/6)	0	1
								`adaptive`	×	67 (ms)	(1/6)	0	(1/0)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	82 (ms)	(1/6)	0	1
								`adaptive`	×	79 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	100 (ms)	(1/6)	0	1
								`adaptive`	×	96 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	107 (ms)	(2/6)	0	2
								`adaptive`	×	103 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{trap}}^{*}, \mathcal{V}_{flow}$	`learn`	×	103 (ms)	(2/6)	0	2
								`adaptive`	×	99 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	108 (ms)	(2/6)	0	2
								`adaptive`	×	104 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	102 (ms)	(2/6)	0	2
								`adaptive`	×	99 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	76 (ms)	(2/6)	0	2
								`adaptive`	×	69 (ms)	(2/6)	0	(2/0)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}$	`learn`	×	87 (ms)	(2/6)	0	2
								`adaptive`	×	82 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	110 (ms)	(3/6)	0	3
								`adaptive`	×	110 (ms)	(3/6)	0	(3/0)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	106 (ms)	(3/6)	0	3
								`adaptive`	×	104 (ms)	(3/6)	0	(3/0)

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^{Σ}	# instances
Firefly contd.	1	16	4	`crowd`	One cell falsely claims exclusive access (other cell is shared)	4	$\mathcal{V}_{trap}$	`learn`	×	95 (ms)	(4/21)	2	2
								`adaptive`	×	90 (ms)	(6/24)	2	(2/1)
							${\mathcal{V}_{trap}}^{*}$	`learn`	×	112 (ms)	(4/21)	2	2
								`adaptive`	×	108 (ms)	(6/24)	2	(2/1)
							$\mathcal{V}_{siphon}$	`learn`	×	17 (ms)	(1/8)	0	1
								`adaptive`	×	68 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	83 (ms)	(1/8)	0	1
								`adaptive`	×	80 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	99 (ms)	(1/8)	0	1
								`adaptive`	×	97 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	139 (ms)	(5/21)	2	2
								`adaptive`	×	131 (ms)	(7/24)	2	(3/1)
							${\mathcal{V}_{trap}}^{*}, \mathcal{V}_{flow}$	`learn`	×	138 (ms)	(5/21)	2	3
								`adaptive`	×	132 (ms)	(7/24)	2	(3/1)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	107 (ms)	(2/8)	0	2
								`adaptive`	×	103 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	103 (ms)	(2/8)	0	2
								`adaptive`	×	99 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	100 (ms)	(5/21)	2	2
								`adaptive`	×	93 (ms)	(7/24)	2	(3/1)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}$	`learn`	×	120 (ms)	(5/21)	2	3
								`adaptive`	×	112 (ms)	(7/24)	2	(3/1)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	143 (ms)	(6/21)	2	4
								`adaptive`	×	138 (ms)	(8/24)	2	(4/1)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	141 (ms)	(6/21)	2	4
								`adaptive`	×	133 (ms)	(8/24)	2	(4/1)
					One cell falsely claims exclusive access (other cell claims dirty access)	4	$\mathcal{V}_{trap}$	`learn`	×	81 (ms)	(1/8)	0	1
								`adaptive`	×	69 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{trap}}^{*}$	`learn`	×	81 (ms)	(1/8)	0	1
								`adaptive`	×	78 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}$	`learn`	×	17 (ms)	(1/8)	0	1
								`adaptive`	×	67 (ms)	(1/8)	0	(1/0)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	81 (ms)	(1/8)	0	1
								`adaptive`	×	80 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	100 (ms)	(1/8)	0	1
								`adaptive`	×	96 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	109 (ms)	(2/8)	0	2
								`adaptive`	×	103 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{trap}}^{*}, \mathcal{V}_{flow}$	`learn`	×	104 (ms)	(2/8)	0	2
								`adaptive`	×	101 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	109 (ms)	(2/8)	0	2
								`adaptive`	×	104 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(2/8)	0	2
								`adaptive`	×	101 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	76 (ms)	(2/8)	0	2
								`adaptive`	×	71 (ms)	(2/8)	0	(2/0)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}$	`learn`	×	88 (ms)	(2/8)	0	2
								`adaptive`	×	83 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	111 (ms)	(3/8)	0	3
								`adaptive`	×	111 (ms)	(3/8)	0	(3/0)
							${\mathcal{V}_{trap}}^{*}, {\mathcal{V}_{siphon}}^{*}, \mathcal{V}_{flow}$	`learn`	×	109 (ms)	(3/8)	0	3
								`adaptive`	×	106 (ms)	(3/8)	0	(3/0)
					No transition can be executed	16	$\mathcal{V}_{trap}$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	63 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{trap}}^{*}$	`learn`	✓	65 (ms)	(1/2)	0	0
								`adaptive`	✓	63 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}$	`learn`	✓	65 (ms)	(1/2)	0	0
								`adaptive`	✓	63 (ms)	(1/2)	0	(0/0)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	63 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{flow}$	`learn`	✓	67 (ms)	(1/2)	0	0
								`adaptive`	✓	64 (ms)	(1/2)	0	(0/0)

Dragon

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
Dragon	1	23	5	`crowd`	Two cells are dirty at the same time	3	$\mathcal{V}_{trap}$	`learn`	✓	177 (ms)	(4/33)	3	2
								`adaptive`	✓	144 (ms)	(14/45)	3	(2/2)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	190 (ms)	(4/33)	3	2
								`adaptive`	✓	162 (ms)	(14/45)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	67 (ms)	(1/6)	0	1
								`adaptive`	×	66 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	81 (ms)	(1/6)	0	1
								`adaptive`	×	79 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	102 (ms)	(1/6)	0	1
								`adaptive`	×	100 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	113 (ms)	(2/6)	0	2
								`adaptive`	×	111 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	110 (ms)	(2/6)	0	2
								`adaptive`	×	105 (ms)	(2/6)	0	(2/0)
					Two cells are exclusive at the same time	3	$\mathcal{V}_{trap}$	`learn`	✓	882 (ms)	(13/73)	4	5
								`adaptive`	✓	118 (ms)	(16/40)	4	(4/4)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	1.4 (s)	(12/106)	5	3
								`adaptive`	✓	242 (ms)	(14/70)	4	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	68 (ms)	(1/6)	0	1
								`adaptive`	×	67 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	81 (ms)	(1/6)	0	1
								`adaptive`	×	78 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	103 (ms)	(1/6)	0	1
								`adaptive`	×	102 (ms)	(1/6)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	111 (ms)	(2/6)	0	2
								`adaptive`	×	111 (ms)	(2/6)	0	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	110 (ms)	(2/6)	0	2
								`adaptive`	×	106 (ms)	(2/6)	0	(2/0)
					One cell falsely claims exclusive access (other cell claims dirty and shared access)	4	$\mathcal{V}_{trap}$	`learn`	✓	40 (s)	(25/236)	6	6
								`adaptive`	✓	210 (ms)	(30/83)	6	(4/4)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	302 (ms)	(7/76)	5	3
								`adaptive`	✓	189 (ms)	(22/87)	5	(3/3)
							$\mathcal{V}_{siphon}$	`learn`	×	70 (ms)	(1/8)	0	1
								`adaptive`	×	67 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	84 (ms)	(1/8)	0	1
								`adaptive`	×	80 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	104 (ms)	(1/8)	0	1
								`adaptive`	×	102 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	113 (ms)	(2/8)	0	2
								`adaptive`	×	113 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	111 (ms)	(2/8)	0	2
								`adaptive`	×	109 (ms)	(2/8)	0	(2/0)
					One cell falsely claims exclusive access (other cell claims shared access)	4	$\mathcal{V}_{trap}$	`learn`	✓	4.6 (s)	(16/138)	5	4
								`adaptive`	✓	181 (ms)	(22/47)	4	(3/3)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	2 (s)	(15/73)	4	2
								`adaptive`	✓	169 (ms)	(14/66)	4	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	69 (ms)	(1/8)	0	1
								`adaptive`	×	68 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	82 (ms)	(1/8)	0	1
								`adaptive`	×	80 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	105 (ms)	(1/8)	0	1
								`adaptive`	×	102 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	114 (ms)	(2/8)	0	2
								`adaptive`	×	113 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	111 (ms)	(2/8)	0	2
								`adaptive`	×	108 (ms)	(2/8)	0	(2/0)
					One cell falsely claims exclusive access (other cell claims dirty access)	4	$\mathcal{V}_{trap}$	`learn`	✓	184 (ms)	(4/50)	4	3
								`adaptive`	✓	143 (ms)	(14/40)	3	(2/2)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	193 (ms)	(4/39)	3	2
								`adaptive`	✓	163 (ms)	(14/45)	3	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	69 (ms)	(1/8)	0	1
								`adaptive`	×	68 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	83 (ms)	(1/8)	0	1
								`adaptive`	×	79 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	105 (ms)	(1/8)	0	1
								`adaptive`	×	103 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	113 (ms)	(2/8)	0	2
								`adaptive`	×	114 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	111 (ms)	(2/8)	0	2
								`adaptive`	×	110 (ms)	(2/8)	0	(2/0)

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
Dragon contd.	1	23	5	`crowd`	One cell falsely claims dirty access (other cell claims shared access)	4	$\mathcal{V}_{trap}$	`learn`	✓	3.3 (s)	(15/176)	4	4
								`adaptive`	✓	149 (ms)	(14/46)	3	(2/2)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	2.1 (s)	(15/73)	4	2
								`adaptive`	✓	172 (ms)	(14/66)	4	(2/2)
							$\mathcal{V}_{siphon}$	`learn`	×	71 (ms)	(1/8)	0	1
								`adaptive`	×	68 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	82 (ms)	(1/8)	0	1
								`adaptive`	×	78 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	104 (ms)	(1/8)	0	1
								`adaptive`	×	102 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	114 (ms)	(2/8)	0	2
								`adaptive`	×	112 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	110 (ms)	(2/8)	0	2
								`adaptive`	×	108 (ms)	(2/8)	0	(2/0)
					One cell falsely claims dirty access (other cell claims dirty and shared access)	4	$\mathcal{V}_{trap}$	`learn`	×	72 (ms)	(1/8)	0	1
								`adaptive`	×	174 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{trap}{}^*$	`learn`	×	84 (ms)	(1/8)	0	1
								`adaptive`	×	85 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}$	`learn`	×	73 (ms)	(1/8)	0	1
								`adaptive`	×	70 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	86 (ms)	(1/8)	0	1
								`adaptive`	×	83 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	×	107 (ms)	(1/8)	0	1
								`adaptive`	×	106 (ms)	(1/8)	0	(1/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{flow}$	`learn`	×	116 (ms)	(2/8)	0	2
								`adaptive`	×	114 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}{}^*, \mathcal{V}_{flow}$	`learn`	×	113 (ms)	(2/8)	0	2
								`adaptive`	×	111 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	117 (ms)	(2/8)	0	2
								`adaptive`	×	116 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	100 (ms)	(2/8)	0	2
								`adaptive`	×	113 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	80 (ms)	(2/8)	0	2
								`adaptive`	×	78 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}{}^*, \mathcal{V}_{siphon}{}^*$	`learn`	×	90 (ms)	(2/8)	0	2
								`adaptive`	×	93 (ms)	(2/8)	0	(2/0)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	124 (ms)	(3/8)	0	3
								`adaptive`	×	123 (ms)	(3/8)	0	(3/0)
							$\mathcal{V}_{trap}{}^*, \mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	119 (ms)	(3/8)	0	3
								`adaptive`	×	118 (ms)	(3/8)	0	(3/0)
					No transition can be executed	16	$\mathcal{V}_{trap}$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	61 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	62 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}$	`learn`	✓	64 (ms)	(1/2)	0	0
								`adaptive`	✓	60 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	✓	63 (ms)	(1/2)	0	0
								`adaptive`	✓	62 (ms)	(1/2)	0	(0/0)
							$\mathcal{V}_{flow}$	`learn`	✓	65 (ms)	(1/2)	0	0
								`adaptive`	✓	63 (ms)	(1/2)	0	(0/0)

B.6 Termination detection

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^{Σ}	# instances
Termination detection	1	6	4	×	Two tokens moving down	3	$\mathcal{V}_{trap}$	`learn`	✓	249 (ms)	(5/76)	6	3
							${\mathcal{V}_{trap}}^{*}$	`learn`	✓	243 (ms)	(5/63)	5	3
							$\mathcal{V}_{siphon}$	`learn`	×	69 (ms)	(1/6)	0	1
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	82 (ms)	(1/6)	0	1
							$\mathcal{V}_{flow}$	`learn`	✓	294 (ms)	(3/364)	3	2
					Two tokens moving up	3	$\mathcal{V}_{trap}$	`learn`	✓	402 (ms)	(5/787)	9	7
							${\mathcal{V}_{trap}}^{*}$	`learn`	✓	170 (ms)	(3/111)	4	3
							$\mathcal{V}_{siphon}$	`learn`	×	72 (ms)	(1/6)	0	1
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	82 (ms)	(1/6)	0	1
							$\mathcal{V}_{flow}$	`learn`	✓	301 (ms)	(3/364)	3	2
					No transition can be executed	7	$\mathcal{V}_{trap}$	`learn`	✓	291 (ms)	(5/495)	6	4
							${\mathcal{V}_{trap}}^{*}$	`learn`	✓	229 (ms)	(5/220)	6	3
							$\mathcal{V}_{siphon}$	`learn`	×	73 (ms)	(1/9)	0	1
							${\mathcal{V}_{siphon}}^{*}$	`learn`	×	82 (ms)	(1/9)	0	1
							$\mathcal{V}_{flow}$	`learn`	✓	303 (ms)	(3/360)	3	2

B.7 Dining cryptographers

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^{Σ}	# instances
Dining cryptographers	2	8	12	`ring`	Paying cryptographer	4	$\mathcal{V}_{trap}$	`learn`	✓	9.6 (s)	(10/1249)	16	13
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							${\mathcal{V}_{trap}}^{*}$	`learn`	✓	30 (s)	(13/1996)	16	11
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							$\mathcal{V}_{siphon}$	`learn`	✓	5.1 (s)	(9/742)	12	11
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	✓	54 (s)	(16/2355)	21	15
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							$\mathcal{V}_{flow}$	`learn`	`oot`	20 (min)	(×/×)	×	×
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
					No paying cryptographer	2	$\mathcal{V}_{trap}$	`learn`	✓	2.1 (s)	(10/235)	8	7
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							${\mathcal{V}_{trap}}^{*}$	`learn`	✓	18 (s)	(11/536)	11	9
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							$\mathcal{V}_{siphon}$	`learn`	✓	3.1 (s)	(9/428)	12	11
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							${\mathcal{V}_{siphon}}^{*}$	`learn`	✓	54 (s)	(16/702)	18	14
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)
							$\mathcal{V}_{flow}$	`learn`	`oot`	20 (min)	(×/×)	×	×
								`adaptive`	`oot`	20 (min)	(×/×)	×	(×/×)

B.8 Leader election

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
Herman	2	11	2	×	Only followers	1	$\mathcal{V}_{trap}$	`learn`	✓	69 (ms)	(1/4)	1	1
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	81 (ms)	(1/4)	1	1
							$\mathcal{V}_{siphon}$	`learn`	×	74 (ms)	(3/6)	1	2
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	89 (ms)	(3/6)	1	2
							$\mathcal{V}_{flow}$	`learn`	×	98 (ms)	(1/8)	1	2
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	105 (ms)	(4/6)	1	3
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	102 (ms)	(4/6)	1	3
Israeli-Jafon	2	10	2	×	Only followers	1	$\mathcal{V}_{trap}$	`learn`	✓	69 (ms)	(1/4)	1	1
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	82 (ms)	(1/4)	1	1
							$\mathcal{V}_{siphon}$	`learn`	×	75 (ms)	(3/6)	1	2
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	88 (ms)	(3/6)	1	2
							$\mathcal{V}_{flow}$	`learn`	×	98 (ms)	(1/8)	1	2
							$\mathcal{V}_{siphon}, \mathcal{V}_{flow}$	`learn`	×	104 (ms)	(4/6)	1	3
							$\mathcal{V}_{siphon}{}^*, \mathcal{V}_{flow}$	`learn`	×	102 (ms)	(4/6)	1	3

B.9 Token passing

Name	$\lvert I\rvert$	$\lvert T\rvert$	$\lvert\Sigma\rvert$	Topology	Property	$\lvert B\rvert$	Interpretations	Mode	Result	Time	# expl. abs.	# 2^Σ	# instances
With invariant	2	3	2	×	There is no token	2	$\mathcal{V}_{trap}$	`learn`	✓	69 (ms)	(1/6)	1	1
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	81 (ms)	(1/6)	1	1
							$\mathcal{V}_{siphon}$	`learn`	×	70 (ms)	(1/10)	1	2
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	81 (ms)	(1/10)	1	2
							$\mathcal{V}_{flow}$	`learn`	✓	89 (ms)	(1/12)	1	1
					There are many tokens	3	$\mathcal{V}_{trap}$	`learn`	✓	86 (ms)	(4/23)	2	3
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	110 (ms)	(4/19)	2	3
							$\mathcal{V}_{siphon}$	`learn`	×	65 (ms)	(1/7)	0	1
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	76 (ms)	(1/7)	0	1
							$\mathcal{V}_{flow}$	`learn`	✓	90 (ms)	(1/35)	2	2
Without invariant	2	3	2	`bow`	There is no token	2	$\mathcal{V}_{trap}$	`learn`	✓	68 (ms)	(1/6)	1	1
								`adaptive`	✓	190 (ms)	(1/6)	1	(1/0)
							$\mathcal{V}_{trap}{}^*$	`learn`	✓	11 (ms)	(1/6)	1	1
								`adaptive`	✓	100 (ms)	(1/6)	1	(1/0)
							$\mathcal{V}_{siphon}$	`learn`	×	70 (ms)	(1/10)	1	2
								`adaptive`	×	69 (ms)	(1/10)	1	(2/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	83 (ms)	(1/10)	1	2
								`adaptive`	×	83 (ms)	(1/10)	1	(2/1)
							$\mathcal{V}_{flow}$	`learn`	✓	88 (ms)	(1/12)	1	1
								`adaptive`	✓	104 (ms)	(1/12)	1	(1/0)
					There are many tokens	3	$\mathcal{V}_{trap}$	`learn`	×	80 (ms)	(3/18)	2	3
								`adaptive`	×	77 (ms)	(11/17)	2	(4/3)
							$\mathcal{V}_{trap}{}^*$	`learn`	×	94 (ms)	(3/18)	2	3
								`adaptive`	×	92 (ms)	(11/17)	2	(4/3)
							$\mathcal{V}_{siphon}$	`learn`	×	65 (ms)	(1/7)	0	1
								`adaptive`	×	65 (ms)	(1/7)	0	(1/0)
							$\mathcal{V}_{siphon}{}^*$	`learn`	×	76 (ms)	(1/7)	0	1
								`adaptive`	×	76 (ms)	(1/7)	0	(1/0)
							$\mathcal{V}_{flow}$	`learn`	✓	99 (ms)	(1/35)	2	2
								`adaptive`	✓	99 (ms)	(7/21)	2	(2/2)
							$\mathcal{V}_{trap}, \mathcal{V}_{siphon}$	`learn`	×	87 (ms)	(4/18)	2	4
								`adaptive`	×	82 (ms)	(12/17)	2	(5/3)
							$\mathcal{V}_{trap}{}^*, \mathcal{V}_{siphon}{}^*$	`learn`	×	100 (ms)	(4/18)	2	4
								`adaptive`	×	95 (ms)	(12/17)	2	(5/3)