Semantic Symmetry in Transducers

Antonio Abu Nassar

Semantic Symmetry in Transducers

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Antonio Abu Nassar

Submitted to the Senate
of the Technion — Israel Institute of Technology
Iyyar 5782 Haifa May 2022

This research was carried out under the supervision of Dr. Shaull Almagor, in the Faculty of Computer Science.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author's doctoral research period, the most up-to-date versions of which being:

Antonio Abu Nassar and Shaull Almagor. Simulation by rounds of letter-to-letter transducers, full version. *ArXiv*, abs/2105.01512, 2022.

Antonio Abu Nassar and Shaull Almagor. Simulation by rounds of letter-to-letter transducers. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, 3:1–3:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

Model checking is a verification paradigm for systems, where we are given a system and a specification and we check whether every possible computation of the system satisfies the specification.

Often, systems over multiple processes exhibit some type of symmetry in their structure or their behaviour. Symmetry is also commonly manifested in the specifications of multiple process systems. When such symmetries are present in the system or the specification, they can be exploited by the designer and the verification algorithm to alleviate some of the complexity of model checking, as well as to gain insight into the behaviour of the system. For instance, symmetric systems enable the designer to use only representative specifications where iteration over the process identities was formerly needed. Thus, we want to decide whether a given system or specification exhibits symmetry.

Symmetry is not a well-defined concept and might come in various forms, each capturing a different characteristic behaviour. In this work, we focus on *process symmetry*, where every process j has a corresponding input and output signal i_j and o_j, and the input and output alphabets of the model are 2^I and 2^O respectively. Process symmetry addresses the scenario where the identities of the processes may be scrambled – that is, permuted. For example, if the input $\{i_1, i_2\}$ is generated, the system might actually receive an input $\{i_7, i_4\}$. Then, a system exhibits process symmetry if, intuitively, its outputs are permuted in a similar way to the inputs. Unfortunately, deterministic systems that are process symmetric are extremely naive, as process symmetry is too restrictive for them.

In our setting, each of the system and the specification are modelled by a finite-state machine called a *transducer*. In addition, words are partitioned into rounds, and a transducer $\mathcal{T}$ is *k-round symmetric* if for every permutation π of the signals and for every input word x, we can scramble the letters within each round in $\pi(x)$ to obtain x', such that the output of $\mathcal{T}$ on x' is itself a scramble of the output of $\mathcal{T}$ on x. In other words, when $\mathcal{T}$ is round symmetric, there is a way to scramble the permuted input, so that the resulting output is a scramble of the permuted output (i.e. the "expected" output in process symmetry).

Round symmetry is *semantic*: it does not consider the structure, rather the behaviour of the system. Round symmetry gives rise to the following decision problems:

- ★ In *fixed round symmetry* we are given a transducer $\mathcal{T}$ and $k > 0$, and we need to decide whether $\mathcal{T}$ is k-round symmetric.
- ★ In *existential round symmetry* we are given a transducer $\mathcal{T}$, and we need to decide whether there exists $k > 0$ such that $\mathcal{T}$ is k-round symmetric.

Notice that round symmetry defines a property of a transducer. The way we approach the

decision problems is by first translating the definition of symmetry to a definition of a relation between two transducers, called *round simulation*, then showing that round symmetry can be reduced to round simulation, we solve it as such.

A transducer $\mathcal{T}_2$ *k-round simulates* transducer $\mathcal{T}_1$ if for every input word x, we can scramble the letters within each round in x, such that the output of $\mathcal{T}_2$ on the scrambled word is itself a scramble of the output of $\mathcal{T}_1$ on x. In fact, we consider a somewhat more elaborate setting, by also allowing the inputs to $\mathcal{T}_1$ to be restricted to some regular language Λ.

The mapping between the input words for $\mathcal{T}_1$ and the scrambled input for $\mathcal{T}_2$ is called the *simulation mapping* and is also studied in the continuation of this work.

Round symmetry and round simulation – the decision problems and their solutions, upper and lower bounds and the simulation mapping – are the main contribution of this work. Additionally, several more notions of symmetry are presented and discussed, including variations of round symmetry, and symmetry in the setting of infinite words.

We use tools and techniques from logic, algebra and automata theory.

Notation and Abbreviations

DCW	deterministic co-Büchi automaton over words	41
DFA	deterministic finite automaton	9
NFA	nondeterministic finite automaton	9
PA	Presburger arithmetic	21
RR	Round Robin	12

Chapter 1

Introduction

Reactive systems interact with their environment by receiving inputs, corresponding to the state of the environment, and sending outputs, which describe actions of the system. Finite-state reactive systems are often modeled by *transducers* – finite-state machines over alphabets Σ_I and Σ_O of inputs and outputs, respectively, which read an input letter in Σ_I, and respond with an output in Σ_O. Such transducers are amenable to automatic verification of certain properties (e.g., LTL model-checking), and are therefore useful in practice. Nonetheless, modeling complex systems may result in huge transducers, which makes verification procedures prohibitively expensive, and makes understanding the constructed transducers difficult.

A common approach to gain a better understanding of a transducer (or more generally, any system) is *simulation* [Mil71], whereby a transducer $\mathcal{T}_1$ is simulated by a "simpler" transducer $\mathcal{T}_2$ in such a way that model checking is easier on $\mathcal{T}_2$, and the correctness of the desired property is preserved under the simulation. Usually, "simpler" means smaller, as in standard simulation [Mil71] and fair simulation [HKR97], but one can also view e.g., linearization of concurrent programs [HW87] as a form of simulation by a simpler machine.

In this work, we introduce and study new notions of simulation and of equivalence for transducers, based on *rounds*: consider an input word $x \in \Sigma_I^*$ whose length is $k \cdot R$ for some $k, R > 0$. We divide the word into R disjoint infixes of length k, each called a round of w. We then say that two words $x, x' \in \Sigma_I^{kR}$ are k-round equivalent, denoted $x' \asymp_k x$, if x' is obtained from x by permuting the letters within each round of x. For example *abcabc* and *cbaacb* are 3-round equivalent, since *cba* is a permutation of *abc* and so is *acb*. Example 3.1 presents a pair of words that are 3-round equivalent but not 4-round equivalent. We now say that a transducer $\mathcal{T}_1$ is *k-round simulated* by a transducer $\mathcal{T}_2$, denoted $\mathcal{T}_1 \prec_k \mathcal{T}_2$, if for every[1] input $x \in \Sigma_I^{kR}$ we can find $x' \asymp_k x$ such that the outputs of $\mathcal{T}_1$ on x and $\mathcal{T}_2$ on x', denoted y, y' respectively, are also round equivalent: $y' \asymp_k y$. Intuitively, $\mathcal{T}_1 \prec_k \mathcal{T}_2$ means that every behaviour of $\mathcal{T}_1$ is captured by $\mathcal{T}_2$, up to permutations within each round. When we have both $\mathcal{T}_1 \prec_k \mathcal{T}_2$ and $\mathcal{T}_2 \prec_k \mathcal{T}_1$, we say that they are k-round equivalent and denote this by $\mathcal{T}_1 \equiv_k \mathcal{T}_2$.

The benefit of k-round simulation is twofold. First, it may serve as an alternative simulation technique for reducing the state space while maintaining the correctness of certain properties. Second, we argue that k-round simulation is in and of itself a design concern. Indeed, in certain scenarios, as follows, we can naturally design a transducer $\mathcal{T}_2$ that performs a certain task in

[1] Our formal definition allows to also restrict the input to some regular language $\Lambda \subseteq \Sigma_I^*$, see Chapter 3.

an ideal, but not realistic, way, and we want to check that an existing design, namely $\mathcal{T}_1$, is simulated by this ideal. In particular, this is useful when dealing with systems that naturally work in rounds, such as schedulers (e.g., Round Robin, cf. Example 3.2), arbiters, and other resource allocation systems.

We now demonstrate both benefits by an example.

▶ **Example 1.1.** Consider a monitor M for the fairness of a distributed system with 10 processes $\mathcal{P} = \{1, \ldots, 10\}$. At each timestep, M receives as input the ID of the process currently working. The monitor then verifies that in each round of 10 steps, every process works exactly once. As long as this holds, the monitor keeps outputting `safe`; otherwise, it outputs `error`.

M can be modeled by a transducer $\mathcal{T}_1$ that keeps track of the set of processes that have worked in the current round. Thus, the transducer has at least 2^{10} states, as it needs to keep track of the subset of processes that have been seen.

It is not hard to see that $\mathcal{T}_1$ is 10-round simulated by an "ideal" transducer $\mathcal{T}_2$ which expects to see the processes in the order $1, \ldots, 10$. This transducer needs roughly 10 states, as it only needs to know the index of the next process it expects to see.

Now, suppose we want to verify some correctness property which is invariant to permutations of the processes within each round of length 10, such as "if there is no `error`, then Process 3 works at least once every 20 steps". Then we can verify this against the much smaller $\mathcal{T}_2$.

The notion of k-round simulation arises naturally in the setting of *process symmetry*. There, the input and output alphabets are $\Sigma_I = 2^I$ and $\Sigma_O = 2^O$ respectively, where $I = \{i_1, \ldots, i_m\}$ and $O = \{o_1, \ldots, o_m\}$ represent signals corresponding to m processes. Process symmetry addresses the scenario where the identities of the processes may be scrambled. For example, if the input $\{i_1, i_2\}$ is generated, the system might actually receive an input $\{i_7, i_4\}$. A system exhibits process symmetry if, intuitively, its outputs are permuted in a similar way to the inputs. Unfortunately, deterministic systems that are process symmetric are extremely naive, as process symmetry is too restrictive for them. While this can be overcome using probabilistic systems, as studied in [Alm20], it is also desirable to find a definition that is suited for deterministic systems. As we show in Chapter 6, k-round simulation provides such a definition.

The main contributions of this work are as follows. We introduce the notion of k-round simulation and k-round equivalence, and define two decision problems pertaining to them: in *fixed round simulation* we need to decide whether $\mathcal{T}_1 \prec_k \mathcal{T}_2$ for a given value of k, and in *existential round simulation* we need to decide whether there exists some value of k for which $\mathcal{T}_1 \prec_k \mathcal{T}_2$ holds. In fact, we consider a somewhat more elaborate setting, by also allowing the inputs to $\mathcal{T}_1$ to be restricted to some regular language Λ. We solve the first problem by reducing it to the containment of two nondeterministic automata. For the second problem, things become considerably more difficult, and the solution requires several constructions, as well as tools such as Presburger arithmetic and Parikh's theorem. In addition, we demonstrate the usefulness of the definitions in relation to process symmetry.

Related Work

Simulation relations between systems are a well studied notion. We refer the reader to [CHVB18, Chapter 13] and references therein for an exposition. The connection of our notion with standard

simulation is only up to motivation, as our measure is semantic: it does not directly relate to the state space; instead, it refers to the *behaviour* of the system rather than its structure.

On the technical level, our work is closely related to *commutative automata* [BS73] and *jumping automata* [FPS15; MZ12] — models of automata capable of reading their input in a discontinuous manner, by jumping from one letter to another. Indeed, our notion of round simulation essentially allows the simulating transducer to read the letters within rounds in a discontinuous manner. This similarity is manifested implicitly in Section 5.2, where we encounter similar structures as e.g. the commutative closure in [Hof20] (although the analysis here has a different purpose).

Finally, the initial motivation for this work comes from *process symmetry* [Alm20; CEFJ96; ES96; ID96; LNRS16]. We explore the connections in depth in Chapter 6.

Thesis Organization

The rest of this work is organized as follows. In Chapter 2 we present some basic definitions used throughout our research. In Chapter 3 we introduce k-round simulation and equivalence, define the relevant decision problems, and study some fundamental properties of the definitions. In Chapter 4 we solve fixed round simulation, while developing some technical tools and characterizations that are reused later. Chapter 5 is our main technical result, where we develop a solution for existential round simulation. In particular, in Section 5.1 we give an overview of the solution, before going through the technical details in Section 5.2. In Section 5.3 we give lower bounds for the existential setting. In Chapter 6 we use round simulation to obtain a definition of process symmetry for deterministic transducers, along with an algorithm for deciding it, and in Chapter 7 we go into further detail about the mapping between rounds of transducers within a simulation. Other notions of symmetry and simulation are considered in Chapter 8, and finally, we discuss these variants and conclude with some open problems in Chapter 9.

Chapter 2

Preliminaries

Automata. A *deterministic finite automaton (DFA)* is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to Q$ is a transition function, and $F \subseteq Q$ is the set of accepting states.

The *run* of $\mathcal{A}$ on a word $w = \sigma_0 \cdot \sigma_2 \cdots \sigma_{n-1} \in \Sigma^*$ is a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_{i+1} = \delta(q_i, \sigma_i)$ for all $0 \leq i < n$. The run is *accepting* if $q_n \in F$. A word $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if the run of $\mathcal{A}$ on w is accepting. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. We also consider *nondeterministic finite automata (NFAs)*, where $\delta : Q \times \Sigma \to 2^Q$ and there can be multiple initial states. Then, a run of $\mathcal{A}$ on a word $w \in \Sigma^*$ as above is a sequence of states $q_0, q_1, \ldots, q_n$ such that q_0 is an initial state and $q_{i+1} \in \delta(q_i, \sigma_i)$ for all $0 \leq i < n$. Analogously to the deterministic setting, the language of $\mathcal{A}$ is the set of words that have an accepting run. We denote by $|\mathcal{A}|$ the number of states of $\mathcal{A}$.

As usual, we denote by δ^* the transition function lifted to words. For states q, q' and $w \in \Sigma^*$, we write $q \xrightarrow{w}_{\mathcal{A}} q'$ if $q' \in \delta^*(q, w)$. That is, if there is a run of $\mathcal{A}$ from q to q' while reading w.

An NFA $\mathcal{A}$ can be viewed as a morphism from Σ^* to the monoid $\mathbb{B}^{Q \times Q}$ of $Q \times Q$ Boolean matrices, where we associate with a letter $\sigma \in \Sigma$ its *type* $\tau_{\mathcal{A}}(\sigma) \in \mathbb{B}^{Q \times Q}$ defined by $(\tau_{\mathcal{A}}(\sigma))_{q,q'} = 1$ if $q \xrightarrow{\sigma}_{\mathcal{A}} q'$, and $(\tau_{\mathcal{A}}(\sigma))_{q,q'} = 0$ otherwise. We lift the definition of types to Σ^* by defining, for a word $w = \sigma_1 \cdots \sigma_n \in \Sigma^*$, its type as $\tau_{\mathcal{A}}(w) = \tau_{\mathcal{A}}(\sigma_1) \cdots \tau_{\mathcal{A}}(\sigma_n)$ where the concatenation denotes Boolean matrix product. It is easy to see that $(\tau_{\mathcal{A}}(w))_{q,q'} = 1$ iff $q \xrightarrow{w}_{\mathcal{A}} q'$. For example, the types of the letters a and b in the automaton in Figure 2.1 are the 3×3 matrices

$$\tau_{\mathcal{A}}(a) = \begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \end{array} \begin{array}{c} \begin{array}{ccc} q_0 & q_1 & q_2 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}, \quad \tau_{\mathcal{A}}(b) = \begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \end{array} \begin{array}{c} \begin{array}{ccc} q_0 & q_1 & q_2 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array},$$

and the type of the word $w = ab$ in the transducer in Figure 2.1 is the matrix

$$\tau_{\mathcal{A}}(w) = \begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \end{array} \begin{array}{c} \begin{array}{ccc} q_0 & q_1 & q_2 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array} = \tau_{\mathcal{A}}(a) \cdot \tau_{\mathcal{A}}(b).$$

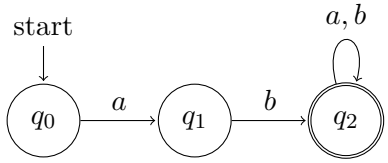

Figure 2.1: A nondeterministic automaton with one initial state q_0 and one accepting state q_2.

Transducers. Consider two sets Σ_I and Σ_O representing input and output alphabets, respectively. A Σ_I/Σ_O *transducer* is $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \boldsymbol{\ell} \rangle$ where Q, $q_0 \in Q$, and $\delta : Q \times \Sigma_I \to Q$ are as in a DFA, and $\boldsymbol{\ell} : Q \to \Sigma_O$ is a labelling function on the states. For a word $w \in \Sigma_I^*$, consider the run $\rho = q_0, \ldots, q_n$ of $\mathcal{T}$ on w. We define its output $\boldsymbol{\ell}(\rho) = \boldsymbol{\ell}(q_1) \cdots \boldsymbol{\ell}(q_n) \in \Sigma_O^*$, and we define the output of $\mathcal{T}$ on w to be $\mathcal{T}(w) = \boldsymbol{\ell}(\rho)$. Observe that we ignore the labelling of the initial state in the run, so that the length of the output matches that of the input.

Words and rounds. Consider a word $w = \sigma_0 \cdots \sigma_{n-1} \in \Sigma^*$. We denote its length by $|w|$, and for $0 \leq i \leq j < |w|$ we define $w[i : j] = \sigma_i \cdots \sigma_j$. For $k > 0$, if $|w| = kR$ for some $R \in \mathbb{N}$, then for every $0 \leq r < R$ we refer to $w[rk : r(k+1) - 1]$ as the *r-th round* in w (of length k), and we write $w = \gamma_0 \cdots \gamma_{R-1}$ where γ_r is the r-th round. We emphasize that k indicates the length of each round, not the number of rounds.

In particular, throughout this work we consider words $(x, y) \in (\Sigma_I^k \times \Sigma_O^k)^*$ and their rounds of length k. In such cases, we sometimes use the natural embedding of $(\Sigma_I^k \times \Sigma_O^k)^*$ in $(\Sigma_I \times \Sigma_O)^*$ and in $\Sigma_I^* \times \Sigma_O^*$, and refer to these sets interchangeably.

Parikh vectors and permutations. Consider an alphabet Σ. For a word $w \in \Sigma^*$ and a letter $\sigma \in \Sigma$, we denote by $\#_\sigma(w)$ the number of occurrences of σ in w. The *Parikh map* $\mathfrak{P} : \Sigma^* \to \mathbb{N}^\Sigma$ maps every word $w \in \Sigma^*$ to a *Parikh vector* $\mathfrak{P}(w) \in \mathbb{N}^\Sigma$, where $\mathfrak{P}(w)(\sigma) = \#_\sigma(w)$. We lift this to languages by defining, for $L \subseteq \Sigma^*$, $\mathfrak{P}(L) = \{\mathfrak{P}(w) : w \in L\}$.

For $\boldsymbol{p} \in \mathbb{N}^\Sigma$ (in the following we consistently denote vectors in $\mathbb{N}^\Sigma$ by bold letters) we write $|\boldsymbol{p}| = \sum_{\sigma \in \Sigma} \boldsymbol{p}(\sigma)$. In particular, for a word $w \in \Sigma^*$ we have $|\mathfrak{P}(w)| = |w|$.

By Parikh's theorem [Par66], for every NFA $\mathcal{A}$ we have that $\mathfrak{P}(L(\mathcal{A}))$ is a *semilinear set* – that is, a finite union of sets of the form $\{ \boldsymbol{p} + \lambda_1 \boldsymbol{s_1} + \ldots + \lambda_1 \boldsymbol{s_m} \mid \lambda_1, \ldots, \lambda_m \in \mathbb{N} \}$ where $\boldsymbol{p}, \boldsymbol{s_1}, \ldots, \boldsymbol{s_m} \in \mathbb{N}^d$.

Consider words $x, y \in \Sigma^*$. We say that x is a *permutation* of y if $\mathfrak{P}(x) = \mathfrak{P}(y)$ (indeed, in this case y can be obtained from x by permuting its letters). In particular this implies $|x| = |y|$.

Chapter 3

Round Simulation and Round Equivalence

Consider two k-round words $x, y \in \Sigma^{kR}$ with the same number of rounds R, and denote their rounds by $x = \alpha_0 \cdots \alpha_{R-1}$ and $y = \beta_0 \cdots \beta_{R-1}$. We say that x and y are *k-round equivalent*, denoted $x \asymp_k y$ (or $x \asymp y$, when k is clear from context)[1], if for every $0 \leq r < R$ we have that $\mathfrak{P}(\alpha_r) = \mathfrak{P}(\beta_r)$. That is, $x \asymp y$ iff the r-th round of y is a permutation of the r-th round of x, for every r. Indeed, $\asymp$ is an equivalence relation.

▶ **Example 3.1** (Round-equivalence for words)**.** Consider the words $x = abaabbabbbaa$ and $y = baabbaabbaba$ over the alphabet $\Sigma = \{a, b\}$. Looking at the words as 3-round words, one can see in Table 3.1 that rounds of length 3 in y are all permutations of those in x, which gives $x \asymp_3 y$. However, looking at the rounds of length 4 of x, y, the number of occurrences of b already in the first round of x and of y is different, so $x \not\asymp_4 y$, as illustrated in Table 3.2.

Table 3.1: x and y are 3-round equivalent

x	aba	abb	abb	baa
y	baa	bba	abb	aba

Table 3.2: x and y are not 4-round equivalent

x	*abaa*	bbab	bbaa
y	*baab*	baab	baba

Let Σ_I and Σ_O be input and output alphabets, let $\Lambda \subseteq \Sigma_I^*$ be a regular language, and let $k > 0$. Consider two Σ_I/Σ_O transducers $\mathcal{T}_1$ and $\mathcal{T}_2$. We say that $\mathcal{T}_2$ *k-round simulates* $\mathcal{T}_1$ *restricted to* Λ, denoted $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, if for every k-round word $x \in \Lambda$ there exists a k-round word $x' \in \Sigma_I^*$ such that $x \asymp_k x'$ and $\mathcal{T}_1(x) \asymp_k \mathcal{T}_2(x')$.

Intuitively, $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ if for every input word $x \in \Lambda$, we can permute each round of length k in x to obtain a new word x', such that the outputs of $\mathcal{T}_1$ on x and of $\mathcal{T}_2$ on x' are k-round equivalent. Note that the definition is not symmetric: the input x for $\mathcal{T}_1$ is universally quantified, while x' is chosen according to x. We illustrate this in Example 3.3.

If $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ and $\mathcal{T}_2 \prec_{k,\Lambda} \mathcal{T}_1$ we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are *k-round equivalent restricted to* Λ, denoted $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$. In the special case where $\Lambda = \Sigma_I^*$ (i.e., when we require the simulation to hold for every input), we omit it from the subscript and write $\mathcal{T}_1 \prec_k \mathcal{T}_2$.

[1] Conveniently, our symbol for round equivalence is a rounded equivalence.

▶ **Example 3.2** (Round Robin). We consider a simple version of the Round Robin (RR) scheduler for three processes $\mathcal{P} = \{0, 1, 2\}$. In each time step, the scheduler outputs either a singleton set containing the ID of the process whose request is granted, or an empty set if the process whose turn it is did not make a request. Depending on the ID $i \in \{0, 1, 2\}$ of the first process, we model the scheduler as a $2^{\mathcal{P}}/2^{\mathcal{P}}$ transducer $\mathcal{T}_i = \left\langle 2^{\mathcal{P}}, 2^{\mathcal{P}}, Q, q_{(i-1)\%3}, \delta, \boldsymbol{\ell} \right\rangle$ depicted in Figure 3.1, where % is the mod operator, $Q = \{q_0, q_1, q_2, q'_0, q'_1, q'_2\}$, $\delta(q_i, \sigma) = q_{(i+1)\%3}$ if $i + 1 \in \sigma$ and $\delta(q_i, \sigma) = q'_{(i+1)\%3}$ otherwise, $\boldsymbol{\ell}(q_i) = \{i\}$ and $\boldsymbol{\ell}(q'_i) = \emptyset$.

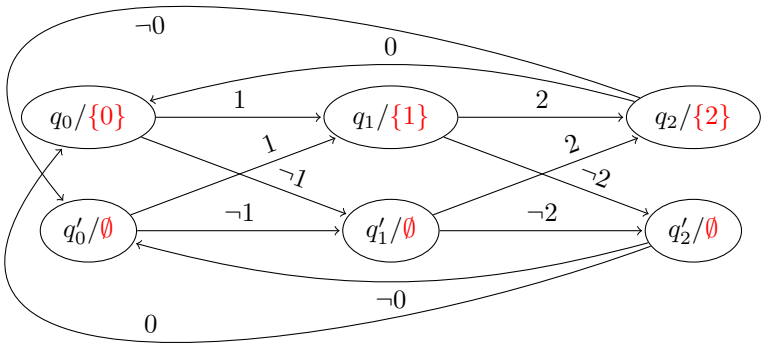

Figure 3.1: The transducer $\mathcal{T}_i$ for RR, initial state omitted. The input letters σ and $\neg\sigma$ mean all letters from $2^{\mathcal{P}}$ that, respectively, contain or do not contain σ. The labels are written in red.

Technically, the initial state changes the behaviour of $\mathcal{T}_i$ significantly (e.g. $\mathcal{T}_0(\{0\}\{2\}\{1\}) = \{0\}\emptyset\emptyset$ whereas $\mathcal{T}_1(\{0\}\{2\}\{1\}) = \emptyset\{2\}\emptyset$). Conceptually, however, changing the initial state does not alter the behaviour, as long as the requests are permuted accordingly. This is captured by round equivalence, as follows.

We argue that, if we allow permutation of the input letters, then the set of processes whose requests are granted in each round is independent of the start state. This is equivalent to saying $\mathcal{T}_0 \equiv_k \mathcal{T}_j$ for $j \in \{1, 2\}$, which indeed holds: if $j = 1$ then we permute all rounds of the form $\sigma_0\sigma_1\sigma_2$ to $\sigma_1\sigma_2\sigma_0$, and similarly if $j = 2$ then we permute all rounds to $\sigma_2\sigma_0\sigma_1$. It is easy to see that the run of $\mathcal{T}_i$ on the permuted input grants outputs that k-round equivalent to the output of $\mathcal{T}_0$ on the non-permuted input.

In Example 3.2, the transducers satisfied not only round simulation, but also round equivalence. We now show that this is not always the case for simulating transducers.

▶ **Example 3.3** (Round simulation is not symmetric). Consider the Σ_I/Σ_O transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ over the alphabet $\Sigma_I = \{a, b\}$ and $\Sigma_O = \{0, 1\}$, depicted in Figure 3.2. We claim that $\mathcal{T}_1 \prec_2 \mathcal{T}_2$

Figure 3.2: Transducers $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (right) illustrate the asymmetry in the definition of round equivalence (see Example 3.3).

but $\mathcal{T}_2 \not\prec_2 \mathcal{T}_1$. Starting with the latter, observe that $\mathcal{T}_2(ab) = 00$, but $\mathcal{T}_1(ab) = \mathcal{T}_1(ba) = 01$. Since $00 \not\approx_2 01$, we have $\mathcal{T}_2 \not\prec_2 \mathcal{T}_1$.

We turn to show that $\mathcal{T}_1 \prec_2 \mathcal{T}_2$. Observe that for every input word of the form $x \in (ab+ba)^m$, we have $\mathcal{T}_1(x) = (01)^m$, and $x \succ_2 (ba)^m$. So in this case we have that $\mathcal{T}_2((ba)^m) = (10)^m \succ_2 (01)^m$. Next, for $x \in (ab+ba)^m \cdot bb \cdot w$ for some $w \in \Sigma_I^*$ we have $\mathcal{T}_1(x) = (01)^m 011^{|w|}$ and $x \succ_2 (ba)^m \cdot bb \cdot w$, for which $\mathcal{T}_2((ba)^m \cdot bb \cdot w) = (01)^m 101^{|w|} \succ_2 \mathcal{T}_1(x)$. The case where $x \in (ab+ba)^m \cdot aa \cdot w$ is handled similarly. We conclude that $\mathcal{T}_1 \prec_2 \mathcal{T}_2$.

Round simulation and round equivalence give rise to the following decision problems:

- ★ In *fixed round simulation* (resp. *fixed round equivalence*) we are given transducers $\mathcal{T}_1, \mathcal{T}_2$, an NFA for the language Λ, and $k > 0$ in unary, and we need to decide whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ (resp. whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$).
- ★ In *existential round simulation* (resp. *existential round equivalence*) we are given transducers $\mathcal{T}_1, \mathcal{T}_2$ and an NFA for the language Λ, and we need to decide whether there exists $k > 0$ such that $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ (resp. $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$).

In the following we identify Λ with an NFA (or DFA) for it, as we do not explicitly rely on its description.

We start by showing that deciding equivalence (both fixed and existential) is reducible, in polynomial time, to the respective simulation problem.

▶ **Lemma 3.4.** *Fixed (resp. existential) round equivalence is Turing reducible in polynomial time to fixed (resp. existential) round simulation.*

Proof. First, we can clearly reduce fixed round equivalence to fixed round simulation: given an algorithm that decides, given $\mathcal{T}_1, \mathcal{T}_2$, Λ and $k > 0$, whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, we can decide whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$ by using it twice to decide whether both $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ and $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ hold.

A slightly more careful examination shows that the same approach can be taken to reduce existential round equivalence to existential round simulation, using the following observation: if $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, then for every $m \in \mathbb{N}$ it holds that $\mathcal{T}_1 \prec_{mk,\Lambda} \mathcal{T}_2$. Indeed, we can simply group every m rounds of length k and treat them as a single round of length mk.

Now, given an algorithm that decides, given $\mathcal{T}_1, \mathcal{T}_2$ and Λ, whether there exists $k > 0$ such that $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, we can decide whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$ by using the algorithm twice to decide whether there exists k_1 such that $\mathcal{T}_1 \prec_{k_1,\Lambda} \mathcal{T}_2$ and k_2 such that $\mathcal{T}_2 \prec_{k_2,\Lambda} \mathcal{T}_1$ hold. If there are no such k_1, k_2, then clearly $\mathcal{T}_1 \not\equiv_{k,\Lambda} \mathcal{T}_2$. However, if there are such k_1, k_2, then by the observation above we have $\mathcal{T}_1 \equiv_{k_1 k_2,\Lambda} \mathcal{T}_2$ (we can also take $\mathrm{lcm}(k_1, k_2)$ instead of $k_1 k_2$). ◀

By Lemma 3.4, for the purpose of upper-bounds, we focus henceforth on round simulation.

Chapter 4

Deciding Fixed Round Simulation

In this chapter we show decidability of fixed round simulation (and, by Lemma 3.4, fixed round equivalence). The tools we develop will be used in Chapter 5 to handle the existential variant.

Let Σ_I and Σ_O be input and output alphabets. Consider two Σ_I/Σ_O transducers $\mathcal{T}_1$ and $\mathcal{T}_2$, and let $\Lambda \subseteq \Sigma_I^*$ and $k > 0$. In order to decide whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, we proceed as follows. First, we cast the problem to a problem about deterministic automata. Then, we translate rounds into letters, by working over the alphabets Σ_I^k and Σ_O^k. We construct an NFA, dubbed the *permutation closure*, for each transducer $\mathcal{T}$, that captures the behaviour of $\mathcal{T}$ on words and their permutations. Intuitively, the NFA takes as input a word $(x, y) \in (\Sigma_I^k \times \Sigma_O^k)^*$, guesses a round equivalent word $x' \asymp x$, and verifies that $\mathcal{T}(x') \asymp \mathcal{T}(x)$. We then show that round simulation amounts to deciding the containment of these NFAs.

We now turn to give the details of the construction of these NFAs.

The trace DFA. Consider a transducer $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \boldsymbol{\ell} \rangle$, we define its *trace DFA* $\mathtt{Tr}(\mathcal{T}) = \langle \Sigma_I \times \Sigma_O, Q \cup \{q_\perp\}, q_0, \eta, Q \rangle$ where for $q \in Q$ and $(\sigma, \sigma') \in \Sigma_I \times \Sigma_O$ we define $\eta(q, (\sigma, \sigma')) = \delta(q, \sigma)$ if $\mathcal{T}^q(\sigma) = \sigma'$ and $\eta(q, (\sigma, \sigma')) = q_\perp$ otherwise. $q_\perp$ is a rejecting sink.

$\mathtt{Tr}(\mathcal{T})$ captures the behaviour of $\mathcal{T}$ in that $L(\mathtt{Tr}(\mathcal{T})) = \{ (x, y) \in (\Sigma_I \times \Sigma_O)^* \mid \mathcal{T}(x) = y \}$.

The permutation closure NFA. Consider an NFA $\mathcal{N} = \langle \Sigma_I \times \Sigma_O, S, s_0, \eta, F \rangle$, and let $k > 0$. We obtain from $\mathcal{N}$ an NFA $\mathtt{Perm}_k(\mathcal{N}) = \left\langle \Sigma_I^k \times \Sigma_O^k, S, s_0, \mu, F \right\rangle$ where the alphabet is $\Sigma_I^k \times \Sigma_O^k$, and the transition function μ is defined as follows. For a letter $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ and a state $s \in S$, we think of (α, β) as a word in $(\Sigma_I \times \Sigma_O)^*$. Then we have

$$\mu(s, (\alpha, \beta)) = \bigcup \{ \eta^*(s, (\alpha', \beta')) \mid \mathfrak{P}(\alpha') = \mathfrak{P}(\alpha) \wedge \mathfrak{P}(\beta) = \mathfrak{P}(\beta') \} . \tag{4.1}$$

That is, upon reading (α, β), $\mathtt{Perm}_k(\mathcal{N})$ can move to any state s' that is reachable in $\mathcal{N}$ from s by reading a permutation of α, β (denoted α', β'). Recall that for two words x, x' we have that $x \asymp_k x'$ if for every two corresponding rounds α, α' in x and x' we have $\mathfrak{P}(\alpha) = \mathfrak{P}(\alpha')$. Thus, we have the following.

▶ **Observation 4.1.** *In the notations above, it holds that* $L(\mathtt{Perm}_k(\mathcal{N})) = \{(x, y) \in \Sigma_I^* \times \Sigma_O^* \mid \exists x' \asymp_k x, y' \asymp_k y, (x', y') \in L(\mathcal{N}) \wedge \ |x| = |y| = kR$ *for some* $R \in \mathbb{N}\}$.

Since the transition function of $\texttt{Perm}_k(\mathcal{N})$ is only defined using permutations of its input letters, we have the following property, which we refer to as *permutation invariance*:

▶ **Observation 4.2** (Permutation invariance). *For every state $s \in S$ and letters $(\alpha, \beta), (\alpha', \beta') \in \Sigma_I^k \times \Sigma_O^k$, if $\mathfrak{P}(\alpha) = \mathfrak{P}(\alpha')$ and $\mathfrak{P}(\beta) = \mathfrak{P}(\beta')$ then $\mu(s, (\alpha, \beta)) = \mu(s, (\alpha', \beta'))$.*

Given a transducer $\mathcal{T}$, we apply the permutation closure to the trace DFA of $\mathcal{T}$. In order to account for the restriction given by $\Lambda \subseteq \Sigma_I^*$, we identify it with $\Lambda \subseteq \Sigma_I^* \times \Sigma_O^*$. We remind that Λ denotes both a language and a corresponding NFA (or DFA), so what this means is that the NFA, reading input from $\Sigma_I^* \times \Sigma_O^*$, simply ignores the second component.

▶ **Lemma 4.3.** *Consider transducers $\mathcal{T}_1, \mathcal{T}_2$, an NFA Λ and $k > 0$. Let $\mathcal{A}_1^k = \texttt{Perm}_k(\texttt{Tr}(\mathcal{T}_1) \cap \Lambda)$ (where the intersection implies the product NFA construction) and $\mathcal{A}_2^k = \texttt{Perm}_k(\texttt{Tr}(\mathcal{T}_2))$, then*

$$L(\mathcal{A}_1^k) = \{\, (x, y) \in \Sigma_I^* \times \Sigma_O^* \mid \exists x' \asymp_k x,\ \mathcal{T}_1(x') \asymp_k y \ \wedge\ |x| = |y| = kR \text{ where } R \in \mathbb{N} \wedge x' \in \Lambda \,\},$$
$$L(\mathcal{A}_2^k) = \{\, (x, y) \in \Sigma_I^* \times \Sigma_O^* \mid \exists x' \asymp_k x,\ \mathcal{T}_2(x') \asymp_k y \ \wedge\ |x| = |y| = kR \text{ where } R \in \mathbb{N} \,\}.$$

Proof. Recall that $\texttt{Tr}(\mathcal{T})$ accepts a word (x', y') iff $\mathcal{T}(x') = y'$. The claim then follows from Observation 4.1, by replacing the expression $y \asymp y' \wedge (x', y') \in L(\texttt{Tr}(\mathcal{T}))$ with the equivalent expression $\mathcal{T}(x') \asymp_k y$. ◀

We now reduce round simulation to the containment of permutation closure NFAs.

▶ **Lemma 4.4.** *Consider transducers $\mathcal{T}_1, \mathcal{T}_2$, an NFA Λ and $k > 0$. Let $\mathcal{A}_1^k = \texttt{Perm}_k(\texttt{Tr}(\mathcal{T}_1) \cap \Lambda)$ and $\mathcal{A}_2^k = \texttt{Perm}_k(\texttt{Tr}(\mathcal{T}_2))$, then $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ iff $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$.*

Proof. For the first direction, assume $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, and let $(x, y) \in L(\mathcal{A}_1^k)$. By Lemma 4.3, x and y are k-round words, and there exists a word $x' \in \Lambda$ such that $x \asymp x'$ and $\mathcal{T}_1(x') \asymp y$. Since $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, then applying the definition on x' yields that there exists a k-round word x'' such that $x' \asymp x''$ and such that $\mathcal{T}_1(x') \asymp \mathcal{T}_2(x'')$. Since $\asymp$ is an equivalence relation, it follows that $x \asymp x''$ and $\mathcal{T}_2(x'') \asymp y$, so again by Lemma 4.3 we have $(x, y) \in L(\mathcal{A}_2^k)$.

Conversely, assume $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$, we wish to prove that for every k-round word $x \in \Lambda$ there exists a word x' such that $x \asymp x'$ and $\mathcal{T}_1(x) \asymp \mathcal{T}_2(x')$. Let $x \in \Lambda$ be a k-round word, and let $y = \mathcal{T}_1(x)$, then clearly $(x, y) \in L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$ (since $x \asymp x$, $\mathcal{T}_1(x) = y \asymp y$ and $x \in \Lambda$). By Lemma 4.3, there exists x' such that $x \asymp x'$ and $\mathcal{T}_2(x') \asymp y = \mathcal{T}_1(x)$, so $\mathcal{T}_2(x') \asymp \mathcal{T}_1(x)$, thus concluding the proof. ◀

▶ **Remark 4.5.** The proof of Lemma 4.4 does not require taking the permutation closure of $\texttt{Tr}(\mathcal{T}_1) \cap \Lambda$, and it could be simplified by using instead of $\mathcal{A}_1^k$, the augmentation of $\texttt{Tr}(\mathcal{T}_1) \cap \Lambda$ to k-round words. However, such an NFA is not permutation invariant, which is key to our solution for existential round simulation. Since this simplification does not reduce the overall complexity, we use a uniform setting for both solutions.

Lemma 4.4 shows that deciding fixed round equivalence amounts to deciding containment of NFAs. By analyzing the size of the NFAs, we obtain the following.

▶ **Theorem 4.6.** *Given transducers $\mathcal{T}_1, \mathcal{T}_2$, an NFA Λ, and $k > 0$ in unary, the problem of deciding whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ is in* **PSPACE**.

Proof. Let $\mathcal{A}_1^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda)$ and $\mathcal{A}_2^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$. By Lemma 4.4, deciding whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ amounts to deciding whether $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$. Looking at the dual problem, recall that for two NFAs $\mathcal{N}_1, \mathcal{N}_2$ we have that $L(\mathcal{N}_1) \not\subseteq L(\mathcal{N}_2)$ iff there exists $w \in L(\mathcal{N}_2) \setminus L(\mathcal{N}_1)$ with $|w| \leq |\mathcal{N}_1| \cdot 2^{|\mathcal{N}_2|}$ (this follows immediately by bounding the size of an NFA for $L(\mathcal{N}_1) \cap \overline{L(\mathcal{N}_2)}$). Thus, we can decide whether $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$ by guessing a word w over $\Sigma_I^k \times \Sigma_O^k$ of single-exponential length (in the size of $\mathcal{A}_1^k$ and $\mathcal{A}_2^k$), and verifying that it is accepted by $\mathcal{A}_1^k$ and not by $\mathcal{A}_2^k$.

Observe that to this end, we do not explicitly construct $\mathcal{A}_1^k$ nor $\mathcal{A}_2^k$, as their alphabet size is exponential. Rather, we evaluate them on each letter of w based on their construction from $\mathcal{T}$. At each step we keep track of a counter for the length of w, a state of $\mathcal{A}_1^k$, and a set of states of $\mathcal{A}_2^k$. Since the number of states in $\mathcal{A}_1^k$ and $\mathcal{A}_2^k$ is the same as that of $\mathcal{T}_1$ and $\mathcal{T}_2$, this requires polynomial space.

By Savitch's theorem we have that **coNPSPACE** = **PSPACE**, and the proof is concluded. ◀

We now establish a **PSPACE**-hardness lower bound, thus concluding that the problem is **PSPACE**-complete. In fact, we show a lower bound for round equivalence. Note that a priori, this does not entail a lower bound for round simulation by Lemma 3.4, since the reduction there is a Turing reduction. However, our **PSPACE**-hardness proof actually explicitly shows the hardness of both simulation and equivalence.

▶ **Theorem 4.7.** *The problem of deciding, given transducers $\mathcal{T}_1, \mathcal{T}_2$, whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$, is* **PSPACE***-hard, even for $k = 2$ and Λ of constant size (given as a 4-state DFA).*

Proof sketch. We show a reduction from the universality problem for NFAs over alphabet $\{0,1\}$ where all states are accepting and the degree of nondeterminism is at most 2. See Appendix A for a proof of **PSPACE**-hardness of this problem and for the full reduction.

Consider an NFA $\mathcal{N} = \langle Q, \{0,1\}, \delta, q_0, Q\rangle$ where $|\delta(q,\sigma)| \leq 2$ for every $q \in Q$ and $\sigma \in \{0,1\}$. Set $\Lambda = (ab + cd)^*$. We construct two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ over input and output alphabets $\Sigma_I = \{a,b,c,d\}$ and $\Sigma_O = \{\top, \bot\}$ such that $L(\mathcal{N}) = \{0,1\}^*$ iff $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$.

Intuitively, our reduction encodes $\{0,1\}$ over $\{a,b,c,d\}$ by identifying 0 with ab and with ba, and 1 with cd and with dc. Then, $\mathcal{T}_1$ keeps outputting $\top$ for all inputs in Λ, thus mimicking a universal language in $\{0,1\}^*$ (see Figure A.1), whereas $\mathcal{T}_2$ is obtained by replacing every nondeterministic transition of $\mathcal{N}$ on e.g. 0 by two deterministic branches, on e.g. ab and ba (see Figure A.2). Hence, when we are allowed to permute ab and ba by round equivalence, we capture the nondeterminism of $\mathcal{N}$.

We show that $L(\mathcal{N}) = \{0,1\}^*$ iff $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$ by showing that permuting a word $w \in \Lambda$ essentially amounts to choosing an accepting run of $\mathcal{N}$ on the corresponding word in $\{0,1\}^*$. ◀

▶ **Corollary 4.8.** *Given transducers $\mathcal{T}_1, \mathcal{T}_2$, an NFA Λ, and $k > 0$ in unary, the problem of deciding whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ is* **PSPACE***-complete.*

Chapter 5

Deciding Existential Round Simulation

In Chapter 4, we established a method for deciding k-round simulation for a given k. This case is for when the systems in question exhibit an apparent symmetry with a round length that a developer can guess; such as RR where the round length is the number of processes involved. However, k is not necessarily given in the general sense.

We turn to solve existential round simulation. That is, given $\mathcal{T}_1, \mathcal{T}_2$ and Λ, we wish to decide whether there exists $k > 0$ such that $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$. By Lemma 4.4, this is equivalent to deciding whether there exists $k > 0$ such that $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$, as defined therein.

Recall that solving the decision problems of round simulation will aid us in solving the initial problem of round symmetry, which gave the motivation for this work. The transition between the problems is explained in Chapter 6.

5.1 Intuitive Overview

We start with an intuitive explanation of the solution and its challenges. For simplicity, assume for now $\Lambda = \Sigma_I^*$, so it can be ignored. The overall approach is to present a practical method for hunting k: in Theorem 5.1, the main result of this chapter, we give an upper bound on the minimal $k > 0$ for which $\mathcal{T}_1 \prec_k \mathcal{T}_2$, rendering the search space finite. In order to obtain this bound, we proceed as follows. Observe that for a transducer $\mathcal{T}$ and for $0 < k \neq k'$ the corresponding permutation closure NFAs $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}))$ and $\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}))$ are defined on the same state space, but differ by their alphabet ($\Sigma_I^k \times \Sigma_O^k$ vs $\Sigma_I^{k'} \times \Sigma_O^{k'}$). Thus, by definition, these NFAs obtained from an increasing round length form infinitely many distinct automata. Nonetheless, there are only finitely many possible types of letters (indeed, at most $|\mathbb{B}^{Q \times Q}| = 2^{|Q|^2}$). Therefore, there are only finitely many *type profiles* for NFAs – that is, the set of letter types occurring in the NFA – up to multiplicities of the letter types.

Recall that by Lemma 4.4, we have that $\mathcal{T}_1 \prec_k \mathcal{T}_2$ iff $L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1))) \subseteq L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2)))$. Intuitively, one could hope that if $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_i))$ and $\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_i))$ have the same type profile, for each $i \in \{1, 2\}$, then $L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1))) \subseteq L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2)))$ iff $L(\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_1))) \subseteq L(\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_2)))$. Then, if one can bound the index k after which no further type profiles are encountered, then the problem reduces to checking a finite number of containments.

Unfortunately, this is not the case, the reason being that the mapping of letters induced by the equal type profiles $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1))$ and $\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_1))$ may differ from the mapping induced by $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$ and $\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_2))$, and thus one cannot translate language containment between the two pairs. We overcome this difficulty, however, by working from the start with product automata that capture the structure of both $\mathcal{T}_1$ and $\mathcal{T}_2$ simultaneously, and thus unify the letter mapping. We dub them *redundant product automata* for their apparent redundancy.

We are now left with the problem of bounding the minimal k after which no new type profiles appear. In order to provide this bound, we show that for every type profile, the set of indices in which it occurs is semilinear. Then, by finding a bound for each type profile, we obtain the overall bound. The main result of this chapter is the following.

▶ **Theorem 5.1.** *Given transducers $\mathcal{T}_1, \mathcal{T}_2$ and Λ, we can effectively compute $K_0 > 0$ such that if $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ for some $k \in \mathbb{N}$, then $\mathcal{T}_1 \prec_{k',\Lambda} \mathcal{T}_2$ for some $k' \leq K_0$.*

Which by Lemma 4.4 immediately entails the following.

▶ **Corollary 5.2.** *Existential round simulation is decidable.*

We prove Theorem 5.1 in Section 5.2, organized as follows. We start by lifting the definition of types in an NFA to Parikh vectors, and show how these relate to the NFA (in Lemma 5.3). We then introduce Presburger arithmetic and its relation to Parikh's theorem. In Lemma 5.4 we show that the set of Parikh vectors that share a type τ is definable in Presburger arithmetic, which provides the first main step towards our bound.

We then proceed to define the redundant product automata mentioned above, which serve to unify the types between $\mathcal{T}_1$ and $\mathcal{T}_2$. In Observations 5.5 and 5.6 we formalize the connection of these products to the transducers $\mathcal{T}_1$ and $\mathcal{T}_2$. Then, we formally define the type profiles and prove in Lemma 5.7 that they exhibit a semilinear behaviour. Finally, in Lemma 5.8 we prove that when two redundant product automata have the same type profile, then the containment mentioned above can be shown. Combining these results, we obtain Theorem 5.1.

A flow diagram for the proof is illustrated in Figure 5.1.

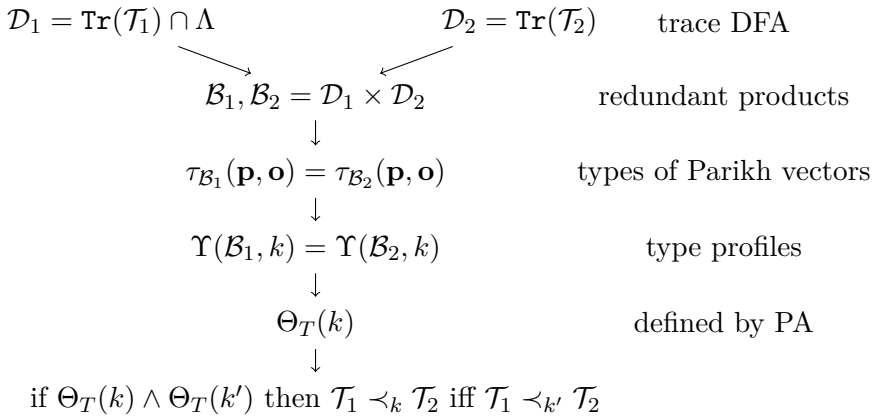

Figure 5.1: A flow diagram for the proof in Section 5.2.

5.2 Proof of Theorem 5.1

Type matrices of Parikh vectors. Consider the alphabet $\Sigma_I^k \times \Sigma_O^k$ for some $k > 0$. Recall that by Observation 4.2, permutation closure NFAs are permutation invariant, and from Chapter 2, the type of a word in an NFA is the transition matrix it induces. In particular, for permutation invariant NFAs, two letters $(\alpha, \beta), (\alpha', \beta') \in \Sigma_I^k \times \Sigma_O^k$ with $\mathfrak{P}(\alpha) = \mathfrak{P}(\alpha')$ and $\mathfrak{P}(\beta) = \mathfrak{P}(\beta')$ have the same type.

Following this, we now lift the definition of types to Parikh vectors. Consider an NFA $\mathcal{N} = \langle \Sigma_I \times \Sigma_O, S, s_0, \eta, F \rangle$, and let $\boldsymbol{p} \in \mathbb{N}^{\Sigma_I}, \boldsymbol{o} \in \mathbb{N}^{\Sigma_O}$ be Parikh vectors with $|\boldsymbol{p}| = |\boldsymbol{o}| = k$. We define the type $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) \in \mathbb{B}^{S \times S}$ to be $\tau_{\mathtt{Perm}_k(\mathcal{N})}(\alpha, \beta)$ where $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ are such that $\mathfrak{P}(\alpha) = \boldsymbol{p}$ and $\mathfrak{P}(\beta) = \boldsymbol{o}$. By permutation invariance, this is well-defined, i.e. is independent of the choice of α and β.

Note that we use different automata to extract the type of words of different lengths. We obtain a more uniform description as follows.

▶ **Lemma 5.3.** *In the notations above, for every $s_1, s_2 \in S$, we have that $(\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}))_{s_1, s_2} = 1$ iff there exists $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ with $\mathfrak{P}(\alpha) = \boldsymbol{p}$ and $\mathfrak{P}(\beta) = \boldsymbol{o}$ such that $s_1 \xrightarrow{(\alpha,\beta)}_{\mathtt{Perm}_k(\mathcal{N})} s_2$.*

Proof. By the definitions preceding the lemma, we have that $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau_{\mathtt{Perm}_k(\mathcal{N})}(\alpha', \beta')$ for some $(\alpha', \beta') \in \Sigma_I^k \times \Sigma_O^k$ are such that $\mathfrak{P}(\alpha') = \boldsymbol{p}$ and $\mathfrak{P}(\beta') = \boldsymbol{o}$. According to the transition function of $\mathtt{Perm}_k(\mathcal{N})$ (as defined in Chapter 4), for every $s_1, s_2 \in S$ we have that $s_1 \xrightarrow{(\alpha',\beta')}_{\mathtt{Perm}_k(\mathcal{N})} s_2$ iff there exist $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ with $\mathfrak{P}(\alpha) = \mathfrak{P}(\alpha') = \boldsymbol{p}$ and $\mathfrak{P}(\beta) = \mathfrak{P}(\beta') = \boldsymbol{o}$ such that $s_1 \xrightarrow{(\alpha,\beta)}_{\mathcal{N}} s_2$. Since the type encodes the reachable pairs of states, this concludes the proof. ◀

Presburger arithmetic. The first ingredient in the proof of Theorem 5.1 is to characterize the set of Parikh vectors whose type is some fixed matrix $\tau \in \mathbb{B}^{Q \times Q}$. For this characterization, we employ the first-order theory of the naturals with addition and order $\mathrm{Th}(\mathbb{N}, 0, 1, +, <, =)$, commonly known as *Presburger arithmetic (PA)*. We do not give a full exposition of PA but refer the reader to [Haa18] (and references therein) for a survey. In the following we briefly cite the results we need.

For our purposes, a PA formula $\varphi(x_1, \ldots, x_d)$, where $x_1, \ldots, x_d$ are free variables, is evaluated over $\mathbb{N}^d$, and *defines* the set $\left\{ (a_1, \ldots, a_d) \in \mathbb{N}^d \;\middle|\; (a_1, \ldots, a_d) \models \varphi(x_1, \ldots, x_d) \right\}$. For example, the formula $\varphi(x_1, x_2) := x_1 < x_2 \wedge \exists y. x_1 = 2y$ defines the set $\{ (a, b) \in \mathbb{N}^2 \mid a < b \wedge a \text{ is even} \}$.

A fundamental result about PA is that the definable sets in PA are exactly the semilinear sets. In particular, Parikh's theorem states that for every NFA $\mathcal{A}$, $\mathfrak{P}(L(\mathcal{A}))$ is PA definable. In fact, by [VSS05], one can efficiently construct a linear-sized existential PA formula for $\mathfrak{P}(L(\mathcal{A}))$. We can now show that the set of Parikh vectors whose type is τ is PA definable.

▶ **Lemma 5.4.** *Consider an NFA $\mathcal{N} = \langle \Sigma_I \times \Sigma_O, S, s_0, \eta, F \rangle$, and a type $\tau \in \mathbb{B}^{S \times S}$, then the set $\left\{ (\boldsymbol{p}, \boldsymbol{o}) \in \mathbb{N}^{\Sigma_I} \times \mathbb{N}^{\Sigma_O} \;\middle|\; \tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau \right\}$ is PA definable.*

Proof. Let $\tau \in \mathbb{B}^{S \times S}$, and consider a Parikh vector $(\boldsymbol{p}, \boldsymbol{o}) \in \mathbb{N}^{\Sigma_I} \times \mathbb{N}^{\Sigma_O}$ with $k = |\boldsymbol{p}| = |\boldsymbol{o}|$. By Lemma 5.3, we have that $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau$ iff the following holds for every $s_1, s_2 \in S$: we have $\tau_{s_1, s_2} = 1$ iff there exists a letter $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ such that $\mathfrak{P}(\alpha) = \boldsymbol{p}, \mathfrak{P}(\beta) = \boldsymbol{o}$, and $s_1 \xrightarrow{(\alpha,\beta)}_{\mathcal{N}} s_2$.

Consider $s_1, s_2 \in S$ and define $\mathcal{N}_{s_2}^{s_1}$ to be the NFA obtained from $\mathcal{N}$ by setting the initial state to be s_1 and a single accepting state s_2. Then, we have $s_1 \xrightarrow{(\alpha,\beta)}_{\mathcal{N}} s_2$ iff $(\alpha, \beta) \in L(\mathcal{N}_{s_2}^{s_1})$.

Thus, $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau$ iff for every $s_1, s_2 \in S$ we have that $\tau_{s_1,s_2} = 1$ iff there exists a word (α, β) with $\mathfrak{P}(\alpha') = \boldsymbol{p}$ and $\mathfrak{P}(\beta') = \boldsymbol{o}$ such that $(\alpha, \beta) \in L(\mathcal{N}_{s_2}^{s_1})$. Equivalently, we have $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau$ iff for every $s_1, s_2 \in S$ it holds that $\tau_{s_1,s_2} = 1$ iff $(\boldsymbol{p}, \boldsymbol{o}) \in \mathfrak{P}(L(\mathcal{N}_{s_2}^{s_1}))$.

By Parikh's theorem, for every $s_1, s_2 \in S$ we can compute a PA formula ψ_{s_1,s_2} such that $(\boldsymbol{p}, \boldsymbol{o}) \models \psi_{s_1,s_2}$ iff $(\boldsymbol{p}, \boldsymbol{o}) \in \mathfrak{P}(L(\mathcal{N}_{s_2}^{s_1}))$. Now we can construct a PA formula Ψ_τ such that $\tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau$ iff $(\boldsymbol{p}, \boldsymbol{o}) \models \Psi_\tau$, as follows:

$$\Psi_\tau := \bigwedge_{s_1,s_2 \,:\, \tau_{s_1,s_2}=1} \psi_{s_1,s_2} \wedge \bigwedge_{s_1,s_2 \,:\, \tau_{s_1,s_2}=0} \neg\psi_{s_1,s_2}.$$

Finally, observe that Ψ_τ defines the set in the premise of the lemma, so we are done. ◀

The redundant product construction. As mentioned in Section 5.1, for the remainder of the proof we want to reason about the types of $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda)$ and $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$ simultaneously. In order to so, we present an auxiliary product construction.

Let $\mathcal{T}_1, \mathcal{T}_2$ be transducers, $\Lambda \subseteq \Sigma_I^*$ be given by an NFA, and let $\mathcal{D}_1 = \mathtt{Tr}(\mathcal{T}_1) \cap \Lambda$ and $\mathcal{D}_2 = \mathtt{Tr}(\mathcal{T}_2)$. We now consider the product automaton of $\mathcal{D}_1$ and $\mathcal{D}_2$, and endow it with two different acceptance conditions, capturing that of $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. Formally, for $i \in \{1, 2\}$, denote $\mathcal{D}_i = \langle \Sigma_I \times \Sigma_O, S_i, s_0^i, \eta_i, F_i \rangle$, then the product automaton is defined as $\mathcal{B}_i = \langle \Sigma_I \times \Sigma_O, S_1 \times S_2, (s_0^1, s_0^2), \eta_1 \times \eta_2, G_i \rangle$, where $G_1 = F_1 \times Q_2$ and $G_2 = Q_1 \times F_2$, and $\eta_1 \times \eta_2$ denotes the standard product transition function, namely $\eta_1 \times \eta_2((s_1, s_2), (\sigma, \sigma')) = (\eta_1(s_1, (\sigma, \sigma')), \eta_2(s_2, (\sigma, \sigma')))$. Thus, $\mathcal{B}_i$ tracks both $\mathcal{D}_1$ and $\mathcal{D}_2$, but has the same acceptance condition as $\mathcal{D}_i$. This seemingly "redundant" product construction has the following important properties, which are crucial for our proof:

▶ **Observation 5.5.** *In the notations above, we have the following:*

1. $L(\mathcal{B}_1) = L(\mathcal{D}_1)$ *and* $L(\mathcal{B}_2) = L(\mathcal{D}_2)$.
2. *For every letter* $(\sigma, \sigma') \in \Sigma_I \times \Sigma_O$, *we have* $\tau_{\mathcal{B}_1}(\sigma, \sigma') = \tau_{\mathcal{B}_2}(\sigma, \sigma')$.

Indeed, Item 1 follows directly from the acceptance condition, and Item 2 is due to the identical transition function of $\mathcal{B}_1$ and $\mathcal{B}_2$.

By Observation 4.1, $L(\mathtt{Perm}_k(\mathcal{D}_i))$ depends only on $L(\mathcal{D}_i)$. We thus have the following.

▶ **Observation 5.6.** *The following holds for every* $k > 0$*:*

1. $L(\mathtt{Perm}_k(\mathcal{B}_1)) = L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda))$.
2. $L(\mathtt{Perm}_k(\mathcal{B}_2)) = L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2)))$.

Type profiles. We now consider the set of types induced by the redundant product automata $\mathcal{B}_1$ and $\mathcal{B}_2$ on Parikh vectors of words of length k. By Item 2 of Observation 5.5, it is enough to consider $\mathcal{B}_1$.

For $k > 0$, we define the *k-th type profile* of $\mathcal{B}_1$ to be the set of all types of Parikh vectors $(\boldsymbol{p}, \boldsymbol{o})$ with $|\boldsymbol{p}| = |\boldsymbol{o}| = k$ that are induced by $\mathcal{B}_1$; i.e. it is the set $\Upsilon(\mathcal{B}_1, k) = \left\{ \tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha), \mathfrak{P}(\beta)) \;\middle|\; (\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k \right\}$. Clearly, there is only a finite number of type profiles, as $\Upsilon(\mathcal{B}_1, k) \subseteq \mathbb{B}^{S' \times S'}$, where S' is the state space of $\mathcal{B}_1$. Therefore, as k increases, after some finite K_0, every type profile that is ever attained will have been encountered already. We now place an upper bound on K_0.

▶ **Lemma 5.7.** *We can effectively compute $K_0 > 0$ such that for every $k > 0$ there exists $k' \leq K_0$ with $\Upsilon(\mathcal{B}_1, k') = \Upsilon(\mathcal{B}_1, k)$.*

Proof. Consider a type τ, and let Ψ_τ be the PA formula constructed as per Lemma 5.4 for the NFA $\mathcal{B}_1$. Observe that for a Parikh vector $(\boldsymbol{p}, \boldsymbol{o})$ and for $k > 0$, the expression $|\boldsymbol{p}| = |\boldsymbol{o}| = k$ is PA definable. Indeed, writing $\boldsymbol{p} = (x_1, \ldots, x_{|\Sigma_I|})$ and $\boldsymbol{q} = (y_1, \ldots, y_{|\Sigma_O|})$, the expression is defined by $x_1 + \ldots + x_{|\Sigma_I|} = k \wedge y_1 + \ldots + y_{|\Sigma_O|} = k$.

Let $T \subseteq \mathbb{B}^{S' \times S'}$ be a set of types (i.e., a potential type profile). We define a PA formula $\Theta_T(z)$ over a single free variable z such that $k \models \Theta_T(z)$ iff $\Upsilon(\mathcal{B}_1, k) = T$, as follows.

$$\Theta_T(z) = \left(\forall \boldsymbol{p}, \boldsymbol{o}, |\boldsymbol{p}| = |\boldsymbol{o}| = z \rightarrow \bigvee_{\tau \in T} \Psi_\tau(\boldsymbol{p}, \boldsymbol{o}) \right) \wedge \left(\bigwedge_{\tau \in T} \exists \boldsymbol{p}, \boldsymbol{o}, |\boldsymbol{p}| = |\boldsymbol{o}| = z \wedge \Psi_\tau(\boldsymbol{p}, \boldsymbol{o}) \right)$$

Intuitively, $\Theta_T(z)$ states that every Parikh vector $(\boldsymbol{p}, \boldsymbol{o})$ with $|\boldsymbol{p}| = |\boldsymbol{o}| = z$ has a type within T, and that all the types in T are attained by some such Parikh vector.

By [FR74; BT76], we can effectively determine for every T whether $\Theta_T(z)$ is satisfiable and, if it is, find a witness M_T such that $M_T \models \Theta_T(z)$. By doing so for every set $T \subseteq \mathbb{B}^{S' \times S'}$, we can set $K_0 = \max \{ M_T \mid \Theta_T(z) \text{ is satisfiable} \}$. Then, for every $k > K_0$ if $\Upsilon(\mathcal{B}_1, k) = T$, then T has already been encountered at $M_T \leq K_0$, as required. ◀

The purpose of the bound K_0 obtained in Lemma 5.7 is to bound the minimal k for which $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, or equivalently $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$ (by Lemma 4.4 and Observation 5.6). This is captured in the following.

▶ **Lemma 5.8.** *Let $k, k' > 0$ such that $k \neq k'$ and $\Upsilon(\mathcal{B}_1, k') = \Upsilon(\mathcal{B}_1, k)$, then we have $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$ iff $L(\mathtt{Perm}_{k'}(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_{k'}(\mathcal{B}_2))$.*

Proof. By the symmetry between k and k', it suffices to prove w.l.o.g. that if $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$, then $L(\mathtt{Perm}_{k'}(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_{k'}(\mathcal{B}_2))$.

Assume the former, and let $w = (x', y') \in L(\mathtt{Perm}_{k'}(\mathcal{B}_1))$, where $(x', y') \in (\Sigma_I^{k'} \times \Sigma_O^{k'})^*$, and we denote $(x', y') = (\alpha'_1, \beta'_1) \cdots (\alpha'_n, \beta'_n)$ with $(\alpha'_j, \beta'_j) \in \Sigma_I^{k'} \times \Sigma_O^{k'}$ for every $1 \leq j \leq n$.

Since $\Upsilon(\mathcal{B}_1, k') = \Upsilon(\mathcal{B}_1, k)$, there is a mapping φ that takes every letter $(\alpha'_j, \beta'_j) \in \Sigma_I^{k'} \times \Sigma_O^{k'}$ in w to a letter $(\alpha_j, \beta_j) \in \Sigma_I^k \times \Sigma_O^k$ that has same type in $\mathtt{Perm}_k(\mathcal{B}_1)$, so that we can find $(x, y) = (\alpha_1, \beta_1) \cdots (\alpha_n, \beta_n)$ such that for every $1 \leq j \leq n$ we have $\tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha_j), \mathfrak{P}(\beta_j)) = \tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha'_j), \mathfrak{P}(\beta'_j))$.

By the definition of the type of a Parikh vector, we have that

$$\tau_{\mathtt{Perm}_k(\mathcal{B}_1)}(\alpha_j, \beta_j) = \tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha_j), \mathfrak{P}(\beta_j)) = \tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha'_j), \mathfrak{P}(\beta'_j)) = \tau_{\mathtt{Perm}_{k'}(\mathcal{B}_1)}(\alpha'_j, \beta'_j).$$

In particular, since the type of a word is the concatenation (i.e., Boolean matrix product) of its underlying letters, we have that $\tau_{\mathtt{Perm}_k(\mathcal{B}_1)}(x,y) = \tau_{\mathtt{Perm}_{k'}(\mathcal{B}_1)}(x',y')$. Since $(x',y') \in L(\mathtt{Perm}_{k'}(\mathcal{B}_1))$, it follows that also $(x,y) \in L(\mathtt{Perm}_k(\mathcal{B}_1))$. Indeed, $(\tau_{\mathtt{Perm}_{k'}(\mathcal{B}_1)}(x',y'))_{s_0^1,s_f^1} = 1$ where s_0^1 and s_f^1 are an initial state and an accepting state of $\mathtt{Perm}_{k'}(\mathcal{B}_1)$, respectively. But the equality of the types implies that $\left(\tau_{\mathtt{Perm}_k(\mathcal{B}_1)}(x,y)\right)_{s_0^1,s_f^1} = 1$ as well, so $\mathtt{Perm}_k(\mathcal{B}_1)$ has an accepting run on (x,y).

By our assumption, $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$, so $(x,y) = \varphi(w) \in L(\mathtt{Perm}_k(\mathcal{B}_2))$, or equivalently, $\varphi(w) \in L(\mathtt{Perm}_k(\mathcal{B}_2))$. We now essentially reverse the arguments above, but with $\mathcal{B}_2$ instead of $\mathcal{B}_1$. However, this needs to be done carefully, so that the mapping of letters lands us back at (x',y'), and not a different word. Thus, instead of finding a round equivalent word, we observe that for every $1 \leq j \leq n$, we also have

$$\tau_{\mathtt{Perm}_k(\mathcal{B}_2)}(\alpha_j,\beta_j) = \tau_{\mathcal{B}_2}(\mathfrak{P}(\alpha_j),\mathfrak{P}(\beta_j)) = \tau_{\mathcal{B}_2}(\mathfrak{P}(\alpha'_j),\mathfrak{P}(\beta'_j)) = \tau_{\mathtt{Perm}_{k'}(\mathcal{B}_2)}(\alpha'_j,\beta'_j),$$

This follows from Item 2 in Observation 5.5 and the fact that the permutation closure depends only on the transitions (and not on accepting states, which are the only difference between $\mathcal{B}_1$ and $\mathcal{B}_2$).

Thus, similarly to the arguments above, we have that $(x',y') \in L(\mathtt{Perm}_{k'}(\mathcal{B}_2))$, and the mapping applied is in fact the the inverse map φ^{-1}, where $\varphi^{-1}(\varphi(w)) = w$. We conclude that $L(\mathtt{Perm}_{k'}(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_{k'}(\mathcal{B}_2))$, as required.

The mapping is illustrated in Figure 5.2. ◀

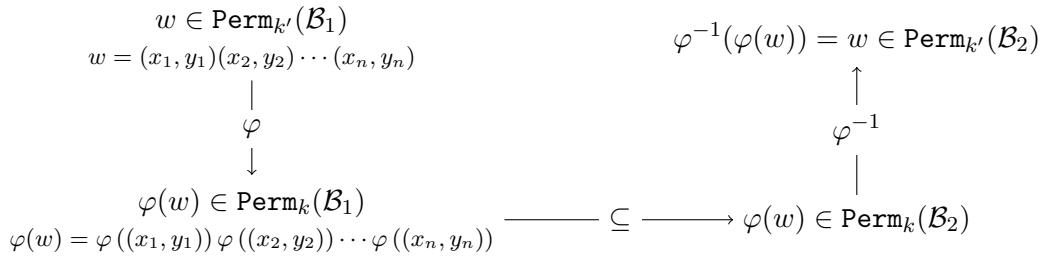

Figure 5.2: A diagram for the proof structure of Lemma 5.8.

Combining Lemmas 5.7 and 5.8, we can effectively compute K_0 such that if it holds that $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$ for some k, then this also holds for some $k < K_0$. Finally, using Lemma 4.4, this concludes the proof of Theorem 5.1. ◀

▶ **Remark 5.9** (Complexity results for Theorem 5.1 and Corollary 5.2)**.** Let n be the number of states in $\mathcal{T}_1 \times \mathcal{T}_2$. Observe that the formula Ψ_τ constructed in Lemma 5.4 comprises a conjunction of $O(n^2)$ PA subformulas, where each subformula is either an existential PA formula of length $O(n)$, or the negation of one. Then, the formula Θ_T in Lemma 5.7 consists of a universal quantification, nesting a disjunction over $|T|$ formulas of the form Ψ_τ, conjuncted with $|T|$ existential quantifications, nesting a single Ψ_τ each. Overall, this amounts to a formula of length $|T| \leq 2^{n^2}$, with alternation depth 3. [1]

[1] Alternation depth is usually counted with the outermost quantifier being existential, which is not the case here, hence 3 instead of 2.

Using quantifier elimination [Coo72; Opp78], we can obtain a witness for the satisfiability of Θ_T of size 4-exponential in n^2. Then, finding the overall bound K_0 amounts to $2^{2^{n^2}}$ calls to find such witnesses. Finally, we need K_0 oracle calls to Lemma 4.4 in order to decide existential simulation, and since K_0 may have a 4-exponential size description, this approach yields a whopping **5-EXP** algorithm. This approach, however, does not exploit any of the structure of Θ_T.

5.3 Lower Bounds for Existential Round Simulation

The complexity bounds in Remark 5.9 are naively analyzed, and we leave it for future work to conduct a more in-depth analysis. In this section, we present lower bounds to delimit the complexity gap. Note that there are two relevant lower bounds: one on the complexity of deciding round simulation, and the other on the minimal value of K_0 in Theorem 5.1.

We start with the complexity lower bound, which applies already for round equivalence.

▶ **Theorem 5.10.** *The problem of deciding, given transducers $\mathcal{T}_1, \mathcal{T}_2$, whether $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$ for* any *k, is* **PSPACE***-hard, even for Λ of a constant size (given as a 5-state DFA).*

Proof sketch. We present a similar reduction to that of Theorem 4.7 from universality of NFAs (see Appendix A.2). In order to account for the unknown value of k, we allow padding words with a fresh symbol $\#$, which is essentially ignored by the transducers. ◀

Next, we show that the minimal value for K_0 can be exponential in the size of the given transducers (in particular, of $\mathcal{T}_2$).

▶ **Example 5.11** (Exponential round length)**.** Let $p_1, p_2, \ldots, p_m$ be the first m prime numbers. We define two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ over input and output alphabet $\mathcal{P} = \{1, \ldots, m\}$, as depicted in Figure 5.3 for $m = 3$. Intuitively, $\mathcal{T}_1$ reads input $w \in \Lambda = (1 \cdot 2 \cdots m)^*$ and simply outputs w, whereas $\mathcal{T}_2$ works by reading a letter $i \in \mathcal{P}$, and then outputting i for p_i steps (while reading p_i arbitrary letters) before getting ready to read a new letter i.

In order for $\mathcal{T}_2$ to k-round simulate $\mathcal{T}_1$, it must be able to output a permutation of $(1 \cdot 2 \cdots m)^*$. In particular, the number of 1's, 2's, etc. must be equal, so k must divide every prime up to p_m, hence it must be exponential in the size of $\mathcal{T}_2$.

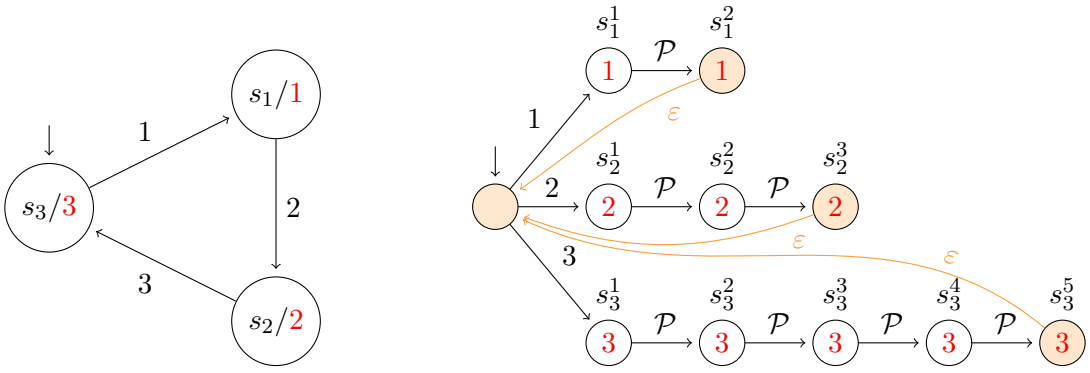

Figure 5.3: The transducers $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (right) for $m = 3$ in Example 5.11. The transition $s \xrightarrow{\varepsilon} t$ in $\mathcal{T}_2$ means that the transition function from state s behaves identically as from t.

The sum of the number of states in $\mathcal{T}_1$ and $\mathcal{T}_2$ is $1 + m + \sum_{i=1}^{m} p_i = \mathsf{O}\left(\sum_{i=1}^{m} p_i\right)$. Set $Q = \prod_{i=1}^{m} p_i$. It is easily verified that $\mathcal{T}_1 \prec_k \mathcal{T}_2$ holds for $k = m \cdot Q$, which is exponential in the number of states. Indeed, for the round $w = (1 \cdots m)^Q$, we consider the permutation $1^Q \cdots m^Q$, on which the run of $\mathcal{T}_2$ induces the same output.

We now show that this k is minimal. For a word $x \in (1 \cdot 2 \cdots m)^*$ in rounds of k to have round equivalent outputs in $\mathcal{T}_1$ and $\mathcal{T}_2$, there must be some word round equivalent word x' in which every appearance of $i \in \mathcal{P}$ is part of a sequence of appearances of i, of length p_i, except maybe at its end. If $m \mid k$, then there are $\frac{k}{m}$ appearances of each i, so $\frac{k}{m}$ must be divisible by all primes, except maybe one. The latter possibility is falsified when considering the next round. If, however, $m \nmid k$, then in the next round, $1 \in \mathcal{P}$ will have one less appearance than in the first round. This, again, makes impossible the round equivalence of the outputs when considering one additional round.

Chapter 6

From Process Symmetry to Round Equivalence

As mentioned in Chapter 1, our original motivation for studying round simulation comes from process symmetry. We present process symmetry with an example before introducing the formal model. Recall the Round Robin (RR) scheduler from Example 3.2. There, at each time step, the scheduler receives as input the IDs of processes in $\mathcal{P} = \{0, 1, 2\}$ that are making a request, and it responds with the IDs of those that are granted (either a singleton $\{i\}$ or $\emptyset$).

In process symmetry, we consider a setting where the identities of the processes may be permuted. This corresponds to the IDs representing, for instance, ports, and the processes not knowing which port they are plugged into. Thus, the input received may be under any permutation of the actual identities of the processes. Note that a permutation in this case is a bijection over identities, not indices as in previous chapters. Then, a transducer is *process symmetric* if the outputs are permuted in a way that matches the permutation of identities. For example, in the RR scheduler, the output corresponding to input $\{1, 2\}\{3\}\{3\}$ is $\{1\}\emptyset\{3\}$. If we permute the identities by swapping 1 and 3, obtaining the input $\{3, 2\}\{1\}\{1\}$, the output letters have to be permuted in the same manner for RR to be process symmetric. However, the output for the input with permuted identities is $\emptyset\emptyset\emptyset$, so RR is not process symmetric.

In [Alm20], several definitions of process symmetry are studied for probabilistic transducers. In the deterministic case, however, process symmetry is a very strict requirement. In order to overcome this, we allow some flexibility by letting the transducer do some local order changes in the word in a way that corresponds to the permutation. This way, for instance, if we are allowed to rearrange (i.e. permute, in the former sense) the input $\{3, 2\}\{1\}\{1\}$ to $\{1\}\{1\}\{3, 2\}$, then the output becomes $\{1\}\emptyset\{3\}$, and once we apply the inverse permutation, this becomes $\{3\}\emptyset\{1\}$. This, in turn, can be again rearranged to obtain the original output $\{1\}\emptyset\{3\}$. In this sense, the scheduler is "locally stable" against permutations of the identities of processes.

We now turn to give the formal model. Consider a set of processes $\mathcal{P} = \{1, \ldots, m\}$ and $k > 0$. For a permutation π of $\mathcal{P}$ (i.e. a bijection $\pi : \mathcal{P} \to \mathcal{P}$) and a letter $\sigma \in 2^{\mathcal{P}}$, we obtain $\pi(\sigma) \in 2^{\mathcal{P}}$ by applying π to each process in σ. We lift this to words $x \in (2^{\mathcal{P}})^*$ by applying the permutation letter-wise to obtain $\pi(x)$. A $2^{\mathcal{P}}/2^{\mathcal{P}}$ transducer $\mathcal{T} = \left\langle 2^{\mathcal{P}}, 2^{\mathcal{P}}, Q, q_0, \delta, \boldsymbol{\ell} \right\rangle$ is *k-round symmetric* if for every permutation π of $\mathcal{P}$ and for every k-round word $x \in (2^{\mathcal{P}})^*$ there exists $x' \in (2^{\mathcal{P}})^*$ such that $\pi(x) \asymp_k x'$ and $\pi(\mathcal{T}(x)) \asymp_k \mathcal{T}(x')$. We say that $\mathcal{T}$ is k-round symmetric

w.r.t. π if the above holds for a certain permutation π.

▶ **Example 6.1.** Consider the RR scheduler for n processes (cf. Example 3.2), and let $\mathcal{T}$ be a transducer for it. When a permutation $\pi \in \mathcal{S}_n$ is applied on the signals, then intuitively, to preserve the behaviour of the system (i.e. the number of grants for each process), we need to change the order of handling the requests of processes (and giving grants) such that it matches the new order of requests. Formally, given input x, for the i-th round $b_1 b_2 \cdots b_n$ of $\pi(x)$ (the input under permutation π) we set the corresponding round in x' to $b_{\pi^{-1}(1)} b_{\pi^{-1}(2)} \cdots b_{\pi^{-1}(n)}$. For example, if $\pi = (0\ 1)$ and $n = 3$, given $x = \{0,2\}\{1\}\{2\}$ we get $\pi(x) = \{1,2\}\{0\}\{2\}$ and choose $x' = \{0\}\{1,2\}\{2\}$. Thus, it holds that $\mathcal{T}(x) = \pi^{-1}(\mathcal{T}(x'))$ or equivalently, $\pi(\mathcal{T}(x)) = \mathcal{T}(x')$, so RR is n-round symmetric.

Example 6.1 shows that RR exhibits round symmetry w.r.t. all permutations. In the general sense, round symmetry might hold w.r.t. some permutations but not others, as is the case in the following.

▶ **Example 6.2.** Set $\mathcal{P} = \{0,1,2\}$ and let $\mathcal{T}$ be the $2^{\mathcal{P}}/2^{\mathcal{P}}$ transducer illustrated in Figure 6.1. It is not difficult to see that $\mathcal{T}$ satisfies 2-round symmetry w.r.t. $\pi = (0\ 1)$ but not w.r.t. some other permutations, e.g. $(0\ 2)$.

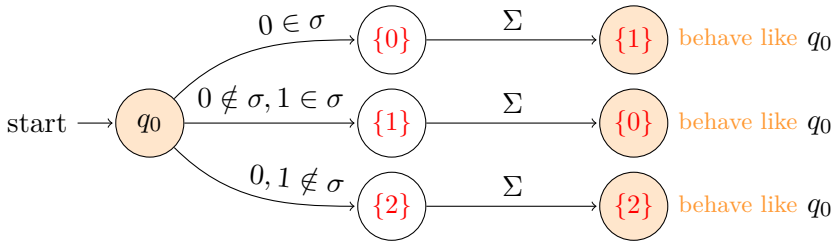

Figure 6.1: Transducer $\mathcal{T}$ satisfying round symmetry w.r.t. $\pi = (0\ 1)$ but not $(0\ 2)$.

In round symmetry, too, we consider two main decision problems: *fixed round symmetry* (where k is fixed) and *existential round symmetry* (where we decide whether there exists $k > 0$ for which this holds). Observe that $\Lambda = (2^{\mathcal{P}})^*$, and is therefore ignored.

From round symmetry to round simulation. In order to solve the decision problems above, we reduce them to the respective problems about round symmetry. We start with the case where the permutation π is given.

Given the transducer $\mathcal{T}$ as above, we obtain from $\mathcal{T}$ a new transducer $\mathcal{T}^\pi$ which is identical to $\mathcal{T}$ except that it acts on a letter $\sigma \in 2^{\mathcal{P}}$ as $\mathcal{T}$ would act on $\pi^{-1}(\sigma)$, and it outputs σ where $\mathcal{T}$ would output $\pi^{-1}(\sigma)$. Figure 6.2 presents the transducer $\mathcal{T}^\pi$ that corresponds to $\mathcal{T}$ of Example 6.2 and $\pi = (0\ 1)$.

Formally, $\mathcal{T}^\pi = \left\langle 2^{\mathcal{P}}, 2^{\mathcal{P}}, Q, q_0, \delta^\pi, \boldsymbol{\ell}^\pi \right\rangle$ where $\delta^\pi(q,\sigma) = \delta(q, \pi^{-1}(\sigma))$ and $\boldsymbol{\ell}^\pi(q) = \pi(\boldsymbol{\ell}(q))$. It is easy to verify that for every $x \in (2^{\mathcal{P}})^*$ we have $\mathcal{T}^\pi(x) = \pi(\mathcal{T}(\pi^{-1}(x)))$. As we now show, once we have $\mathcal{T}^\pi$, round symmetry is equivalent to round simulation, so we can use the tools developed in Chapters 4 and 5 to solve the problems at hand.

▶ **Lemma 6.3.** *For a permutation π and $k > 0$, $\mathcal{T}$ is k-round symmetric w.r.t. π iff $\mathcal{T}^\pi \prec_k \mathcal{T}$.*

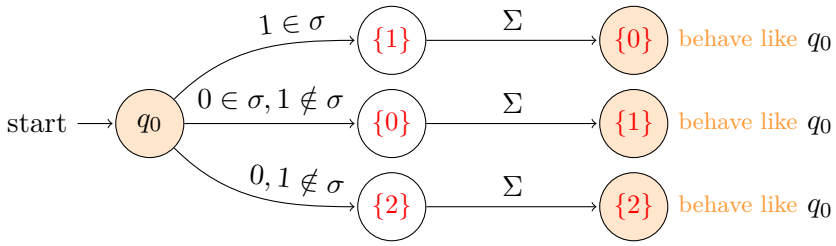

Figure 6.2: Transducer $\mathcal{T}^\pi$ for the $\mathcal{T}$ in Example 6.2 and $\pi = (0\ 1)$.

Proof. By definition, we have that $\mathcal{T}^\pi \prec_k \mathcal{T}$ iff for every $x \in (2^\mathcal{P})^*$ there exists $x' \asymp x$ such that $\mathcal{T}^\pi(x) \asymp \mathcal{T}(x')$. We show that this is equivalent to the definition of round symmetry.

For the first direction, assume $\mathcal{T}$ is k-round symmetric w.r.t. π, and let $x \in (2^\mathcal{P})^*$. Applying the definition of k-round symmetry to $y = \pi^{-1}(x)$ shows that there exists $x' \asymp \pi(y)$ such that $\pi(\mathcal{T}(y)) \asymp \mathcal{T}(x')$. Since $\pi(y) = x$ we get that $x' \asymp x$ and $\pi(\mathcal{T}(\pi^{-1}(x))) \asymp \mathcal{T}(x')$. By the above, $\mathcal{T}^\pi(x) = \pi(\mathcal{T}(\pi^{-1}(x)))$, so we have $\mathcal{T}^\pi(x) \asymp x'$.

For the second direction, assume $\mathcal{T}^\pi \prec_k \mathcal{T}$, and let $x \in (2^\mathcal{P})^*$. Applying the definition of round simulation to $z = \pi(x)$, there exists $x' \asymp z$ such that $\mathcal{T}^\pi(z) \asymp \mathcal{T}(x')$. Thus, $\pi(\mathcal{T}(\pi^{-1}(z))) \asymp \mathcal{T}(x')$, but $\pi^{-1}(z) = x$, so we get $\pi(\mathcal{T}(x)) \asymp \mathcal{T}(x')$, and we are done. ◀

Closure under composition. In order to deal with the general problem of symmetry under all permutations, one could naively check for symmetry against each of the $m!$ permutations. We show, however, that the definition above is closed under composition of permutations.

▶ **Lemma 6.4.** *Consider two permutations π, χ. If $\mathcal{T}^\pi \prec_k \mathcal{T}$ and $\mathcal{T}^\chi \prec_k \mathcal{T}$ then $\mathcal{T}^{\pi\circ\chi} \prec_k \mathcal{T}$.*

Proof. Using the first definition of round symmetry, let $x \in (2^\mathcal{P})^*$, then there exists $x' \asymp_k \pi(x)$ such that $\mathcal{T}(x') \asymp_k \pi(\mathcal{T}(x))$. Moreover, there exists $x'' \asymp_k \chi(x') \asymp_k \chi(\pi(x))$ such that $\mathcal{T}(x'') \asymp_k \chi(\mathcal{T}(x')) \asymp_k \chi(\pi(\mathcal{T}(x)))$, and we are done. ◀

Recall that the group of all permutations of $\mathcal{P}$ is generated by two permutations: the transposition $(1\ 2)$ and the cycle $(1\ 2\ \cdots\ m)$ [Cam+99]. By Lemma 6.4 it is sufficient to check symmetry for these two generators in order to obtain symmetry for every permutation. Note that for the existential variant of the problem, even if every permutation requires a different k, by taking the product of the different values we conclude that there is a uniform k for all permutations. We thus have the following.

▶ **Theorem 6.5.** *Both fixed and existential round symmetry are decidable. Moreover, fixed round symmetry is in* **PSPACE**.

Finally, the reader may notice that our definition of round symmetry w.r.t. π is not symmetric, as was the case with round simulation compared to round equivalence. However, when we consider round symmetry w.r.t. to all permutations, the definition becomes inherently symmetric, as a consequence of Lemma 6.4.

▶ **Lemma 6.6.** *In the notations above, if $\mathcal{T}^\pi \prec_k \mathcal{T}$ then $\mathcal{T} \prec_k \mathcal{T}^\pi$.*

Proof. Recall that for every permutation π we have $\pi^{m!} = \mathtt{id}$, where $\mathtt{id}$ is the identity permutation. In particular, $\pi^{m!-1} = \pi^{-1}$.

By Lemma 6.4, we now have that if $\mathcal{T}^{\pi} \prec_k \mathcal{T}$, then $\mathcal{T}^{\pi^{m!-1}} \prec_k \mathcal{T}$, so $\mathcal{T}^{\pi^{-1}} \prec_k \mathcal{T}$. Applying π to both sides gives us $\mathcal{T} \prec_k \mathcal{T}^{\pi}$. ◀

Thus, for symmetry, the notions of round simulation and round equivalence coincide.

Chapter 7

The Simulation Mapping

The definition of round simulation in Chapter 3 consists of existential statements: it considers the existence of words x' that satisfy the requirement of round simulation. However, our framework lacks an additional feature – we settle for x' to *exist*, but that does not mean we can actually compute it. In some cases, being able to calculate it is beneficial; in other cases, it is necessary.

Computing x' has several benefits. Consider the monitor from Example 1.1. We have modelled it by a transducer $\mathcal{T}_1$ and presented a simpler transducer $\mathcal{T}_2$ that round simulated $\mathcal{T}_1$, which allowed us then to verify a desired property on M against the much smaller $\mathcal{T}_2$: "if there is no `error`, then Process 3 works at least once every 20 steps". The way the verification is performed under simulation is by rewriting the property in a way that exploits the simpler behaviour of $\mathcal{T}_2$. For instance, since we know that $\mathcal{T}_2$ expects to see the requests in the order of the process IDs, then in particular in the case of no `error`, in every round of length 10 the third letter must be $\{3\}$. Then we can rewrite the property in terms of $\mathcal{T}_2$ in the following manner: "if there is no `error`, then for all $n \in \mathbb{N}$, the input must have $\{3\}$ in one of the indices $20n + 3$ and $20n + 13$". An algorithm could then be designed to verify this more specific property. Observe that to rewrite the property we needed some insight on the behaviour of $\mathcal{T}_2$; specifically, we had to know the structure of x' for any input x that satisfies the premise of the property.

We showed an example of a case where we need be able to calculate x' for verification of properties. Generally, being able to compute x' has the additional benefit of *explainability*: it allows us to give a solid reason for the verification result. For instance, if some property does not hold for x, a developer might want to get insight by computing the run of $\mathcal{T}_2$ on x', instead of the possibly much more expensive computation of the run of $\mathcal{T}_1$ on x. It turns out that, in the general sense, calculating x' is not a simple task.

Consider two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $\mathcal{T}_1 \prec_k \mathcal{T}_2$, and an input word $x \in \left(\Sigma_I^k\right)^*$. This means, by definition, that there is a way to permute the rounds in x to obtain a word x' such that $\mathcal{T}_2(x')$ is a permutation of $\mathcal{T}_1(x)$. Consider then a mapping $\psi_{\mathcal{T}_1,\mathcal{T}_2} : x \mapsto x'$ that maps every input word x to a corresponding round equivalent word x' (and we drop the subscript when the transducers are clear from context). We call this a *simulation mapping between* $\mathcal{T}_1$ *and* $\mathcal{T}_2$. In this section we study this mapping and present some properties that it does or does not satisfy. We begin with an example.

▶ **Example 7.1.** Consider the transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ depicted in Figure 7.1, with input and

output alphabets $\Sigma_I = \{a, b\}$ and $\Sigma_O = \{0, 1\}$. $\mathcal{T}_1$ expects to see either ab or ba in every round, outputting 00 in both cases, and otherwise outputs 01 in that round. $\mathcal{T}_2$ expects the first round to be ab and the second to be ba, otherwise outputs 01 in the round not meeting expectations; and beginning from the third round, it behaves like $\mathcal{T}_1$. We have that $\mathcal{T}_1 \prec_2 \mathcal{T}_2$ by a permutation that corrects the order of the letters in the first two rounds of the input. Moreover, we have $\psi(ab) = \psi(ba) = ab$ whereas $\psi(abba) = abba \neq \psi(ab) \cdot \psi(ba)$.

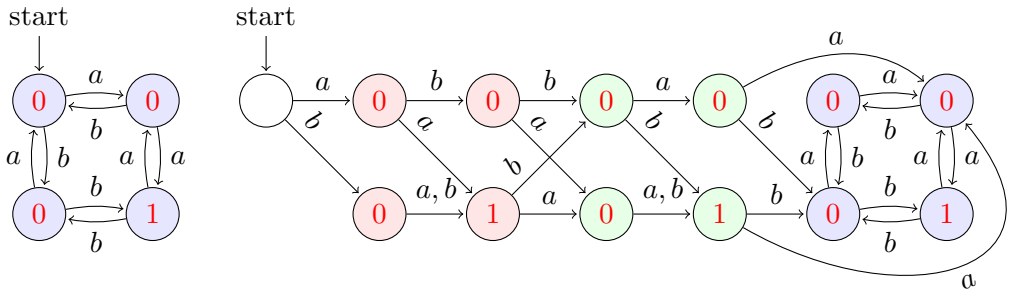

Figure 7.1: The transducers $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (right) in Example 7.1. The states of $\mathcal{T}_2$ in red, green and blue manage the first, second and later rounds, respectively.

Example 7.1 shows that ψ is not a homomorphism: it does not satisfy $\psi(xy) = \psi(x)\psi(y)$. Next, we show that ψ cannot be described by a look-ahead machine, i.e. one that reads a "compound" of rounds in every step.

▶ **Example 7.2.** Set $\Lambda = L[ab \cdot (cc)^* \cdot (ab + ba)]$ and $k = 2$, and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the transducers in Figure 7.2, satisfying $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$. Denote the simulation mapping by $\psi^* : (\Sigma_I^k)^* \to (\Sigma_I^k)^*$.

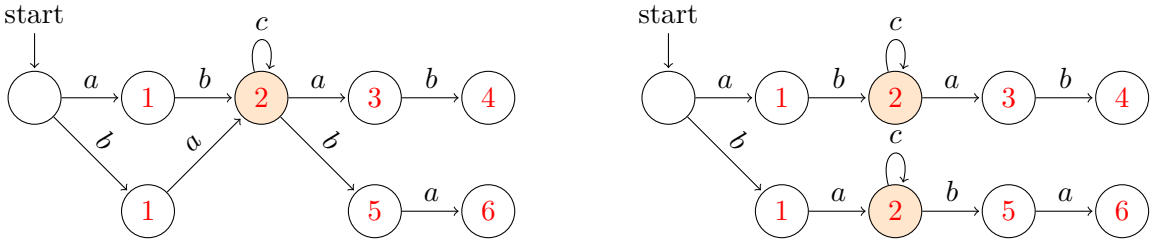

Figure 7.2: The transducers $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (right) in Example 7.2.

We claim that for any r, there is no look-ahead machine that defines a function $\psi_r : (\Sigma_I^{rk})^* \to (\Sigma_I^{rk})^*$ such that $\psi^*(x) = \psi(x)$ for all input words x.

Indeed, let $r \in \mathbb{N}$, and assume by way of contradiction that such ψ_r exists. Now consider the input word $x = ab \cdot c^{rk-2}$. $\psi_r(x)$ must start with either ab or ba. Without loss of generality, assume the former, and consider the input word $x' := x \cdot ba \cdot c^{rk-2}$. Since ψ_r works on r rounds each time, the first r rounds are fixed when it reads the $(r+1)$-th round. Moreover, since $\psi_r(x')$ must induce a valid path in $\mathcal{T}_2$, the only option for the $(r+1)$-th round of $\psi_r(x')$ is ab. Hence, the output of $\mathcal{T}_1$ on x' is different from the output of $\mathcal{T}_2$ on $\psi(x')$, and we have a contradiction.

In Example 7.2, we used the definition of round simulation with restriction; however, it is indeed possible to get rid of the restriction language Λ – it was kept for clarity. Moreover, the

example uses a round length of $k = 3$, but it can be extended to any round length $k > 0$ in a similar manner.

We showed that a look-ahead of r rounds does not suffice for any r. Moreover, a sliding-window variation of the look-ahead model, in which at any given round the machine can read $r - 1$ additional rounds in the future, would not give any additional benefit either; in fact, the same pair of transducers in Example 7.2 show that it is generally impossible to determine the output of the first round without knowing the entire input.

Currently, the best algorithm we have for computing ψ is straight-forward: for an input x, we iterate over all round equivalent words $x' \asymp_k x$ and check for satisfaction of $\mathcal{T}_1(x) \asymp_k \mathcal{T}_2(x')$. Since the length of inputs is unlimited, this algorithm can only be modelled by a Turing machine. The question of whether ψ can be defined by a simpler finite-state model remains open.

Chapter 8

Additional Notions of Symmetry and Simulation

8.1 Variations of Round Symmetry and Round Simulation

Recall from Chapter 2 that y is a permutation of x if their Parikh images are equal: $\mathfrak{P}(x) = \mathfrak{P}(y)$. Furthermore, two words x and y are round equivalent, denoted by $x \asymp_k y$, when every round in y is a permutation of the same round in x. Notice that the rounds need not be permuted in the same manner; the i-th round is permuted by a possibly distinct $\tau_i : [k] \to [k]$ (a permutation of indices). When, however, all permutations coincide, i.e. $\tau_1 = \tau_2 = \cdots$, then we say that x and y are *uniformly round equivalent* and denote by $x \asymp_k^{\mathfrak{u}} y$.

In round symmetry described in Chapter 6, we were given a transducer $\mathcal{T}$ and we checked whether for every input x there exists $x' \asymp_k x$ such that $\pi(\mathcal{T}(x)) \asymp_k \mathcal{T}(x')$. One could require instead that the equivalence between the two pairs of words be uniform. We call the variation of symmetry under this constraint *uniform round symmetry*. Formally, a $2^{\mathcal{P}}/2^{\mathcal{P}}$ transducer $\mathcal{T}$ is *uniformly k-round symmetric* if for every permutation π of $\mathcal{P}$ (a permutation of signals) and input x, there exists $x' \asymp_k^{\mathfrak{u}} x$ such that $\pi(\mathcal{T}(x)) \asymp_k^{\mathfrak{u}} \mathcal{T}(x')$. Note that the uniform permutation between x and x' is not necessarily identical to that between $\pi(\mathcal{T}(x))$ and $\mathcal{T}(x')$.

▶ **Example 8.1** (Round Robin)**.** Consider the RR scheduler for n processes, shown to be n-round symmetric in Example 6.1. Recall that in the proof of its symmetry when the permutation π was applied to the signals, we had to change the order of handling the requests such that it matched the new order of received requests: given input x, for the i-th round $b_1 b_2 \cdots b_n$ of $\pi(x)$ (the input under permutation π) we set the corresponding round in x' to $b_{\pi^{-1}(1)} b_{\pi^{-1}(2)} \cdots b_{\pi^{-1}(n)}$. Since the same permutation π was applied for all rounds of the input x, the permutation by which the rounds of x' were obtained was identical for all rounds. It follows that RR exhibits uniform round symmetry.

Dually, a weaker notion of symmetry than round symmetry is what we call *Parikh round symmetry*: keeping in mind that the letters in process transducers are subsets of process identities in $\mathcal{P}$, a permutation in Parikh round symmetry can not only move letters but also signals, as long as every $i \in \mathcal{P}$ appears the same number of times in the round as originally. For example, if $\mathcal{P} = \{1, 2, 3\}$ then $\Sigma_I = 2^{\mathcal{P}}$ and the round $\{1, 2\}\{3\}\emptyset \in (\Sigma_I)^*$ can be permuted to $\emptyset\{2, 3\}\{1\}$

by moving the signal 2 from the first letter to the second, and the signal 1 to the third. To state this formally, we first need to expand our terminology a step further.

Let $\mathcal{P} = \{1, \ldots, n\}$. For a word $x = x_1 \ldots x_k \in (2^{\mathcal{P}})^k$, define $\#(x, i) = |\{\, j \mid i \in x_j \,\}|$ to be the number of occurrences of i in x. Then, we define the *Parikh image w.r.t.* $\mathcal{P}$ of x as $\mathfrak{P}_{\mathcal{P}}(x) = (\#(x, 1), ..., \#(x, n)) \in \mathbb{N}^n$. If it holds that $\mathfrak{P}_{\mathcal{P}}(x) = \mathfrak{P}_{\mathcal{P}}(y)$, we say that y is a *signal permutation* of x. Furthermore, for words $x, y \in \left((2^{\mathcal{P}})^k\right)^*$, if every round of y is a signal permutation of the same round in x, we say that x and y are *Parikh round equivalent* and write $x \asymp_k^{\mathsf{P}} y$.

Following this, we formally call a $2^{\mathcal{P}}/2^{\mathcal{P}}$ transducer $\mathcal{T}$ *Parikh k-round symmetric* if for every permutation π of $\mathcal{P}$ (a permutation of signals) and input x, there exists $x' \asymp_k^{\mathsf{P}} x$ such that $\pi(\mathcal{T}(x)) \asymp_k^{\mathsf{P}} \mathcal{T}(x')$.

The original notion of round symmetry described in Chapter 6 is hereby called *symbol-wise round symmetry*, reflecting the permutation over symbols (as opposed to signals) between rounds. The original relation of equivalence for words is correspondingly called *symbol-wise round equivalence*.

▶ **Example 8.2** (Parikh symmetry does not imply symbol-wise symmetry)**.** Set $\pi = (0\ 1)$ and let $k \in \mathbb{N}$ and $m \geq 3$. We construct a transducer that is Parikh k-round symmetric, but not symbol-wise k'-round symmetric for any k'.

Consider the $2^{\mathcal{P}}/2^{\mathcal{P}}$ transducer $\mathcal{T} = \left\langle 2^{\mathcal{P}}, 2^{\mathcal{P}}, S, s_0, \delta, \boldsymbol{\ell} \right\rangle$ depicted in Figure 8.1, where $\mathcal{P} = [m] = \{0, \cdots, m-1\}$.

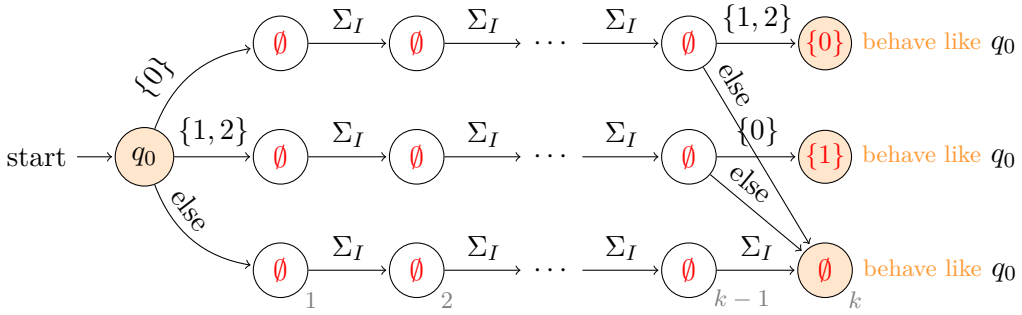

Figure 8.1: $\mathcal{T}$ exhibits Parikh, but not symbol-wise, round symmetry (see Example 8.2).

Observe that every round starts at q_0. There are three possible forms for the output of each round depending on the input, as summarized in Table 8.1.

Table 8.1: The inputs and their corresponding outputs in $\mathcal{T}$ of Example 8.2.

Input	Output
$\{0\}\sigma_2 \cdots \sigma_{k-1}\{1,2\}$	$\emptyset^{k-1}\{0\}$
$\{1,2\}\sigma_2 \cdots \sigma_{k-1}\{0\}$	$\emptyset^{k-1}\{1\}$
else	$\emptyset^k$

We first show that $\mathcal{T}$ is Parikh round symmetric. Let x be an input word and π a permutation of $\mathcal{P}$. Like in Chapter 6, $\pi(x)$ is the word obtained from x by permuting every signal according

to π. If x is of one of the first two forms in Table 8.1, then by moving the signal $2 \in \mathcal{P}$ (fixed in π) between the first and last letters, we get $x' \asymp^{\mathtt{p}} \pi(x)$ such that $T(x') \asymp^{\mathtt{p}} \pi(T(x))$, as desired. Now assume x is of some other form, having the output $\emptyset^k$. If $2 \in \mathcal{P}$ appears in both the first and last letters, or it appears in neither, then set $x' = \pi(x)$; otherwise, move the signal 2 to the other letter, and the output will remain $\emptyset^k$. Thus, $\mathcal{T}$ is Parikh round symmetric.

On the other hand, $\mathcal{T}$ is not symbol-wise k'-round symmetric for any $k' > 0$. To see this, take the input $x = \{0\}^{k-1} \cdot \{1,2\} \cdot \emptyset^{k'k-k}$. We have $|x| = k'k$ which is divisible by k', $\mathcal{T}(x) = \emptyset^{k-1} \cdot \{0\} \cdot \emptyset^{k'k-k}$. It holds that $\pi(x) = \{1\}^{k-1} \cdot \{0,2\} \cdot \emptyset^{k'k-k}$, which contains neither the letter $\{0\}$ nor $\{1,2\}$. Thus, regardless of how we permute $\pi(x)$ to obtain x', the output of any $x' \asymp^{\mathtt{s}} \pi(x)$ is always $\emptyset^{k'k}$, which is not a permutation of $\mathcal{T}(x)$.

Three types of round equivalence for words have been presented in total, and each of them was used to define a variation of round symmetry: Parikh, symbol-wise and uniform. Collectively, we call them the *modes of permutation* and define $\mathbf{MOP} = \{\mathtt{p}, \mathtt{s}, \mathtt{u}\}$, where $\mathtt{p}$, $\mathtt{s}$ and $\mathtt{u}$ stand for Parikh, symbol-wise and uniform. In the remainder of this section, we extend the definitions to round simulation and consider how these three modes relate to each other.

Extension to round simulation. Similarly to symmetry, round simulation can also be extended to variations of its original notion described in Chapter 3. As an example, consider RR for three processes. Let $\mathcal{T}_0$ and $\mathcal{T}_1$ be two copies of RR with different initial states (cf. Example 3.2): $\mathcal{T}_0$ first considers requests from signal 0, whereas $\mathcal{T}_1$ from signal 1. Example 3.2 established that $\mathcal{T}_0 \prec_k \mathcal{T}_1$. In fact, the permutation of indices $\tau = (0\ 1\ 2)$ is the only permutation used in the simulation: by permuting the rounds of x according to τ, one obtains x' such that all rounds of $\mathcal{T}_1(x')$ are obtained from those of $\mathcal{T}_0(x)$ by applying τ. In other words, for every input x, there exists $x' \asymp_k^{\mathtt{u}} x$ such that $\mathcal{T}_0(x) \asymp_k^{\mathtt{u}} \mathcal{T}_1(x')$. It follows that a notion of uniformity in round simulation is exhibited; that is, the uniform mode of permutation $\mathtt{u} \in \mathbf{MOP}$ can be used to measure equivalence of words in round simulation, just as it has been for round symmetry.

Formally, we say a transducer $\mathcal{T}_1$ is $\langle \mathtt{u}, \mathtt{u}, k \rangle$*-round simulated by* $\mathcal{T}_2$ if for every input x there exists $x' \asymp_k^{\mathtt{u}} x$ such that $\mathcal{T}_1(x) \asymp_k^{\mathtt{u}} \mathcal{T}_2(x')$ (and the reason behind the double appearance of $\mathtt{u}$ will be clear in what follows).

As we would expect, round simulation can similarly be extended for the remaining mode of permutation, $\mathtt{p} \in \mathbf{MOP}$. However, we can also measure the equivalence of the input and the output words according to different symmetry notions, thereby combining two symmetry notions. For this end, we say a transducer $\mathcal{T}_1$ is $\langle \eta, \eta', k \rangle$*-round simulated by* $\mathcal{T}_2$ if for any input word x, there exists $x' \asymp_k^{\eta} x$ such that $\mathcal{T}_2(x') \asymp_k^{\eta'} \mathcal{T}_1(x)$. When this holds between $\mathcal{T}_1$ and $\mathcal{T}_2$, we denote this by $\mathcal{T}_1 \prec_k^{\eta,\eta'} \mathcal{T}_2$ (for simplicity, we do not consider restriction languages in this section).

We go a step further and define a partial order on the set of all types of round simulation according to this definition, i.e. the set $\mathbf{MOP}_k^2 := \{ \langle \eta, \eta', k \rangle \mid \eta, \eta' \in \mathbf{MOP} \}$ for a fixed $k > 0$. The meaning of the order between two types of simulation is aimed to be implication in the following sense: if $\langle \eta, \eta', k \rangle \leq \langle \mu, \mu', k \rangle$, then $\mathcal{T}_1 \prec_k^{\mu,\mu'} \mathcal{T}_2$ implies $\mathcal{T}_1 \prec_k^{\eta,\eta'} \mathcal{T}_2$. Before defining the order on $\mathbf{MOP}_k^2$, we begin with defining an order on $\mathbf{MOP}$ as such: $\mathtt{p} \leq \mathtt{s} \leq \mathtt{u}$. Here, too, the meaning is implication, as established by the following lemma.

▶ **Lemma 8.3.** *Let x, y be words over Σ. For any $k > 0$ and $\eta, \mu \in \mathbf{MOP}$ such that $\eta \leq \mu$, if $x \asymp_k^{\mu} y$ then $x \asymp_k^{\eta} y$.*

Intuitively, this is because if uniform equivalence holds between words, then in particular, symbol-wise equivalence holds too by a simple observation of the definitions; and if symbol-wise equivalence holds between x and y, this means y is a permutation and, in particular, a signal permutation of x.

Following this, we can now define the order on $\mathbf{MOP}_k^2$ to be the *product order* of two copies of $\mathbf{MOP}$: $\langle \eta, \eta', k\rangle \leq \langle \mu, \mu', k\rangle$ if both $\eta \leq \mu$ and $\eta' \leq \mu'$. In fact, $\mathbf{MOP}$ defines a lattice, and $\mathbf{MOP}_k^2$ (upon fixing k and ignoring the third coordinate) is the lattice obtained from the product of two copies of $\mathbf{MOP}$. It is not difficult to see from the definition that the following holds too.

▶ **Lemma 8.4.** *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be transducers. For any $\eta, \eta', \mu, \mu' \in \mathbf{MOP}$ such that $\langle \eta, \eta', k\rangle \leq \langle \mu, \mu', k\rangle$, if $\mathcal{T}_1 \prec_k^{\mu,\mu'} \mathcal{T}_2$ then $\mathcal{T}_1 \prec_k^{\eta,\eta'} \mathcal{T}_2$.*

We furthermore show that these implications are strict.

▶ **Example 8.5.** Recall the transducer $\mathcal{T}$ from Example 8.2, and consider the transducer $\mathcal{T}^{\pi}$ obtained from $\mathcal{T}$ by permuting both the input and the output by π as in Chapter 6. We have shown that $\mathcal{T}$ is Parikh round symmetric. By a reasoning analogous to the transition from symmetry to simulation as per Chapter 6, this gives $\mathcal{T} \prec_k^{\mathtt{p},\mathtt{p}} \mathcal{T}^{\pi}$. However, it does not hold that $\mathcal{T} \prec_k^{\mathtt{s},\mathtt{p}} \mathcal{T}^{\pi}$: for the input $x := \{0\}\sigma_2 \cdots \sigma_{k-1}\{1,2\}$ having output $y := \emptyset^{k-1}\{0\}$ (cf. Table 8.1), any permutation $x' \asymp_k^{\mathtt{s}} x$ will lead to an output of $\emptyset^k \not\asymp_k^{\mathtt{p}} y$. Thus $\mathcal{T} \not\prec_k^{\mathtt{s},\mathtt{p}} \mathcal{T}^{\pi}$ (and in particular, $\mathcal{T} \not\prec_k^{\mathtt{s},\mathtt{s}} \mathcal{T}^{\pi}$ so $\mathcal{T}$ is not symbol-wise symmetric). In the general sense, we conclude that $\mathcal{T}_1 \prec_k^{\mathtt{p},\mathtt{p}} \mathcal{T}_2$ does not imply $\mathcal{T}_1 \prec_k^{\mathtt{s},\mathtt{p}} \mathcal{T}_2$.

Example 8.5 establishes the gap[1] $\langle \mathtt{p}, \mathtt{p}, k\rangle \lneq \langle \mathtt{s}, \mathtt{p}, k\rangle$, illustrated in Figure 8.2. In order to establish the gap $\langle \mathtt{p}, \mathtt{p}, k\rangle \lneq \langle \mathtt{p}, \mathtt{s}, k\rangle$, we use a different pair of transducers. As for the first gap, we use a process-symmetric approach: we define one transducer $\mathcal{T}$ and choose $\mathcal{T}_1$ and $\mathcal{T}_2$ to be $\mathcal{T}$ and $\mathcal{T}^{\pi}$.

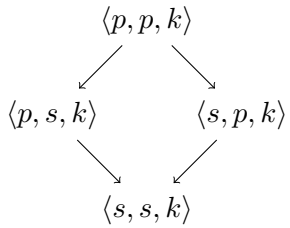

Figure 8.2: A Hasse diagram for a subset of the partial order on $\mathbf{MOP}_k^2$ ($\alpha \to \beta$ implies $\alpha \leq \beta$). We show that all ordered pairs are strict.

▶ **Example 8.6.** Consider the transducer $\mathcal{T}$ in Figure 8.3, whose round-by-round behaviour can once more be summarized in a table (see Table 8.2). $\mathcal{T}$ is Parikh round symmetric: for an input

[1] Inequality clearly holds between the two tuples. However, we use the notation of equality (and strict inequality) between elements in $\mathbf{MOP}_k^2$ to mean the implication of round simulation (or lack of it) between these types, as in Lemma 8.4.

x, choose $x' = \pi(x)$. It is not difficult to show that $\mathcal{T}(x') \asymp_k^{\mathtt{p}} \pi(\mathcal{T}(x))$ by considering the possible forms of x according to Table 8.2. To see that $\mathcal{T} \not\asymp_k^{p,s} \mathcal{T}^\pi$, consider the word $x = \{0\}\emptyset\emptyset$. The output of $\mathcal{T}$ on x is $\{0\}\emptyset\{2\}$. Any round equivalent word x' of x either starts with $\{1\}$ or $\emptyset$, the respective outputs being either $\{1,2\}\emptyset\emptyset$ or $\emptyset^3$. In all cases, we have $T(x') \not\asymp_k^{\mathtt{s}} \mathcal{T}^\pi(x)$.

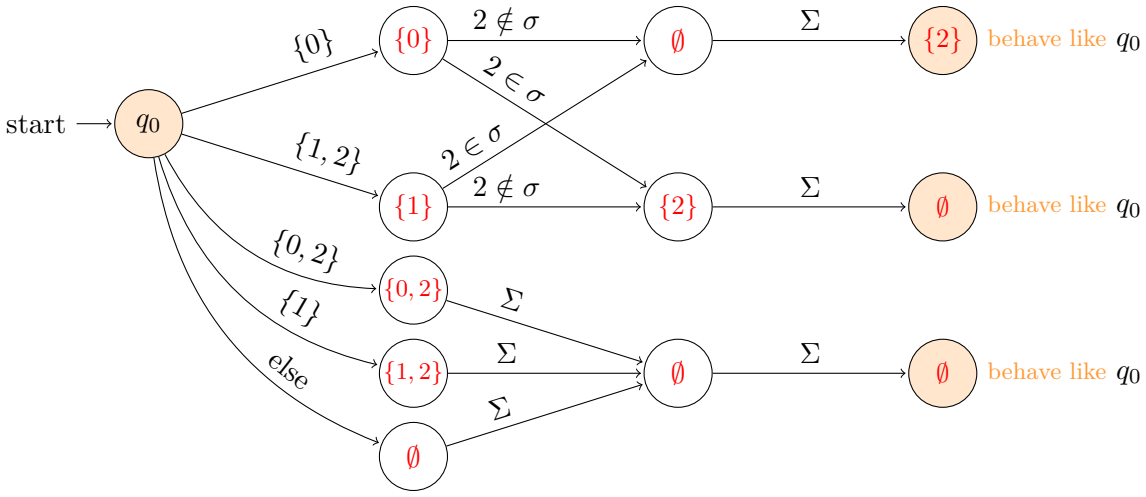

Figure 8.3: The transducer $\mathcal{T}$ for Example 8.6. The transitions $i \in \sigma$ and $i \notin \sigma$ mean all letters from Σ_I that, respectively, contain or do not contain i.

Table 8.2: The inputs and their corresponding outputs in $\mathcal{T}$ of Example 8.6.

Input	Output
$\{0\}(2 \notin \sigma)\,\sigma$	$\{0\}\emptyset\{2\}$
$\{0\}(2 \in \sigma)\,\sigma$	$\{0\}\{2\}\emptyset$
$\{1,2\}(2 \in \sigma)\,\sigma$	$\{1\}\emptyset\{2\}$
$\{1,2\}(2 \notin \sigma)\,\sigma$	$\{1\}\{2\}\emptyset$
$\{0,2\}\sigma\sigma$	$\{0,2\}\emptyset\emptyset$
$\{1\}\sigma\sigma$	$\{1,2\}\emptyset\emptyset$
else	$\emptyset\emptyset\emptyset$

The transducers used in Examples 8.5 and 8.6 have established two gaps from Figure 8.2. In fact, these same transducers can be used to establish the remaining two gaps as well, as follows. The transducer $\mathcal{T}$ in Example 8.5 satisfies $\langle \mathtt{p}, \mathtt{s}, k\rangle$-round simulation with its corresponding $\mathcal{T}^\pi$; indeed, observe that the output labels are either singleton sets or empty sets, so that a signal permutation of the output is equivalent to permuting the letters. The transducer $\mathcal{T}$ in Example 8.6 satisfies $\langle \mathtt{s}, \mathtt{p}, k\rangle$-round simulation with its corresponding $\mathcal{T}^\pi$, which is inferred from the choice of $x' = \pi(x)$, satisfying in particular $x' \asymp^{\mathtt{s}} \pi(x)$. However, neither of the two satisfy $\langle \mathtt{s}, \mathtt{s}, k\rangle$-round simulation, since they are not symbol-wise round symmetric. This finishes the proof of strictness of the gaps illustrated in Figure 8.2.

The full Hasse diagram of the partial order on $\mathbf{MOP}_k^2$ is illustrated in Figure 8.4. We believe that elements in the same row are not comparable and, as in the sub-diagram in Figure 8.2,

all implications are strict. We conclude our contribution for this section by presenting some transducers that aid us in the proof of strictness, all being variants of RR:

1. RR that expects all requests in the beginning of every round, but outputs like the original (e.g. $\{0,2\}\{1\}\{1\}$ would output $\{0\}\emptyset\{2\}$), modelled by $\mathcal{T}_1$.

2. RR that expects input as in the original, but outputs all grants in the end of the round (e.g. $\{0,2\}\{1\}\{1\}$ would output $\emptyset\emptyset\{0,1\}$), modelled by $\mathcal{T}_2$.

3. RR such that every other round begins by considering requests of Process 1 before Process 0 (e.g. $\{0\}\{1\}\emptyset \cdot \{0\}\{1\}\emptyset$ would output $\{0\}\emptyset\emptyset \cdot \emptyset\{1\}\emptyset$), modelled by $\mathcal{T}_3$.

Denote by $\mathcal{T}$ the transducer for RR. It is not difficult to see that $\mathcal{T}_1 \prec^{\mathtt{p},\mathtt{u}} \mathcal{T}$ but $\mathcal{T}_1 \not\prec^{\mathtt{s},\mathtt{u}} \mathcal{T}$; that $\mathcal{T}_2 \prec^{\mathtt{u},\mathtt{p}} \mathcal{T}$ but $\mathcal{T}_2 \not\prec^{\mathtt{u},\mathtt{s}} \mathcal{T}$; and that $\mathcal{T}_3 \prec^{\mathtt{s},\mathtt{s}} \mathcal{T}$ but $\mathcal{T}_3 \not\prec^{\mathtt{s},\mathtt{u}} \mathcal{T}$ and $\mathcal{T}_3 \not\prec^{\mathtt{u},\mathtt{s}} \mathcal{T}$.

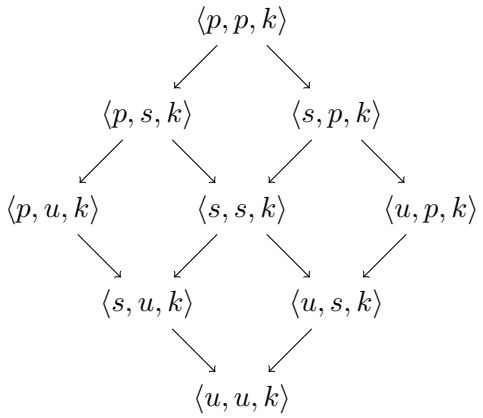

Figure 8.4: A complete Hasse diagram for the partial order on $\mathbf{MOP}_k^2$ ($\alpha \to \beta$ implies $\alpha \leq \beta$).

Finally, Example B.1 presents a pair of transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $\mathcal{T}_1 \prec_2^{\mathtt{s},\mathtt{p}} \mathcal{T}_2$ and $\mathcal{T}_1 \prec_2^{\mathtt{p},\mathtt{s}} \mathcal{T}_2$, but $\mathcal{T}_1 \not\prec_2^{\mathtt{s},\mathtt{s}} \mathcal{T}_2$. This proves that although $\langle \mathtt{s}, \mathtt{s}, k\rangle$-round simulation implies both $\langle \mathtt{p}, \mathtt{s}, k\rangle$ and $\langle \mathtt{s}, \mathtt{p}, k\rangle$-round simulation, the inverse does not hold.

8.2 Symmetry over Infinite Words

So far we have dealt with finite words. However, the setting of infinite words is common in formal verification; it arises naturally in ongoing processes, e.g., elevator controllers, operating systems, etc.

For modelling systems over infinite words, the same model of transducers could be used. Indeed, recall that according to our definition of a transducer in Chapter 2, the output is a word obtained by concatenating labels of the states. This definition extends seamlessly for infinite words, where for an infinite input $x \in (\Sigma_I)^\omega$ the output is also infinite, $\mathcal{T}(x) \in (\Sigma_O)^\omega$.

Consider therefore a $2^I/2^O$ transducer over infinite words $\mathcal{T} = \left\langle 2^I, 2^O, Q, q_0, \delta, \boldsymbol{\ell} \right\rangle$ with input and output signals $I = \{i_1, \ldots, i_k\}$ and $O = \{o_1, \ldots, o_k\}$. We say that $\mathcal{T}$ is *ultimately symmetric* if for every permutation $\pi \in \mathcal{S}_k$ and for every $x \in (2^I)^\omega$ there exists $k \geq 0$ such that $\mathcal{T}(\pi(x))[k : \infty] = \pi(\mathcal{T}(x))[k : \infty]$. That is, for every word x, apart from some finite prefix, the output of $\mathcal{T}$ on $\pi(x)$ is identical to the permuted output $\pi(\mathcal{T}(x))$. We say that $\mathcal{T}$ is ultimately symmetric

w.r.t. π if the above holds for a certain permutation π. The main result of this section is the following.

▶ **Theorem 8.7.** *The problem of deciding whether a transducer $\mathcal{T}$ is ultimately symmetric w.r.t. π can be solved in polynomial time.*

To prove this, we first need to define an additional tool.

A *deterministic co-Büchi automaton over words (DCW)* is a tuple $\mathcal{C} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ where Σ, Q, q_0 and δ are defined just as in an NFA (c.f. Chapter 2), $\alpha \subseteq Q$ and a run r of an *infinite* word $w \in \Sigma^\omega$ is *accepting* if the states appearing infinitely many times in r are elements in α; i.e. $\inf(r) \subseteq \alpha$. If the run of $\mathcal{C}$ on a word w is accepting, we say that $\mathcal{C}$ *accepts* w, and all words accepted by $\mathcal{C}$ comprise the *language* of $\mathcal{C}$, denoted by $L(\mathcal{C})$.

The condition on an accepting run in a DCW is only one among several *acceptance conditions* that could be chosen for automata over infinite words. We refer the reader to [Bok18] for a detailed survey of the most common types and the motivation for introducing them.

Armed with the definition of DCW, we are now ready to prove the theorem.

Proof of Theorem 8.7. Let $\mathcal{T} = \left\langle 2^I, 2^O, Q, q_0, \delta, \boldsymbol{\ell} \right\rangle$. We obtain from $\mathcal{T}$ and π a DCW $\mathcal{C}_{\mathcal{T},\pi} = \left\langle Q \times Q, 2^I, (q_0, q_0), \mu, \alpha \right\rangle$ as follows. Intuitively, $\mathcal{C}_{\mathcal{T},\pi}$ simulates two copies of $\mathcal{T}$ where the second copy is permuted by π, i.e. when seeing input $\sigma \in 2^I$ it simulates the transition of $\mathcal{T}$ with $\pi(\sigma)$. Then, each state (q, r) is marked as accepting if the permuted labelling of q is the same as the labelling of r. We then show that $\mathcal{C}$ accepts a word $x \in (2^I)^\omega$ iff there exists $k \geq 0$ such that $\mathcal{T}(\pi(x))[k : \infty] = \pi(\mathcal{T}(x))[k : \infty]$, so all that remains is to decide whether $L(\mathcal{C}) = (2^I)^\omega$, which can be done in polynomial time.

Formally, we define the components of $\mathcal{C}_{\mathcal{T},\pi}$ as such: $\alpha = \{ (s, t) \mid \pi(\boldsymbol{\ell}(s)) = \boldsymbol{\ell}(t) \}$ and $\mu((s, t), I') = (\delta(s, I'), \delta(t, \pi(I')))$. Observe that for an input $x \in (2^I)^\omega$, we have that $\mathcal{C}(x) = (\mathcal{T}(x), \mathcal{T}(\pi(x)))$. Denote by $r_{\mathcal{C},x}$ the run of $\mathcal{C}$ on x. Then, it holds that $x \in L(\mathcal{C})$ iff $\inf(r_{\mathcal{C},x}) \subseteq \alpha$, iff at some point all states in $r_{\mathcal{C},x}$ are in α; i.e. iff there exists $k > 0$ such that $r_{\mathcal{C},x}[k : \infty] \in \alpha^\star$. But this is equivalent to saying there exists $k > 0$ such that $\pi(\mathcal{T}(x))[k : \infty] = \mathcal{T}(\pi(x))[k : \infty]$. The required result follows. ◀

It is not difficult to show that ultimate symmetry, like round symmetry, is closed under composition of permutations: if $\mathcal{T}$ is ultimately symmetric w.r.t. permutations π and χ then it is also ultimately symmetric w.r.t. $\pi \circ \chi$. Again relying on the fact that the group $\mathcal{S}_k$ of all permutations is generated by two permutations [Cam$^+$99], it follows that the problem of deciding ultimate symmetry is in **P**.

Chapter 9

Conclusion and Open Questions

In this work, we introduced round simulation and provided decision procedures and lower bounds (some with remaining gaps) for the related algorithmic problems.

Round simulation, and in particular its application to round symmetry, is only an instantiation of a more general framework of symmetry, by which we measure the stability of transducers under local changes to the input. In particular, there is place for additional notions of symmetry and simulation to be studied, and the existing ones extended. Some such variants were presented and discussed in Section 8.1. An additional possibly interesting notion of simulation is *window simulation*, where we use a sliding window of size k instead of disjoint k-rounds as in round simulation. In addition, the setting of infinite words is of interest. Beside the notion of *ultimate simulation* presented in Section 8.2, a possible future direction is to define an analogous symmetry for the probabilistic setting, where state transitions are equipped with probabilities and every input leads to some probability distribution over the state space [Alm20]. Finally, other types of transducers may also require variants of simulation, such as streaming-string transducers [Alu10].

Beside extending the study of the different notions of symmetry, a few gaps have remained open along the way of this study. In terms of complexity bounds for the existential case, can we do better than the **5-EXP** bound in Remark 5.9? Furthermore, regarding the simulation mapping from Chapter 7 that maps every input word to a corresponding word for the simulating transducer, can we find a sub-Turing model that defines it? Here, too, other models might help. For this end, a possible direction is looking into streaming-string transducers and bimachines [MP19].

Appendix A

PSPACE Hardness

▶ **Lemma A.1.** *Universality of NFAs over alphabet $\Sigma = \{0, 1\}$, where all states are accepting, and the degree of nondeterminism is at most 2, is* **PSPACE***-complete.*

Proof. In [KRS09], it is shown that universality of NFAs remains **PSPACE**-complete even for NFAs over alphabet $\Sigma = \{0, 1\}$ and all states accepting. Thus, we only need to show that this remains the case under the restriction that $|\delta(q, \sigma)| \leq 2$ for every state q and letter σ.

To see this, we start by observing that universality remains **PSPACE**-complete for NFAs over alphabet $\{0, 1, \$\}$ with nondeterminism degree at most 2. Indeed, given an NFA over $\{0, 1\}$ with maximal nondeterminism degree $d > 2$, we can replace each transition of the form[1] $\delta(q, \sigma) = \{q_1, \ldots, q_d\}$ with a binary tree of depth $\lceil \log d \rceil$, reading \$ on all transitions, which starts at q and ends in $q_1, \ldots, q_d$. Thus, we introduce at most d states for every transition. By marking these states as accepting, this reduction maintains universality, and requires a polynomial blowup.

Next, we observe that the reductions in [KRS09, Lemma 2] first transform an NFA over alphabet size k to an NFA over alphabet size $k + 1$ with all states accepting and with identical nondeterminism degree (indeed, the only added transitions are in fact deterministic), and then transforms an NFA with all states accepting and alphabet size 4 to an NFA with all states accepting and alphabet size 2, with an equal nondeterminism degree (essentially by encoding each of the 4 letters as two letters in $\{0, 1\}$).

Since we start this chain of reductions with an NFA of nondeterminism degree at most 2, we maintain this property throughout the proof. ◀

A.1 Proof of Theorem 4.7

We show a reduction from the universality problem for NFAs over alphabet $\{0, 1\}$ where all states are accepting and the degree of nondeterminism is at most 2, to round-equivalence with $k = 2$ and with Λ given as a DFA of constant size. The former is shown to be **PSPACE**-hard in Lemma A.1.

Consider an NFA $\mathcal{N} = \langle Q, \{0, 1\}, \delta, q_0, Q \rangle$ where $|\delta(q, \sigma)| \leq 2$ for every $q \in Q$ and $\sigma \in \{0, 1\}$. We construct two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ over input and output alphabets $\Sigma_I = \{a, b, c, d\}$ and

[1] We can assume all transitions have degree exactly d by adding redundant transitions

$\Sigma_O = \{\top, \bot\}$ and $\Lambda \subseteq \Sigma_I^*$, such that $L(\mathcal{N}) = \{0,1\}^*$ iff $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$.

Set $\Lambda = (ab + cd)^*$ (described as a 4-state DFA). Intuitively, our reduction encodes $\{0,1\}$ into $\{a,b,c,d\}^2$ by setting 0 to correspond to ab and to ba, and 1 to cd and to dc. Then, $\mathcal{T}_1$ keeps outputting $\top$ for all inputs in Λ, thus mimicking "accepting" every word in $\{0,1\}^*$. We then construct $\mathcal{T}_2$ so that every nondeterministic transition of $\mathcal{N}$ on e.g., 0 is replaced by two deterministic branches on ab and on ba. Hence, when we are allowed to permute ab and ba by round equivalence, we capture the nondeterminism of $\mathcal{N}$.

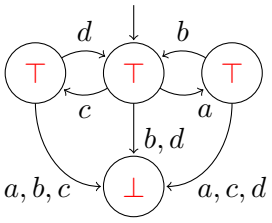

Figure A.1: The transducer $\mathcal{T}_1$ in the proof of Theorem 4.7.

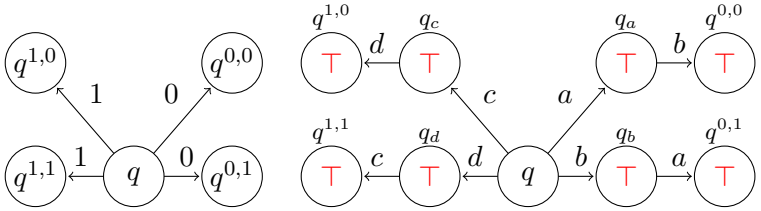

Figure A.2: Every state and its 4 transitions in $\mathcal{N}$ (left) turn into 8 transitions in $\mathcal{T}_2$ (right). All transitions not drawn in the right figure lead to $q_\bot$, a sink state labelled $\bot$.

We now proceed to define the reduction formally. We construct $\mathcal{T}_1$ independently of $\mathcal{N}$, as depicted in Figure A.1, containing 4 states. For every $x \in \Lambda$ we have $\mathcal{T}_1(x) = \top^{|x|}$, and for every other $x \notin \Lambda$ we have $\mathcal{T}_1(x) = \top^m \bot^{|x|-m}$ where m is the length of the maximal prefix of x in $(ab + cd)^*(a + c + \epsilon)$.

We proceed to construct $\mathcal{T}_2$. We can think of the outgoing transitions from every state q as $\delta(q,0) = \{q^{0,0}, q^{0,1}\}$ and $\delta(q,1) = \{q^{1,0}, q^{1,1}\}$ (unless $\mathcal{N}$ has no outgoing transitions on one of the letters, see below). We obtain $\mathcal{T}_2$ from $\mathcal{N}$ by introducing 4 new states q_a, q_b, q_c, q_d for every state $q \in Q$, and setting the transitions and labels as depicted in Figure A.2. In case $\mathcal{N}$ does not have a transition on e.g., 0 from q, then instead of going to q_a or q_b, we proceed to a new state $q_\bot$ labelled $\bot$, which is a sink state. In addition, $q_\bot$ is reached upon any transition not yet defined. Observe that for every $x \in \Lambda$ we have $\mathcal{T}_2(x) = \top^m \bot^{|x|-m}$ for some $0 \leq m \leq |x|$ (since $q_\bot$ is a sink).

We now claim that $L(\mathcal{N}) = \{0,1\}^*$ iff $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$. For the first direction, assume $L(\mathcal{N}) = \{0,1\}^*$. Observe that $\mathcal{T}_2 \prec_{2,\Lambda} \mathcal{T}_1$ independently: for every $x \in (ab + cd)^*$, denote $\mathcal{T}_2(x) = \top^m \bot^{|x|-m}$, then we can construct $x' \asymp_2 x$ such that $\mathcal{T}_1(x') = \top^m \bot^{|x|-m}$ by leaving x unchanged m steps, and then permuting the letters such that the run of $\mathcal{T}_1$ moves to the sink labelled $\bot$ (indeed, observe that m must be even by the construction of $\mathcal{T}_2$, and hence $\mathcal{T}_1$ can permute e.g., ab to ba in order to start outputting $\bot$ on an even step).

Next, we show that $\mathcal{T}_1 \prec_{2,\Lambda} \mathcal{T}_2$. Consider $x \in (ab + cd)^*$, so that $\mathcal{T}_1(x) = \top^{|x|}$, and let $w \in \{0,1\}^*$ be the word obtained from x by identifying ab with 0 and cd with 1. Since $L(\mathcal{N}) = \{0,1\}^*$, there exists a run (and hence an accepting run) of $\mathcal{N}$ on w, denoted $s_0, s_1, \ldots, s_n$. We now obtain $x'' \asymp_2 x$ by identifying each letter 0 in x with either ab or ba, and each letter 1 with cd or dc, such that the run of $\mathcal{T}_2$ on x'' simulates the run of $\mathcal{N}$ on w. Thus, $\mathcal{T}_2(x'') = \top^{|x''|}$, and $\mathcal{T}_2(x'') \asymp_2 \mathcal{T}_1(x)$, so we are done.

Conversely, if $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$, then in particular $\mathcal{T}_1 \prec_{2,\Lambda} \mathcal{T}_2$. We claim that $L(\mathcal{N}) = \{0,1\}^*$. Consider $w \in \{0,1\}^*$. Dually to the above, we obtain from w a word $x \in (ab + cd)^*$ by

identifying 0 with ab and 1 with cd, so that $\mathcal{T}_1(x) = \top^{|x|}$. Since $\mathcal{T}_1 \prec_{2,\Lambda} \mathcal{T}_2$, there exists $x' \asymp_2 x$ such that $\mathcal{T}_2(x') = \top^{|x|}$. Observe that x' must be obtained from x by (possibly) changing each ab to ba and each cd to dc. In particular, the run of $\mathcal{T}_2$ on x' induces a run of $\mathcal{N}$ on w by identifying both ab and ba as 0 and both cd and dc as 1. This gives $w \in L(\mathcal{N})$, so $L(\mathcal{N}) = \{0,1\}^*$, which concludes the proof. ◀

A.2 Proof of Theorem 5.10

In order to show that existential round equivalence is **PSPACE**-hard, we build upon the reduction in the proof of Theorem 4.7: we again show a reduction from the universality problem for NFAs over alphabet $\{0,1\}$ where all states are accepting and the degree of nondeterminism is at most 2 (cf. Lemma A.1).

Consider an NFA $\mathcal{N} = \langle Q, \{0,1\}, \delta, q_0, Q\rangle$ where $|\delta(q,\sigma)| \leq 2$ for every $q \in Q$ and $\sigma \in \{0,1\}$. We construct two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ over input and output alphabets $\Sigma_I = \{a,b,c,d,\#\}$ and $\Sigma_O = \{\top,\bot\}$ and $\Lambda \subseteq \Sigma_I^*$, such that $L(\mathcal{N}) = \{0,1\}^*$ iff $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$.

Intuitively, the idea is to use a similar encoding of $\{0,1\}$ in $\{a,b,c,d\}$ whereby 0 corresponds to either ab or ba and 1 to cd or dc. Now, however, since k is not fixed to 2, we also allow arbitrary padding with sequences of $\#\#$.

Set $\Lambda = (ab + cd + \#\#)^*$ (given as a 5 state DFA). We construct $\mathcal{T}_1$ and $\mathcal{T}_2$ similarly to the proof of Theorem 4.7, by adding self-cycles of length 2 upon reading $\#\#$, from every state except the sink $q_\bot$. See Figures A.3 and A.4 for an illustration.

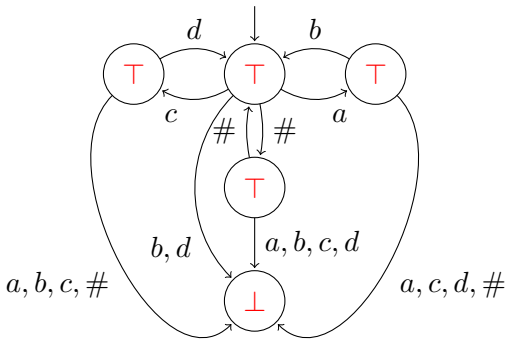

Figure A.3: The transducer $\mathcal{T}_1$ in the proof of Theorem 5.10.

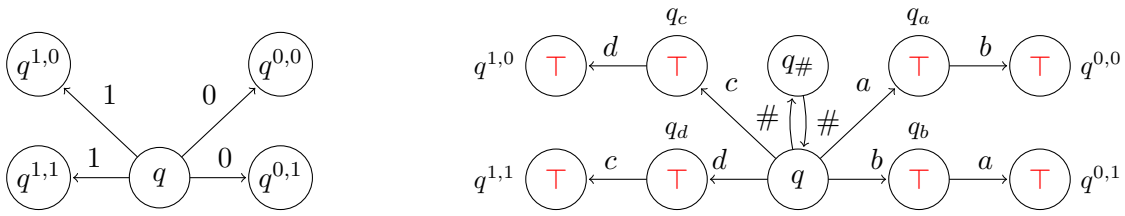

Figure A.4: Every state and its 4 transitions in $\mathcal{N}$ (left) turn into 10 transitions in $\mathcal{T}_2$ (right). All transitions not drawn in the right figure lead to $q_\bot$, a sink state labelled $\bot$.

We claim that $L(\mathcal{N}) = \{0,1\}^*$ iff there exists $k > 0$ such that $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$. For the first

direction, assume $L(\mathcal{N}) = \{0,1\}^*$, then we can show that $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$ by following the proof of Theorem 4.7 line for line, with the addition that blocks of the form ## leave the state of both $\mathcal{T}_1$ and $\mathcal{T}_2$ unchanged.

For the converse direction, assume $\mathcal{T}_1 \equiv_{k,\Lambda} \mathcal{T}_2$, and in fact we only assume $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ for some $k > 0$. We further assume w.l.o.g. that k is even, otherwise we can just take $2k$ (since we also have $\mathcal{T}_1 \prec_{2k,\Lambda} \mathcal{T}_2$).

Consider $w \in \{0,1\}^*$. We obtain from w a word $x \in (ab + cd + \#\#)^*$ by identifying 0 with $ab\#^{k-2}$ and 1 with $cd\#^{k-2}$. Observe that $\mathcal{T}_1(x) = \top^{|x|}$, and that x is indeed a k-round word in Λ, with each round being either $ab\#^{k-2}$ or $cd\#^{k-2}$.

Since $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$, there exists $x' \asymp_k x$ such that $\mathcal{T}_2(x') = \top^{|x|}$. Observe that x' must be obtained from x by (possibly) changing each ab to ba and each cd to dc, and by shifting the location of this pair within the # symbols. Indeed, otherwise the run of $\mathcal{T}_2$ on x' ends in $q_\perp$. In particular, the run of $\mathcal{T}_2$ on x' induces a run of $\mathcal{N}$ on w by identifying both ab and ba as 0 and both cd and dc as 1. Thus, $w \in L(\mathcal{N})$, so $L(\mathcal{N}) = \{0,1\}^*$, and the proof is concluded. ◀

Appendix B

Variations of Round Simulation

▶ **Example B.1.** The transducers in Figure B.1 satisfy $\mathcal{T}_1 \prec_2^{\mathtt{s},\mathtt{p}} \mathcal{T}_2$ and $\mathcal{T}_1 \prec_2^{\mathtt{p},\mathtt{s}} \mathcal{T}_2$. This is proven in Table B.1, which considers all possible forms of each round and gives round equivalent words $x^{\mathtt{s}} \asymp_2^{\mathtt{s}} x$ and $x^{\mathtt{p}} \asymp_2^{\mathtt{p}} x$ that satisfy the requirements of the definitions.

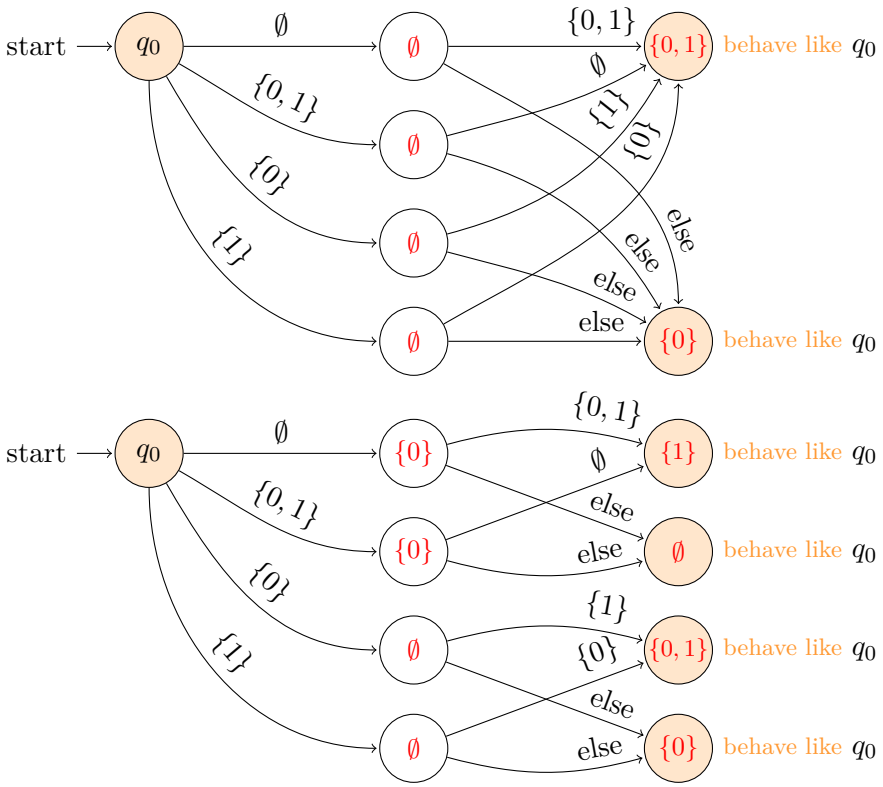

Figure B.1: Transducers $\mathcal{T}_1$ (up) and $\mathcal{T}_2$ (down) in Example B.1, satisfying $\mathcal{T}_1 \prec_2^{\mathtt{s},\mathtt{p}} \mathcal{T}_2$ and $\mathcal{T}_1 \prec_2^{\mathtt{p},\mathtt{s}} \mathcal{T}_2$, but $\mathcal{T}_1 \not\prec_2^{\mathtt{s},\mathtt{s}} \mathcal{T}_2$. See Table B.1 for a table summarizing the possible inputs and outputs for $\mathcal{T}_1$.

However, $\mathcal{T}_1 \not\prec_{k'}^{\mathtt{s},\mathtt{s}} \mathcal{T}_2$ for any $k' > 0$. Indeed, consider the word $x = \{0,1\}\emptyset^{k'-1}$ having output $\mathcal{T}(x) = \emptyset\{0,1\}\emptyset^{k'-2}$. For $\mathcal{T}_2$ to output the letter $\{0,1\}$, it must see one of the input letters $\{0\}$ and $\{1\}$, since the only state labelled $\{0,1\}$ has two incoming transitions with $\{0\}$ and $\{1\}$. But any $x' \asymp_{k'}^{\mathtt{s}} x$ will not contain the letters $\{0\}$ and $\{1\}$, so $\mathcal{T}_1(x) \not\asymp_{k'}^{\mathtt{s}} \mathcal{T}_2(x')$. Therefore $\mathcal{T}_1 \not\prec_{k'}^{\mathtt{s},\mathtt{s}} \mathcal{T}_2$.

Table B.1: A table summarizing the outputs of transducer $\mathcal{T}_1$ in Example B.1 on words x of length 2, and round equivalent words $x^{\mathtt{s}}$ and $x^{\mathtt{p}}$ that satisfy the requirement of x' in the definition of $\mathcal{T}_1 \prec_2^{\mathtt{s},\mathtt{p}} \mathcal{T}_2$ and $\mathcal{T}_1 \prec_2^{\mathtt{p},\mathtt{s}} \mathcal{T}_2$.

x	$\mathcal{T}_1(x)$	$x^{\mathtt{s}}$: $\mathcal{T}_2(x^{\mathtt{s}}) \succ_2^{\mathtt{p}} \mathcal{T}_1(x)$	$\mathcal{T}_2(x^{\mathtt{s}})$	$x^{\mathtt{p}}$: $\mathcal{T}_2(x^{\mathtt{p}}) \succ_2^{\mathtt{s}} \mathcal{T}_1(x)$	$\mathcal{T}_2(x^{\mathtt{p}})$
$\emptyset\emptyset$	$\emptyset\{0\}$	$\emptyset\emptyset$	$\{0\}\emptyset$	$\emptyset\emptyset$	$\{0\}\emptyset$
$\emptyset\{0\}$	$\emptyset\{0\}$	$\emptyset\{0\}$	$\{0\}\emptyset$	$\emptyset\{0\}$	$\{0\}\emptyset$
$\emptyset\{1\}$	$\emptyset\{0\}$	$\emptyset\{1\}$	$\{0\}\emptyset$	$\emptyset\{1\}$	$\{0\}\emptyset$
$\emptyset\{0,1\}$	$\emptyset\{0,1\}$	$\emptyset\{0,1\}$	$\{0\}\{1\}$	$\{0\}\{1\}$	$\emptyset\{0,1\}$
$\{0\}\emptyset$	$\emptyset\{0\}$	$\{0\}\emptyset$	$\emptyset\{0\}$	$\{0\}\emptyset$	$\emptyset\{0\}$
$\{0\}\{0\}$	$\emptyset\{0\}$	$\{0\}\{0\}$	$\emptyset\{0\}$	$\{0\}\{0\}$	$\emptyset\{0\}$
$\{0\}\{1\}$	$\emptyset\{0,1\}$	$\{0\}\{1\}$	$\emptyset\{0,1\}$	$\{0\}\{1\}$	$\emptyset\{0,1\}$
$\{0\}\{0,1\}$	$\emptyset\{0\}$	$\{0\}\{0,1\}$	$\emptyset\{0\}$	$\{0\}\{0,1\}$	$\emptyset\{0\}$
$\{1\}\emptyset$	$\emptyset\{0\}$	$\{1\}\emptyset$	$\emptyset\{0\}$	$\{1\}\emptyset$	$\emptyset\{0\}$
$\{1\}\{0\}$	$\emptyset\{0,1\}$	$\{1\}\{0\}$	$\emptyset\{0,1\}$	$\{1\}\{0\}$	$\emptyset\{0,1\}$
$\{1\}\{1\}$	$\emptyset\{0\}$	$\{1\}\{1\}$	$\emptyset\{0\}$	$\{1\}\{1\}$	$\emptyset\{0\}$
$\{1\}\{0,1\}$	$\emptyset\{0\}$	$\{1\}\{0,1\}$	$\emptyset\{0\}$	$\{1\}\{0,1\}$	$\emptyset\{0\}$
$\{0,1\}\emptyset$	$\emptyset\{0,1\}$	$\{0,1\}\emptyset$	$\{0\}\{1\}$	$\{0\}\{1\}$	$\emptyset\{0,1\}$
$\{0,1\}\{0\}$	$\emptyset\{0\}$	$\{0,1\}\{0\}$	$\{0\}\emptyset$	$\{0,1\}\{0\}$	$\{0\}\emptyset$
$\{0,1\}\{1\}$	$\emptyset\{0\}$	$\{0,1\}\{1\}$	$\{0\}\emptyset$	$\{0,1\}\{1\}$	$\{0\}\emptyset$
$\{0,1\}\{0,1\}$	$\emptyset\{0\}$	$\{0,1\}\{0,1\}$	$\{0\}\emptyset$	$\{0,1\}\{0,1\}$	$\{0\}\emptyset$

Bibliography

[Alm20] S. Almagor. Process symmetry in probabilistic transducers. In *40th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS*, 2020.

[Alu10] R. Alur. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, 2010.

[Bok18] Udi Boker. Why these automata types? In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 57 of *EPiC Series in Computing*, pages 143–163. EasyChair, 2018. URL: `https://easychair.org/publications/paper/G5dD`.

[BS73] J. A. Brzozowski and I. Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, 1973.

[BT76] I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, 1976.

[Cam+99] P. J. Cameron et al. *Permutation groups*, volume 45. Cambridge University Press, 1999.

[CEFJ96] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal methods in system design*, 9(1-2):77–104, 1996.

[CHVB18] E.M. Clarke, T.A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

[Coo72] D. C Cooper. Theorem proving in arithmetic without multiplication. *Machine intelligence*, 7(91-99):300, 1972.

[ES96] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal methods in system design*, 9(1-2):105–131, 1996.

[FPS15] H. Fernau, M. Paramasivan, and M. L. Schmid. Jumping finite automata: characterizations and complexity. In *International Conference on Implementation and Application of Automata*, pages 89–101. Springer, 2015.

[FR74] M.J. Fischer and M.O. Rabin. *Super-exponential Complexity of Presburger Arithmetic*. Project MAC: MAC technical memorandum. Massachusetts Institute of Technology Project MAC, 1974, pages 27–41. URL: https://books.google.co.il/books?id=ijoNHAAACAAJ.

[Haa18] C. Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. URL: https://dl.acm.org/citation.cfm?id=3242964.

[HKR97] T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *Proc. 8th Conference on Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, Warsaw. Springer-Verlag, July 1997.

[Hof20] S. Hoffmann. State complexity bounds for the commutative closure of group languages. In *International Conference on Descriptional Complexity of Formal Systems*, pages 64–77. Springer, 2020.

[HW87] M. P. Herlihy and J. M. Wing. Axioms for concurrent objects. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 13–26, 1987.

[ID96] C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal methods in system design*, 9(1-2):41–75, 1996.

[KRS09] Jui-Yi Kao, Narad Rampersad, and Jeffrey Shallit. On NFAs where all states are final, initial, or both. *Theoretical Computer Science*, 410(47-49):5010–5021, 2009.

[LNRS16] A. W. Lin, T. K. Nguyen, P. Rümmer, and J. Sun. Regular symmetry patterns. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 455–475. Springer, 2016.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd Int. Joint Conf. on Artificial Intelligence*, pages 481–489. British Computer Society, 1971.

[MP19] Anca Muscholl and Gabriele Puppis. The Many Facets of String Transducers (Invited Talk). In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 2:1–2:21, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10241.

[MZ12] A. Meduna and P. Zemek. Jumping finite automata. *International Journal of Foundations of Computer Science*, 23(07):1555–1578, 2012.

[Opp78] D. C. Oppen. A 222pn upper bound on the complexity of presburger arithmetic. *Journal of Computer and System Sciences*, 16(3):323–332, 1978.

[Par66] R. J. Parikh. On context-free languages. *J. of the ACM*, 13(4):570–581, 1966.

[VSS05] K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational horn clauses. In *International Conference on Automated Deduction*, pages 337–352. Springer, 2005.

של מילים אינסופיות, איפה שהגדרנו סימולציה אולטימטיבית. לבסוף, סוגים אחרים של משרנים עשויים לדרוש גרסאות שונות של סימולציה, כגון משרנים הסתברותיים.

מלבד הרחבת החקר במושגים השונים של סימטריה, נותרו כמה פערים פתוחים לאורך מחקר זה. פער אחד ניכר הינו הניתוח הנאיבי של הסיבוכיות והחסם התחתון לאלגוריתמים שהוצגו. יתרה מזאת, הצגנו את מיפוי הסימולציה שממפה כל מילת קלט למילה מתאימה למשרן המסמלץ, ונשאלת השאלה האם נוכל למצוא מודל תת-טיורינג שמגדיר את המיפוי.

אנו משתמשים בכלים וטכניקות מעולם הלוגיקה, האלגברה, תורת המספרים ותורת האוטומטים. כלים נבחרים כוללים חשבון פרסבורגר (Presburger arithmetic) מלוגיקה, תכונות של מספרים ראשוניים מאלגברה ומשפט פריך מתורת המספרים.

גם כאן אנו משתמשים בכלים. הראשון הוא הטיפוס של אותיות באוטומטים. טיפוס של אות נותן תיאור של האופן שבו מצב המכונה משתנה כאשר אות זו נקראת. בהתחשב בסוגי האותיות באוטומט סגור-תמורות, אנו עוברים לדבר על קבוצת כל הטיפוסים שנצפו באוטומט, שאנו מכנים ב- "פרופיל הטיפוס" של האוטומט.

אנו מעוניינים בפרופילי הטיפוס של אוטומטי סגור-תמורות עבור אורכי סיבוב הולכים וגדלים. ישנו מספר סופי של אפשרויות לפרופילי טיפוס, עד כדי ריבוי. עוד אנו טוענים שיש אורך סיבוב כלשהו, שלאחריו לא מופיעים פרופילי טיפוס חדשים. כלומר, בהגעתנו אליו, כבר נראה את כל פרופילי הטיפוס שאנו הולכים לראות. זה לוקח אותנו לכלי הבא: אנו מוכיחים את המשפט הזה על ידי תרגום שלו לבעיה בחשבון פרסבורגר, ופותרים אותו הודות למחקר בלוגיקה.

אורך סיבוב זה נותן לנו בעצם את המטרה שלנו, על ידי התבוננות בתוצאה הסופית הזו: אם פרופילי הטיפוס שווים לאורכי סיבוב k ו- k', ואם זה המקרה גם עבור $\mathcal{T}_1$ ו- $\mathcal{T}_2$, אז ההכלה של השפות של אוטומטי סגור-תמורות של $\mathcal{T}_1$ ושל $\mathcal{T}_2$ (כלומר הבעיה של סימולציה סיבובית באורך סיבוב קבוע) מתקיימת עבור אורך סיבוב k אם ורק אם היא מתקיימת עבור k'. כלומר, הבעיה של סימולציה סיבובית עם אורכי סיבוב k ו- k' הינה חופפת לשניהם, בכל פעם שפרופילי הטיפוס שלהם שווים. זה מסיים את ההוכחה, עד להסתייגות קטנה שטופלה היטב בעבודה.

לאורך העבודה, אנו נתייחס במיוחד למימוש פשוט של מערכת התזמון בשם Round Robin, אשר ידועה והינה בשימוש אף במערכת ההפעלה של המחשב. מערכת זו מפגינה סימטריה בסיבובים באופן ניכר, ולכן היא משמשת דוגמה מאלפת להדגמת ההגדרות ואף למתן אינטואיציה בפתירת הבעיות השונות.

סוגים נוספים של סימטריה במשרנים

גם כן מוצגים ונדונים בעבודה זו מספר מושגים נוספים, הן וריאציות של סימטריה בסיבובים והן סימטריות בסביבות אחרות. אחת הווריאציות של סימטריה בסיבובים מכונה סימטריה בסיבובים ע"פ פריך (Parikh). תחת וריאציה זו, כל אות קלט מהווה תת קבוצה של אותות – או מזהים לתהליכים השונים – כך שכל תהליך מזוהה עם אחד האותות, וכאשר מבצעים תמורה, מותר לנו לא רק להזיז את האותיות בכל סיבוב, אלא גם לערבב את האותות הבודדים בין האותיות בסיבוב.

כאמור, אנו גם בוחנים סימטריות בסביבות שונות ובמכונות חישוב אחרות, משרנים בעלי קלט אינסופי. עבודה עם מילים אינסופיות הינה נפוצה באימות פורמלי; זה נובע מהרעיון שלעיתים מצופה ממערכת לקרוא קלט ללא הפסקה, וסימטריה במערכות כאלה יכולה ללבוש אחת משתי צורות: היא תתפרש על פני כל הקלט שנקרא כבר, או שהיא תרחיב אל העתיד. סימטריה אולטימטיבית לוכדת סוג של סימטריה שמשיגה את המקרה האחרון. בסימטריה אולטימטיבית, עבור כל מילת קלט x, הפלט של $\mathcal{T}$ על $\pi(x)$ זהה לתמורה של הפלט על הקלט המקורי $\pi(\mathcal{T}(x))$, מלבד רישא סופית כלשהי.

הצעות לעבודת המשך

חשוב לציין כי סימולציה בסיבובים, ובמיוחד היישום שלה על סימטריה בסיבובים, הינה רק דוגמה של מסגרת כללית יותר של סימטריה, שבאמצעותה אנו מודדים את היציבות של משרנים תחת שינויים מקומיים בקלט. בפרט, יש מקום לחקור מושגים נוספים של סימטריה וסימולציה ולהרחיב את הקיימים. כמה הרחבות כאלה הוצגו ונדונו בעבודה. כיוון אפשרי הוא הרעיון של סימולציה בחלונות, שבה אנו משתמשים בחלון הזזה באורך k במקום סיבובים זרים, וכיוון נוסף הוא סימטריה בסיבובים ע"פ פריך המתוארת מעלה. בנוסף, יש עניין לסביבה

משרן $\mathcal{T}_2$ מסמלץ משרן $\mathcal{T}_1$ בסיבוביס של k אם עבור כל מילת קלט x, נוכל לערבב את האותיות בכל סיבוב ב-x, כך שהפלט של $\mathcal{T}_2$ על המילה לאחר הערבוב היא עצמה ערבוב של הפלט של $\mathcal{T}_1$ על x. למעשה, אנו נותנים הגדרה קצת יותר משוכללת, בכך שאנו מאפשרים גם לקלט ל-$\mathcal{T}_1$ להיות מוגבל לשפה רגולרית נתונה Λ.

בהתאם, בעיות ההכרעה הנוצרות מההגדרה של סימולציה בסיבובים הן האם $\mathcal{T}_2$ מסמלץ את $\mathcal{T}_1$ בסיבובים של k, (1) כאשר k ניתן כקלט, ו-(2) עבור k בכמת קיומי (קרי, עבור k כלשהו).

המיפוי בין מילות הקלט עבור $\mathcal{T}_1$ לבין הקלט המעורבב עבור $\mathcal{T}_2$ נקרא מיפוי הסימולציה והוא נדון בהמשך עבודה זו.

סימטריה בסיבובים וסימולציה בסיבובים – בעיות ההחלטה ופתרונותיהן, הגבול העליון והתחתון ומיפוי הסימולציה – הם התרומה העיקרית של עבודה זו. בנוסף, מוצגים ונדונים עוד כמה מושגים של סימטריה, כולל וריאציות של סימטריה עגולה וסימטריה בהגדרה של אינסוף מילים.

תקציר לפתרון של בעיות ההכרעה

נגדיר תחילה את המודל העיקרי שאיתו אנו עובדים. משרן (transducer) הינו מכונת מצבים הדומה לאוטומט סופי דטרמיניסטי, פרט לכך שלכל מצב מוצמדת אות פלט, וריצה של משרן על מילת קלט w מחזירה פלט באותו אורך, לפי המצבים שהמילה w עברה בהם. בניגוד לאוטומטים, אין משמעות לקבלה של מילים במשרן.

כאמור, אנו פותרים את בעיות ההכרעה של סימולציה בסיבובים, ומכאן גם נקבל פתרון עבור בעיית הסימטריה בסיבובים.

הבעיה הראשונה היא המקרה הקל מבין שניהם. אנו מתחילים לפתור אותה בהגדרת כלי שהוא אוטומט סופי אי-דטרמיניסטי שאנו מכנים סגור-תמורות (Permutation Closure).

אוטומט סגור-תמורות מתקבל מהמשרן $\mathcal{T}$ ואורך הסיבוב k. בו, כל רצף של k מעברים במשרן המקורי נותן מעבר יחיד באוטומט החדש. יתר על כן, זהו מעבר שאינו מודע לסדר האותיות. זה נותן שהאוטומט למעשה עונה על התכונה שלפיה בחרנו את שמו: סגירות תחת תמורות, משמע שכל התמורות של אותו זוג מילות קלט ופלט מתנהגות באופן זהה זו לזו.

כדי לבנות אוטומט זה, השתמשנו באוטומט ביניים דטרמיניסטי בשם Trace שבעצם מוציא את אותיות הפלט מהמצבים לקשתות הנכנסות.

בהישען על כלי זה, אנו מגיעים ללמה האומרת כי עבור משרנים נתונים $\mathcal{T}_1$, $\mathcal{T}_2$ ואורך סיבוב k, סימולציה בסיבובים מתקיימת בין $\mathcal{T}_1$ ו-$\mathcal{T}_2$ אם ורק אם יש הכלה בין השפות של אוטומטי סגור-תמורות של $\mathcal{T}_1$ ושל $\mathcal{T}_2$.

אכן, אינטואיטיבית, אם נקבל איזשהו קלט עבור $\mathcal{T}_1$ והפלט המתאים לו, ההכלה הזו אומרת שיש תמורה כלשהי של הקלט כך שהפלט הוא עצמו גם תמורה של הפלט הנתון.

מכיוון שצמצמנו את הבעיה לבעיית ההכלה של אוטומטים אי-דטרמיניסטיים, תוצאה ישירה היא שהבעיה היא במחלקת הסיבוכיות **PSPACE**. למעשה, אנו גם מראים שהבעיה הינה **PSPACE**-שלמה.

כעת, במקרה הקיומי, אנו שואלים אם יש איזה אורך סיבוב שעבורו מתקיימת סימולציה סיבובית. אנו עושים זאת על ידי מציאת חסם עליון לאורך הסיבוב המינימלי שעבורו מתקיימת סימולציה סיבובית. זה נותן לנו חסם עליון סופי לקבוצת האורכים שאנו צריכים לבדוק מולם, מה שהופך את הבעיה לניתנת להכרעה.

תקציר

בדיקת מודלים היא פרדיגמת אימות למערכות, בה אנו פועלים להכרעה האם מערכת עומדת במפרט נתון. לעתים קרובות, מערכות מרובות תהליכים מפגינות סוג כלשהו של סימטריה במבנה שלהן או בהתנהגות שלהן. סימטריה מתבטאת בדרך כלל גם במפרטים אשר מערכות מרובות תהליכים נבדקות נגדם. כאשר קיימת סימטריה במערכת או במפרט, היא יכולה להיות מנוצלת על ידי המעצב ואלגוריתם האימות כדי להקל חלק מהמורכבות של בדיקת מודלים, כמו גם כדי לקבל תובנה לגבי התנהגות המערכת. לדוגמה, מערכות סימטריות מאפשרות למעצב להשתמש רק במפרטים מייצגים שבהם היה צורך בעבר באיטרציה על זהויות התהליכים. לפיכך, אנו מעוניינים להחליט אם מערכת נתונה מפגינה סימטריה.

סימטריה אינה מושג מוגדר היטב ועשויה לבוא בצורות שונות, שכל אחת מהן תופסת התנהגות אופיינית אחרת. בעבודה זו, אנו מתמקדים בסימטריית תהליכים, בה לכל תהליך j יש אות קלט ופלט מתאימים i_j ו- o_j, ואלפבית הקלט והפלט של המודל הם 2^I ו- 2^O בהתאמה. סימטריית תהליכים מתייחסת לתרחיש שבו זהויות התהליכים עשויות להיות מעורבבות – כלומר, עברו התמרה. לדוגמה, אם הקלט $\{i_1, i_2\}$ נוצר, המערכת עשויה לקבל למעשה קלט $\{i_7, i_4\}$. לאחר מכן, מערכת מפגינה סימטריית תהליכים אם, באופן אינטואיטיבי, הפלטים שלה עוברים התמרה בצורה דומה לקלטים. לרוע המזל, מערכות דטרמיניסטיות שמפגינות סימטריית תהליכים הן תמימות ביותר, שכן סימטריית תהליכים מגבילה מדי עבורן.

בהגדרה שלנו, המערכת ואף המפרט ממודלים על ידי מכונת מצבים סופית הנקראת משרן. בנוסף, מילת הקלט של המשרן מחולקת לתתי מילים זרות באורך קבוע שנקראות סיבובים, ונאמר שמשרן $\mathcal{T}$ הינו סימטרי בסיבובים של k אם עבור כל תמורה של אותות π, וכל מילת קלט x, נוכל לערבב את האותיות בכל סיבוב בקלט לאחר ההתמרה, $\pi(x)$, כדי לקבל x', כך שהפלט של $\mathcal{T}$ על x' הוא עצמו ערבוב של תוצאת התמורה על הפלט של $\mathcal{T}$ על x. במילים אחרות, כאשר $\mathcal{T}$ הוא סימטרי בסיבובים, יש דרך לערבב את הקלט המתומר, כך שהפלט המתקבל הוא ערבוב של הפלט המתומר (שהוא הפלט "הצפוי" בסימטריית תהליכים).

סימטריה בסיבובים היא סמנטית בכך שהיא לא מתחשבת במבנה, אלא בהתנהגות המערכת. סימטריה בסיבובים מובילה לבעיות ההכרעה הבאות:

- ★ בבעיה הקבועה של סימטריה בסיבובים, ניתן לנו משרן $\mathcal{T}$ וקבוע $k > 0$, ועלינו להחליט אם $\mathcal{T}$ הוא סימטרי בסיבובים של k.
- ★ בבעיה הקיומית של סימטריה בסיבובים, ניתן לנו משרן $\mathcal{T}$, ועלינו להחליט אם קיים $k > 0$ כך ש- $\mathcal{T}$ הוא סימטרי בסיבובים של k.

שימו לב שסימטריה בסיבובים מגדירה תכונה של משרן. הדרך בה אנו ניגשים לבעיות ההחלטה היא על ידי תרגום ההגדרה של סימטריה להגדרה של קשר בין שני משרנים, הנקרא סימולציה בסיבובים, ואז, לאחר שנראה כי ניתן לצמצם סימטריה בסיבובים לסימולציה בסיבובים, אנו פותרים אותה ככזו.

המחקר בוצע בהנחייתו של ד"ר שאול אלמגור, בפקולטה למדעי המחשב.

חלק מן התוצאות בחיבור זה פורסמו כמאמרים מאת המחבר ושותפיו למחקר בכנסים ובכתבי-עת במהלך תקופת מחקר המגיסטר של המחבר, אשר גרסאותיהם העדכניות ביותר הינן:

Antonio Abu Nassar and Shaull Almagor. Simulation by rounds of letter-to-letter transducers, full version. *ArXiv*, abs/2105.01512, 2022.

Antonio Abu Nassar and Shaull Almagor. Simulation by rounds of letter-to-letter transducers. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, 3:1–3:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

סימטריה סמנטית במשרנים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

אנטוניו אבו נסאר

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
אייר התשפ"ב חיפה מאי 2022

סימטריה סמנטית במשרנים

אנטוניו אבו נסאר