Simulation by Rounds of Letter-To-Letter Transducers

Antonio Abu Nassar
Computer Science Department, Technion, Haifa, Israel

Shaull Almagor
Computer Science Department, Technion, Haifa, Israel

Abstract

Letter-to-letter transducers are a standard formalism for modeling reactive systems. Often, two transducers that model similar systems differ locally from one another, by behaving similarly, up to permutations of the input and output letters within "rounds". In this work, we introduce and study notions of simulation by rounds and equivalence by rounds of transducers. In our setting, words are partitioned to consecutive subwords of a fixed length k, called rounds. Then, a transducer $\mathcal{T}_1$ is k-round simulated by transducer $\mathcal{T}_2$ if, intuitively, for every input word x, we can permute the letters within each round in x, such that the output of $\mathcal{T}_2$ on the permuted word is itself a permutation of the output of $\mathcal{T}_1$ on x. Finally, two transducers are k-round equivalent if they simulate each other.

We solve two main decision problems, namely whether $\mathcal{T}_2$ k-round simulates $\mathcal{T}_1$ (1) when k is given as input, and (2) for an existentially quantified k.

We demonstrate the usefulness of the definitions by applying them to process symmetry: a setting in which a permutation in the identities of processes in a multi-process system naturally gives rise to two transducers, whose k-round equivalence corresponds to stability against such permutations.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Concurrency; Theory of computation → Abstraction

Keywords and phrases Transducers, Permutations, Parikh, Simulation, Equivalence

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.3

Related Version *Full Version*: `https://arxiv.org/abs/2105.01512`

Funding *Shaull Almagor*: European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 837327.

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 make 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* [19], 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 [19] and fair simulation [13], but one can also view e.g., linearization of concurrent programs [14] 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 *k-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, called k-rounds. 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. 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 some input $x' \asymp_k x$, such that the outputs of $\mathcal{T}_1$ on x and $\mathcal{T}_2$ on x', denoted y, y' respectively, are also 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 k-round.

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 we can naturally design a transducer $\mathcal{T}_2$ that performs a certain task in 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), arbiters, and other resource allocation systems.

We start with an example demonstrating both benefits.

▶ **Example 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 `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 10-round, 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 by the second author in [1], it is also desirable to find a definition that is suited for deterministic systems. As we show in Section 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

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

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 [7, 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 and does not directly relate to the state space.

Technically, our work is closely related to *commutative automata* [4] and *jumping automata* [10, 18] – 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., [15] (although the analysis here has a different purpose).

Finally, the initial motivation for this work comes from *process symmetry* [1, 6, 9, 16, 17]. We demonstrate the connections in depth in Section 6.

Paper organization. In Section 2 we present some basic definitions used throughout the paper. In Section 3 we introduce k-round simulation and equivalence, define the relevant decision problems, and study some fundamental properties of the definitions. In Section 4 we solve fixed round simulation, while developing some technical tools and characterizations that are reused later. Section 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 Section 6 we use round simulation to obtain a definition of process symmetry for deterministic transducers. Finally, in Section 7, we discuss some variants and open problems.

Due to lack of space, some proofs are omitted and can be found in the full version.

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 automata* (NFA), where $\delta : Q \times \Sigma \to 2^Q$. 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 $q_{i+1} \in \delta(q_i, \sigma_i)$ for all $0 \leq i < n$. The language of $\mathcal{A}$ is defined analogously to the deterministic setting. 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 extend this 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 $q \xrightarrow{w}_{\mathcal{A}} q'$.

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. For a word $w \in \Sigma_I^*$, consider the run $\rho = q_0, \ldots, q_n$ of $\mathcal{T}$ on a word 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.

For a transducer $\mathcal{T}$ and a state s, we denote by $\mathcal{T}^s$ the transducer $\mathcal{T}$ with initial state s.

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

In particular, throughout the paper we consider k-round words $(x, y) \in (\Sigma_I^k \times \Sigma_O^k)^*$. 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[2] $\mathfrak{P} : \Sigma^* \to \mathbb{N}^\Sigma$ 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 [21], for every NFA $\mathcal{A}$ we have that $\mathfrak{P}(L(\mathcal{A}))$ is a *semilinear set*, namely a finite union of sets of the form $\{\boldsymbol{p} + \lambda_1 \boldsymbol{s_1} + \ldots + \lambda_1 \boldsymbol{s_m} : \lambda_1, \ldots, \lambda_m \in \mathbb{N}\}$ where $\boldsymbol{p}, \boldsymbol{s_1}, \ldots, \boldsymbol{s_m} \in \mathbb{N}^d$ (and the translation is effective).

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). Note that in particular this implies $|x| = |y|$.

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)[3], if for every $0 \le r < R$ 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. Clearly $\asymp$ is indeed an equivalence relation.

[2] We use $\mathbb{N}^\Sigma$ rather than $\mathbb{N}^{|\Sigma|}$ to emphasize that the vector's indices are the letters in Σ.
[3] Conveniently, our symbol for round equivalence is a rounded equivalence.

▶ **Example 2** (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 1 that the 3-rounds in y are all permutations of those in x, which gives $x \asymp_3 y$. However, looking at x, y as 4-round words, the number of occurrences of b already in the first 4-round of x and of y is different, so $x \not\asymp_4 y$, as illustrated in Table 2. ⌟

Table 1 x and y are 3-round equivalent.

x	aba	abb	abb	baa
y	baa	bba	abb	aba

Table 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 k-round of x to obtain a new word x', such that the k-rounds of the outputs of $\mathcal{T}_1$ on x and of $\mathcal{T}_2$ on x' are permutations of each other. 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 4.

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 e.g., $\mathcal{T}_1 \prec_k \mathcal{T}_2$.

▶ **Example 3** (Round Robin)**.** We consider a simple version of the Round Robin 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 $\mathcal{T}_i = \langle 2^{\mathcal{P}}, 2^{\mathcal{P}}, Q, q_{(i-1)\%3}, \delta, \ell \rangle$ depicted in Figure 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, $\ell(q_i) = \{i\}$ and $\ell(q_i') = \emptyset$.

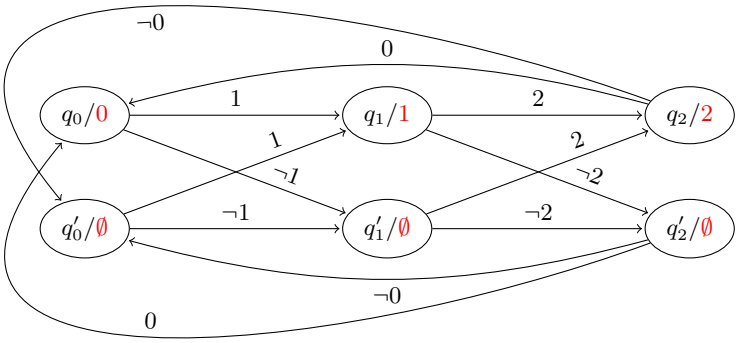

Figure 1 The transducer $\mathcal{T}_i$ for Round Robin, initial state omitted. The input letters σ and $\neg\sigma$ mean all input letters from $2^{\mathcal{P}}$ that, respectively, contain or do not contain σ. The labels are written in red on the states, singleton brackets omitted (e.g., 1 means $\{1\}$).

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 reordering 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 are permutations of the output of $\mathcal{T}_0$ on the non-permuted input. ⌟

▶ **Example 4** (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 2. We claim that $\mathcal{T}_1 \prec_2 \mathcal{T}_2$

Figure 2 Transducers $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (right) illustrate the asymmetry in the definition of round equivalence (see Example 4).

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 $00 \not\succ_2 01$, 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 $k > 0$, 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 5.** *Fixed (resp. existential) round equivalence is 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 mk-round.

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 $\mathcal{T}_1 \prec_{k_2,\Lambda} \mathcal{T}_2$ 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 5, for the purpose of upper-bounds, we focus henceforth on round simulation.

4 Deciding Fixed Round Simulation

In this section we show decidability of fixed round simulation (and, by Lemma 5, fixed round equivalence). The tools we develop will be used in Section 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 k-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.

The Trace DFA. Consider a transducer $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \ell \rangle$, 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 $L(\mathtt{Tr}(\mathcal{T})) = \{(x, y) \in (\Sigma_I \times \Sigma_O)^* : \mathcal{T}(x) = y\}$.

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 $\mathtt{Perm}_k(\mathcal{N}) = \langle \Sigma_I^k \times \Sigma_O^k, S, s_0, \mu, F \rangle$ 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

$$\mu(s, (\alpha, \beta)) = \bigcup \{\eta^*(s, (\alpha', \beta')) : \mathfrak{P}(\alpha') = \mathfrak{P}(\alpha) \wedge \mathfrak{P}(\beta) = \mathfrak{P}(\beta')\}. \tag{1}$$

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 k-rounds α, α' in x and x' we have $\mathfrak{P}(\alpha) = \mathfrak{P}(\alpha')$. Thus, we have the following.

▶ **Observation 6.** $L(\mathtt{Perm}_k(\mathcal{N})) = \{(x, y) \in \Sigma_I^* \times \Sigma_O^* : \exists x' \asymp_k x, y' \asymp_k y, (x', y') \in L(\mathcal{N}) \wedge |x| = |y| = kR \text{ for some } R \in \mathbb{N}\}$

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

▶ **Observation 7** (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 $\Lambda \subseteq \Sigma_I^*$, we identify it with $\Lambda \subseteq \Sigma_I^* \times \Sigma_O^*$ by simply ignoring the Σ_O component. We remind that Λ denotes both a language and a corresponding DFA or NFA.

▶ **Lemma 8.** *Consider transducers $\mathcal{T}_1, \mathcal{T}_2$, an* NFA Λ *and $k > 0$. Let $\mathcal{A}_1^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda)$ (where the intersection is obtained by the product* NFA*) and $\mathcal{A}_2^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$, then*

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

Proof. Recall that $\mathtt{Tr}(\mathcal{T})$ accepts a word (x', y') iff $\mathcal{T}(x') = y'$. The claim then follows from Observation 6, by replacing the expression $y \asymp y' \wedge (x', y') \in L(\mathtt{Tr}(\mathcal{T}))$ with the equivalent expression $\mathcal{T}(x') \asymp_k y$. ◀

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

▶ **Lemma 9.** *Consider transducers $\mathcal{T}_1, \mathcal{T}_2$, an* NFA Λ *and $k > 0$. Let $\mathcal{A}_1^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda)$ and $\mathcal{A}_2^k = \mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$, 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 8, 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 8 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 8, 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 10. The proof of Lemma 9 can be simplified by using instead of $\mathcal{A}_1^k$, the augmentation of $\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda$ to k-round words. However, such a DFA 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 9 shows that deciding fixed round equivalence amounts to deciding containment of NFAs. By analyzing the size of the NFAs, we obtain the following.

▶ **Theorem 11.** *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 9, 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. ◀

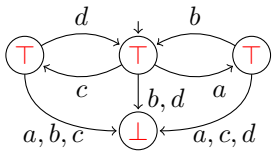

Figure 3 The transducer $\mathcal{T}_1$ in the proof of Theorem 12.

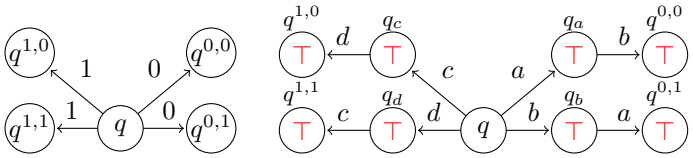

Figure 4 Every state and its 4 transitions in $\mathcal{N}$ (left) turn into 8 transitions in $\mathcal{T}_2$ (right). All transitions not drawn in the right figure lead to $q_\perp$, a sink state labelled $\perp$.

We now give a PSPACE-hardness lower bound, thus concluding the problem is PSPACE-complete. By Lemma 5, we give a stronger lower bound already for round-equivalence.[4]

▶ **Theorem 12.** *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 for a fixed Λ (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, to round-equivalence with $k = 2$ and with Λ given as a DFA of constant size. See the full version for a proof of PSPACE-hardness of the former 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 $\sigma \in \{0,1\}$.

We construct two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ over input and output alphabets $\Sigma_I = \{a,b,c,d\}$ and $\Sigma_O = \{\top, \perp\}$ and $\Lambda \subseteq \Sigma_I^*$, such that $L(\mathcal{N}) = \{0,1\}^*$ iff $\mathcal{T}_1 \equiv_{2,\Lambda} \mathcal{T}_2$.

Set $\Lambda = (ab + cd)^*$. 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, $\mathcal{T}_1$ (Figure 3) keeps outputting $\top$ for all inputs in Λ, thus mimicking a universal language 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 (see Figure 4). Hence, when we are allowed to permute ab and ba by round equivalence, we capture the nondeterminism of $\mathcal{N}$. The outputs in $\mathcal{T}_2$ are all $\top$, except a sink state $q_\perp$ labelled $\perp$, which is reached upon any undefined transition (including transitions from states of $\mathcal{N}$ that do not have an outgoing 0 or 1 transition).

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

▶ **Corollary 13.** *Given transducers $\mathcal{T}_1, \mathcal{T}_2$, an* NFA *$\Lambda$, and $k > 0$ in unary, the problem of deciding whether $\mathcal{T}_1 \prec_{k,\Lambda} \mathcal{T}_2$ is* PSPACE*-complete.*

5 Deciding Existential Round Simulation

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 9, 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.

[4] The reduction in Lemma 5 is a Turing reduction. Nonetheless, our PSPACE-hardness proof actually explicitly shows the hardness of both simulation and equivalence.

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 small-model property for k: in Theorem 14, the main result of this section, we give an upper bound on the minimal $k > 0$ for which $\mathcal{T}_1 \prec_k \mathcal{T}_2$. 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 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 (namely the set of letter types occurring in the NFA), up to multiplicities of the letter types.

Recall that by Lemma 9, 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 between $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1))$ and $\mathtt{Perm}_{k'}(\mathtt{Tr}(\mathcal{T}_1))$ may differ from the one between $\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2))$ 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 are now left with the problem of bounding the minimal k after which all type profiles have been exhausted. 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 attain the overall bound. The main result of this section is the following.

▶ **Theorem 14.** *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 9 immediately entails the following.

▶ **Corollary 15.** *Existential round simulation is decidable.*

We prove Theorem 14 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 (Lemma 16). We then introduce Presburger Arithmetic and its relation to Parikh's theorem. In Lemma 17 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 products", which are the product automata mentioned above, that serve to unify the types between $\mathcal{T}_1$ and $\mathcal{T}_2$. In Observations 18 and 19 we formalize the connection of these products to the transducers $\mathcal{T}_1, \mathcal{T}_2$. We then define the type profiles mentioned above, and prove in Lemma 20 that they exhibit a semilinear behaviour. Finally, in Lemma 21 we prove that when two redundant-product automata have the same type profile, then the containment mentioned above can be shown. We conclude by combining these results to obtain Theorem 14.

5.2 Proof of Theorem 14

Type Matrices of Parikh Vectors. Consider the alphabet $\Sigma_I^k \times \Sigma_O^k$ for some $k > 0$. Recall that by Observation 7, permutation closure NFAs are permutation invariant, and from Section 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 the definition of types is "non-uniform", since we use different automata to extract the type of words of different length. We obtain a more uniform description as follows (see the full version for the proof).

▶ **Lemma 16.** *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 exist $(\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k$ with $\mathfrak{P}(\alpha) = \boldsymbol{p}$ and $\mathfrak{P}(\beta) = \boldsymbol{o}$ such that $s_1 \xrightarrow{(\alpha,\beta)}_{\mathcal{N}} s_2$.*

Presburger Arithmetic. The first ingredient in the proof of Theorem 14 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 [12] (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 $\{(a_1, \ldots, a_d) \in \mathbb{N}^d : (a_1, \ldots, a_d) \models \varphi(x_1, \ldots, x_d)\}$. For example, the formula $\varphi(x_1, x_2) := x_1 < x_2 \wedge \exists y. x_1 = 2y$ defines the set $\{(a, b) \in \mathbb{N}^2 : a < b \wedge a \text{ is even}\}$.

A fundamental result about PA states that the definable sets in PA are exactly the semilinear sets. In particular, by Parikh's theorem we have that for every NFA $\mathcal{A}$, $\mathfrak{P}(L(\mathcal{A}))$ is PA definable. In fact, by [22], 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 17.** *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 $\{(\boldsymbol{p}, \boldsymbol{o}) \in \mathbb{N}^{\Sigma_I} \times \mathbb{N}^{\Sigma_O} : \tau_{\mathcal{N}}(\boldsymbol{p}, \boldsymbol{o}) = \tau\}$ 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 16, 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 exist 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 $s_1, s_2 \in S$ we have 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 $(\boldsymbol{p}, \boldsymbol{o}) \in \mathfrak{P}(L(\mathcal{N}^{s_1}_{s_2}))$. 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 $\mathcal{D}_i = \langle \Sigma_I \times \Sigma_O, S_i, s^i_0, \eta_i, F_i \rangle$, then the product automaton is defined as $\mathcal{B}_i = \langle \Sigma_I \times \Sigma_O, S_1 \times S_2, (s^1_0, s^2_0), \eta_1 \times \eta_2, G_i \rangle$, 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 18.** *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 6, $L(\mathtt{Perm}_k(\mathcal{D}_i))$ depends only on $L(\mathcal{D}_i)$. We thus have the following.

▶ **Observation 19.** *For every* $k > 0$ *we have* $L(\mathtt{Perm}_k(\mathcal{B}_1)) = L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_1) \cap \Lambda))$ *and* $L(\mathtt{Perm}_k(\mathcal{B}_2)) = L(\mathtt{Perm}_k(\mathtt{Tr}(\mathcal{T}_2)))$.

Type Profiles. We now consider the set of types induced by the redundant product constructions $\mathcal{B}_1$ and $\mathcal{B}_2$ on Parikh vectors of words of length k. By Item 2 of Observation 18, it's enough to consider $\mathcal{B}_1$.

For $k > 0$, we define the *k-th type profile* of $\mathcal{B}_1$ to be $\Upsilon(\mathcal{B}_1, k) = \{\tau_{\mathcal{B}_1}(\mathfrak{P}(\alpha), \mathfrak{P}(\beta)) : (\alpha, \beta) \in \Sigma_I^k \times \Sigma_O^k\}$, i.e., the set of all types of Parikh vectors $(\boldsymbol{p}, \boldsymbol{o})$ with $|\boldsymbol{p}| = |\boldsymbol{o}| = k$ that are induced by $\mathcal{B}_1$. 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 distinct type profile will have been encountered. We now place an upper bound on K_0.

▶ **Lemma 20.** *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 17 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 [11, 3], 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 $K_0 = \max\{M_T : \Theta_T(z) \text{ is satisfiable}\}$. 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 20 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 9 and Observation 19). This is captured in the following.

▶ **Lemma 21.** *Let $0 < k \neq k'$ such that $\Upsilon(\mathcal{B}_1, k') = \Upsilon(\mathcal{B}_1, k)$, then $L(\mathtt{Perm}_k(\mathcal{B}_1)) \subseteq L(\mathtt{Perm}_k(\mathcal{B}_2))$ 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 (α'_j, β'_j) in w (over $\Sigma_I^{k'} \times \Sigma_O^{k'}$) 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 $(\tau_{\mathtt{Perm}_k(\mathcal{B}_1)}(x, y))_{s_0^1, s_f^1} = 1$ 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 Parikh-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 18 and the fact that the permutation construction 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. ◀

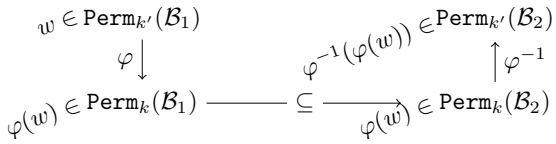

Figure 5 A diagram for the proof structure of Lemma 21.

Combining Lemmas 20 and 21, we can effectively compute K_0 such that if $L(\mathcal{A}_1^k) \subseteq L(\mathcal{A}_2^k)$ for some k, then this holds for some $k < K_0$. Finally, using Lemma 9, this concludes the proof of Theorem 14.

▶ Remark 22 (Complexity Results for Theorem 14 and Corollary 15). Let n be the number of states in $\mathcal{T}_1 \times \mathcal{T}_2$. Observe that the formula Ψ_τ constructed in Lemma 17 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 20 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. [5]

Using quantifier elimination [8, 20], 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 9 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. See Section 7 for additional comments.

5.3 Lower Bounds for Existential Round Simulation

The complexity bounds in Remark 22 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 14.

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

▶ **Theorem 23.** *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 a fixed Λ (given as a 5-state* DFA*).*

Proof sketch. We present a similar reduction to that of Theorem 12, from universality of NFAs (see the full version). 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 $\mathcal{T}_2$). See the full version for the complete details.

▶ **Example 24** (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 6 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.

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

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 $\mathcal{T}_2$.

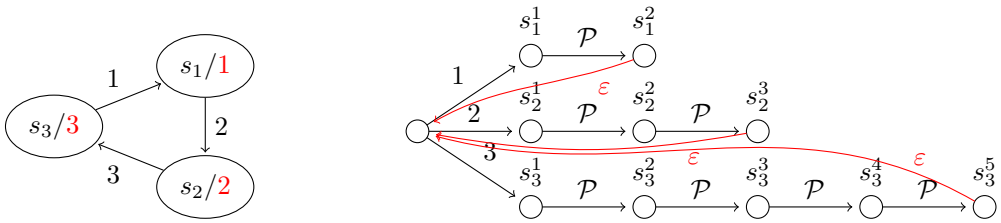

Figure 6 The transducers $\mathcal{T}_1$ (left) and $\mathcal{T}_2$ (right) for $m = 3$ in Example 24. The ε-edge from a state $s_i^{p_i}$ to s_0 in $\mathcal{T}_2$ mean that the transition function from state $s_i^{p_i}$ behaves identically as from s_0.

6 From Process Symmetry to Round Equivalence

As mentioned in Section 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. 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 e.g., ports, and the processes not knowing which port they are plugged into. Thus, the input received may be any permutation of the actual identities of the processes. Then, a transducer is *process symmetric*, if the outputs are permuted similarly to the inputs. For example, in the RR scheduler, the output corresponding to input $\{1, 2\}\{3\}\{3\}$ is $\{1\}\emptyset\{3\}$. However, if we permute the inputs by swapping 1 and 3, the output for $\{3, 2\}\{1\}\{1\}$ is $\emptyset\emptyset\emptyset$, so RR is not process symmetric.

In [1], 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 wriggle room, by letting the transducer do some local order changes in the word that correspond to the permutation. Thus, e.g., if we were allowed to rearrange 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 (i.e., without any permutation) $\{1\}\emptyset\{3\}$. 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 $\mathcal{T} = \langle 2^{\mathcal{P}}, 2^{\mathcal{P}}, Q, q_0, \delta, \boldsymbol{\ell} \rangle$ is *k-round symmetric* if for every permutation π of $\mathcal{P}$ for and every k-round word $x \in (2^{\mathcal{P}})^*$ there exists a k-round word $x' \in (2^{\mathcal{P}})^*$ 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.

Here, 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 hence 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 $\pi^{-1}(\sigma)$. Formally, $\mathcal{T}^\pi = \langle 2^\mathcal{P}, 2^\mathcal{P}, Q, q_0, \delta^\pi, \boldsymbol{\ell}^\pi \rangle$ 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 Sections 4 and 5 to solve the problems at hand (see the full version for the proof).

▶ **Lemma 25.** *For a permutation π and $k > 0$, $\mathcal{T}$ is k-round symmetric w.r.t. π iff $\mathcal{T}^\pi \prec_k \mathcal{T}$.*

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 (see the full version for the proof).

▶ **Lemma 26.** *Consider two permutations π, χ. If $\mathcal{T}^\pi \prec_k \mathcal{T}$ and $\mathcal{T}^\chi \prec_k \mathcal{T}$ then $\mathcal{T}^{\pi\circ\chi} \prec_k \mathcal{T}$.*

Recall that the group of all permutations of $\mathcal{P}$ is generated by two permutations: the transposition $(1\ 2)$ and the cycle $(1\ 2\ \ldots\ m)$ [5]. By Lemma 26 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 27.** *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 commutative, as was the case with round symmetry v.s. round equivalence. However, when we consider round symmetry w.r.t. to all permutations, the definition becomes inherently symmetric, as a consequence of Lemma 26 (see the full version for the proof).

▶ **Lemma 28.** *In the notations above, if $\mathcal{T}^\pi \prec_k \mathcal{T}$ then $\mathcal{T} \prec_k \mathcal{T}^\pi$.*

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

7 Future Work

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, we plan to extend this study to other definitions, such as *window simulation*, where we use a sliding window of size k instead of disjoint k-rounds, and *Parikh round symmetry*, where the alphabet is of the form $2^\mathcal{P}$, and we are allowed not only to permute the letters in each round, but also to shuffle the individual signals between letters in the round. In addition, the setting of infinite words is of interest, where one can define *ultimate simulation*, requiring the simulation to only hold after a finite prefix. Finally, other types of transducers may require variants of simulation, such as probabilistic transducers, or streaming-string transducers [2].

References

1. S. Almagor. Process symmetry in probabilistic transducers. In *40th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2020*, 2020.
2. R. Alur. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010*, 2010.
3. 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.
4. J. A. Brzozowski and I. Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, 1973.
5. P. J. Cameron et al. *Permutation groups*, volume 45. Cambridge University Press, 1999.
6. 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.
7. E.M. Clarke, T.A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018.
8. D. C Cooper. Theorem proving in arithmetic without multiplication. *Machine intelligence*, 7(91-99):300, 1972.
9. E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal methods in system design*, 9(1-2):105–131, 1996.
10. 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.
11. 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. URL: `https://books.google.co.il/books?id=ijoNHAAACAAJ`.
12. 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`.
13. T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *Proc. 8th Conferance on Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, Warsaw, July 1997. Springer-Verlag.
14. 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.
15. 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.
16. C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal methods in system design*, 9(1-2):41–75, 1996.
17. 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.
18. A. Meduna and P. Zemek. Jumping finite automata. *International Journal of Foundations of Computer Science*, 23(07):1555–1578, 2012.
19. 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.
20. D. C. Oppen. A 222pn upper bound on the complexity of presburger arithmetic. *Journal of Computer and System Sciences*, 16(3):323–332, 1978.
21. R. J. Parikh. On context-free languages. *J. of the ACM*, 13(4):570–581, 1966.
22. 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.