The Complexity of Transducer Synthesis from Multi-Sequential Specifications

Léo Exibard[1]
Aix-Marseille Université, Marseille, France
Université libre de Bruxelles, Brussels, Belgium
leo.exibard@ulb.ac.be

Emmanuel Filiot[2]
Université libre de Bruxelles, Brussels, Belgium
efiliot@ulb.ac.be

Ismaël Jecker[3]
Université libre de Bruxelles, Brussels, Belgium
ismael.jecker@ulb.ac.be

Abstract

The transducer synthesis problem on finite words asks, given a specification $S \subseteq I \times O$, where I and O are sets of finite words, whether there exists an implementation $f : I \to O$ which (1) fulfils the specification, i.e., $(i, f(i)) \in S$ for all $i \in I$, and (2) can be defined by some input-deterministic (aka sequential) transducer $\mathcal{T}_f$. If such an implementation f exists, the procedure should also output $\mathcal{T}_f$. The realisability problem is the corresponding decision problem.

For specifications given by synchronous transducers (which read and write alternately one symbol), this is the finite variant of the classical synthesis problem on ω-words, solved by Büchi and Landweber in 1969, and the realisability problem is known to be ExpTime-c in both finite and ω-word settings. For specifications given by asynchronous transducers (which can write a batch of symbols, or none, in a single step), the realisability problem is known to be undecidable.

We consider here the class of multi-sequential specifications, defined as finite unions of sequential transducers over possibly incomparable domains. We provide optimal decision procedures for the realisability problem in both the synchronous and asynchronous setting, showing that it is PSpace-c. Moreover, whenever the specification is realisable, we expose the construction of a sequential transducer that realises it and has a size that is doubly exponential, which we prove to be optimal.

2012 ACM Subject Classification Theory of computation → Logic and verification, Theory of computation → Transducers

Keywords and phrases Transducers, Multi-Sequentiality, Synthesis

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.46

Acknowledgements We warmly thank the anonymous reviewers for their helpful comments and Christof Löding for pointing us to some related references.

[1] L. Exibard is a PhD student funded by a FRIA fellowship from the F.R.S.-FNRS.
[2] E. Filiot is a research associate of F.R.S.- FNRS. He is supported by the ARC Project Transform Fédération Wallonie-Bruxelles and the FNRS CDR project J013116F.
[3] I. Jecker is an "aspirant FNRS" PhD student, funded by the F.R.S.-FNRS.

43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 46; pp. 46:1–46:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The realisability and synthesis problem. In general, the realisability problem is given by some input and output data domains $D_{\mathtt{i}}, D_{\mathtt{o}}$, a *specification* $S \subseteq D_{\mathtt{i}} \times D_{\mathtt{o}}$ defining for every $d \in D_{\mathtt{i}}$ the set of allowed outputs $\{d' \in D_{\mathtt{o}} \mid (d, d') \in S\}$ (assumed to be non-empty) and a class of target *implementations* $\mathcal{I}$ consisting of (total) functions $D_{\mathtt{i}} \to D_{\mathtt{o}}$. It asks whether there exists $f \in \mathcal{I}$ such that for all $d \in D_{\mathtt{i}}$, $(d, f(d)) \in S$, i.e., the implementation f satisfies the specification. The synthesis problem asks to generate (a representation of) f. So, instead of designing f and verifying its correctness *a posteriori*, a synthesis algorithm automatically generates f from it, making it *correct by construction*. The underlying idea behind synthesis is that the specification may be written in a high-level language, e.g. a logic, and an implementation is a low-level computational model e.g. an automaton. It is based on the assumption that it is less error-prone to design a specification, i.e. to describe *what* a system has to do, than designing the system itself, i.e. describing *how* it must do it.

Synchronous transducer synthesis. In the original setting defined by Church [7, 29], $D_{\mathtt{i}}, D_{\mathtt{o}}$ are sets of ω-words over two alphabets $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$ respectively, and the specification S is given by an ω-language $L \subseteq (\Sigma_{\mathtt{i}} \times \Sigma_{\mathtt{o}})^\omega$ as follows: $S = \{(\pi_1(w), \pi_2(w)) \mid w \in L\}$, where π_i is the projection on the ith component. The language L is represented by an MSO-sentence or, equivalently, an automaton. Such automata are also called (non-deterministic) *synchronous transducers*, as they can be seen as machines alternately reading one input symbol and synchronously producing one output symbol. In Church's setting, the target implementations are synchronous *sequential* transducers (also called *input-deterministic*): they alternately read one input symbol and deterministically produce a symbol to output. Determinism is required because implementations are required to use only finite-memory. The Church's instance of the realisability problem is decidable if the specification is given in MSO and ExpTime-c if it is given by a synchronous transducer [21]. For LTL specifications, it is 2ExpTime-c [27]. Motivated by reactive systems, the synthesis problem from LTL specifications has been revisited recently with efficient symbolic methods [20, 28, 14, 10, 18]. The synthesis problem in general has also motivated an active research on infinite games [1, 9, 5].

Asynchronous transducer synthesis. In the asynchronous setting, specifications may not strictly alternate between input and output symbols, hence they can no longer be seen as languages over $\Sigma_{\mathtt{i}} \times \Sigma_{\mathtt{o}}$. Similarly, the target implementations may not be synchronous: the system can delay its production of outputs, or produce several symbols at once. Transducers, in contrast to synchronous transducers, are by definition asynchronous: their transitions are labelled by pairs (i, w) where $i \in \Sigma_{\mathtt{i}}$ is a symbol and $w \in \Sigma_{\mathtt{o}}^*$ a word, possibly empty. Since they are generally non-deterministic, to a single input word may correspond several output words, and thus transducers define subsets of $\Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{o}}^*$. Therefore, they are well-suited to represent (asynchronous) specifications, and in their sequential version, asynchronous implementations. Any asynchronous specification is realisable by some unambiguous (functional) asynchronous transducer [24, 11, 3]. However, evaluating unambiguous transducers on arbitrarily long or even infinite input words may require arbitrarily large memory. Therefore, just as in Church's setting, a sequentiality requirement can be put on implementations for efficient memory usage. However, the realisability of asynchronous specifications by (asynchronous) sequential transducers, which is called the *sequential uniformisation problem* in transducer-theoretic terms, is undecidable for finite words [4, 12]. If the specification is finite-valued (i.e. any input word has a constant number of output words), the problem is in 3ExpTime [12]. The

proof of [12] is based on Ramsey's theorem and word combinatorics arguments, and it is not clear how to reduce the complexity. This raises the question of whether there are natural and non-trivial subclasses of asynchronous specifications with better complexity.

Multi-sequential specifications. In this paper, we consider a class of specifications S on finite words, i.e. $S \subseteq \Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{o}}^*$, which strictly restricts the class of finite-valued specifications to so-called multi-sequential specifications. Such class is obtained by closure under finite unions of graphs of sequential functions. Precisely, $S = \bigcup_{i=1}^{n} S_i$ where S_i is the graph of a (partial) function $f_i : \Sigma_{\mathtt{i}}^* \to \Sigma_{\mathtt{o}}^*$ defined by a sequential transducer. Likewise, a transducer is multi-sequential if it is a union of state-disjoint sequential transducers. For instance, consider the specification S consisting of the pairs (w, w') such that w' is a subword of w of fixed length k. This specification is multi-sequential: $S = \bigcup_{w' \in \Sigma_{\mathtt{i}}^k} S_{w'}$ where $S_{w'} = \{(w, w') \mid w' \text{ subword of } w\}$. Clearly, $S_{w'}$ can be represented by a sequential transducer, because w' is fixed: once the first symbol of w' is met in w, output it, and proceed to the second symbol of w', etc., until the last symbol of w' is produced, reject otherwise. The notion of multi-sequentiality has been introduced for functions in [6] and studied for relations in [19]. An important property of the class of multi-sequential specifications is its decidability in PTime: Given a transducer, it is decidable in PTime whether it defines a multi-sequential specification [19]. This fact and their natural definition as closure of sequential functions under union make multi-sequential specifications a good candidate for a class of specifications with better complexity than the known results of the literature.

Contributions. We investigate the complexity of sequential transducer synthesis from specifications defined by multi-sequential transducers on finite words. We show that both in the synchronous and asynchronous settings, the realisability problem is PSpace-complete. To the best of our knowledge, it is the first non-trivial class of specifications which admits a realisability test below ExpTime. If the specification is realisable, we show how to extract an implementation as a winning strategy in a two-player game called the *synthesis game*. It is parameterised by a value $k \in \mathbb{N}$ which bounds the maximal number of output symbols which can be queued before being outputted, allowing for an incremental synthesis algorithm. To keep track of such output symbols, we use the notion of *delay* [2].

Difficulties and examples. Let us briefly explain what are the main difficulties to overcome. Consider $S = \bigcup_i S_i$ a multi-sequential specification. If all of the S_i have disjoint domains, then S is a function, which is realisable by a sequential transducer iff it is sequential. The latter can be tested in PTime [2]. The problem becomes more interesting and challenging when the S_i have domains which are not necessarily disjoint. Consider for example the 2-sequential transducer $\mathcal{D}_1 \cup \mathcal{D}_2$ of Fig. 1. The transducer $\mathcal{D}_1$ accepts the words containing at least two a's, and replaces b's with a's, and $\mathcal{D}_2$ accepts the words containing at least one b, and replaces a's with b's. This specification can be realised by a sequential transducer which waits two steps before outputting something, since it then knows whether the input contains at least one b or two a's. It then behaves as $\mathcal{D}_1$ in the first case, and as $\mathcal{D}_2$ in the second.

This example shows that a sequential realiser may have to wait before reacting, keeping in memory what remains to be output in the future. Take on the contrary the 2-sequential transducer $\mathcal{D}'_1 \cup \mathcal{D}'_2$ of Fig. 1, which is the same as $\mathcal{D}_1 \cup \mathcal{D}_2$ except that it can additionally read and copy c's at any moment. In that case, a sequential realiser would have to store arbitrary long sequences of c's before outputting them, for instance when processing words in $ac^*\{a, b, c\}^*$. In particular, this specification is not sequentially realisable.

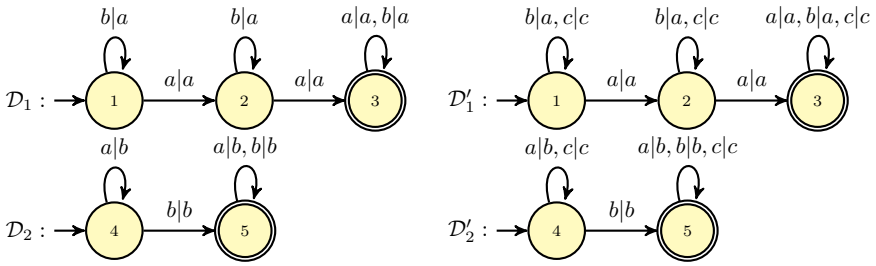

Figure 1 Four synchronous sequential transducers.

Structure of the paper. After a formal definition of transducer synthesis (Section 2), we solve the synchronous case and provide a characterisation of realisable synchronous multi-sequential specifications, decidable in PSpace (Section 3). Then, we present the notion of synthesis game (Section 4), which is a useful tool for the proofs and also to get a synthesis procedure. For the asynchronous setting, we define a recursive characterisation of realisable multi-sequential specifications and show that it can be decided in PSpace (Section 5).

Related work. Games with delays have been used in [4, 15]. Perhaps the closest formulation to ours is that of [4]. However, it is tailored to automatic relation. Our game structure is more general, as it is defined for uniformising any transducer (defining a rational relation). In particular, our game structure is exponentially larger that the one of [4].

We would also like to mention an interesting related line of works on ω-words, where the specification is synchronous, but the implementation may be asynchronous [15, 17, 22, 23, 30, 31, 32]. Unlike the setting where the specification and implementations are both asynchronous, the realisability problem is decidable here, for ω-regular specifications (i.e., regular ω-languages over $\Sigma_{\mathtt{i}} \times \Sigma_{\mathtt{o}}$), and ExpTime-c if the specification is given by a parity automaton [22]. In this setting, the authors often consider a notion of delay games. In these games, the delay is a quantitative notion, corresponding to the waiting time before outputting a symbol, while for us, a delay is a word that still remains to be output (this is a standard terminology in transducer theory). It is known in particular that constant "waiting time" (depending on the specification) is always sufficient to win, for ω-regular specifications. This is different to our setting: for instance, the function f mapping any word of the form $a^n\sigma$, for $n > 0$ and $\sigma \in \{a, b\}$, to σ is realisable by a sequential transducer, but the production of a and b might have to be delayed for an unbounded amount of time.

2 Transducer synthesis problem

Words. For an alphabet Σ, we denote by Σ^* the set of finite words over it, and by ϵ the empty word. The length $|w|$ of a word w is its number of symbols. For $k \in \mathbb{N}$, we denote by Σ^k (resp. $\Sigma^{\leq k}$) the set of words of length k (resp. at most k). For $u, v \in \Sigma^*$, we write $u \preceq v$ if u is a prefix of v, and denote by $u^{-1}v$ the word such that $u(u^{-1}v) = v$. For $L \subseteq \Sigma^*$, the residual language $u^{-1}L$ is $u^{-1}L = \{u' \mid uu' \in L\}$. Given $S \subseteq \Sigma^* \times \Gamma^*$, and $(u, v) \in \Sigma^* \times \Gamma^*$, the residual relation $(u, v)^{-1}S$ is defined by $(u, v)^{-1}S = \{(u', v') \mid (uu', vv') \in S\}$.

Automata. In this paper, finite (non-deterministic) automata over an alphabet Σ are denoted as tuples $\mathcal{A} = (\Sigma, Q, I, F, \Delta)$ where Σ is the alphabet, Q the set of states, among which I (resp. F) denotes the initial (resp. final or accepting) states, and $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation. $\mathcal{A}$ is deterministic if there is only one initial state and for all $(q, \sigma) \in Q \times \Sigma$, there exists at most one $q' \in Q$ such that $(q, \sigma, q') \in \Delta$.

A *run* of $\mathcal{A}$ on a word $w = \sigma_1 \dots \sigma_n$ consists in either a single state $q \in Q$ if $n = 0$, or a sequence $r \in \Delta^*$ of n transitions $t_1 \dots t_n$ such that the target state of t_i equals the source state of t_{i+1} for all $1 \leq i < n$. It is said to be initial if the source state of t_1 is initial, and accepting if the target state of t_n is accepting. If p is the source state of t_1 and q the target state of t_n, we may write $p \xrightarrow{w}_{\mathcal{A}} q$ to mean that there exists a run from p to q on w. The language accepted by an automaton $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words admitting an accepting run. A state $q \in Q$ is reachable (resp. co-reachable) if there is a run from an initial state (resp. to a final state) for some $u \in \Sigma^*$. A state is said to be *useful* if it is both reachable and co-reachable, and $\mathcal{A}$ is said to be *trim* if all its states are useful. It is well-known that any automaton can be transformed into an equivalent trim automaton in PTime. Given two automata $\mathcal{A}_1 = (\Sigma, Q_1, I_1, F_1, \Delta_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, I_2, F_2, \Delta_2)$, their disjoint union $\mathcal{A}_1 \uplus \mathcal{A}_2$ is the automaton $(\Sigma, Q_1 \uplus Q_2, I_1 \uplus I_2, F_1 \uplus F_2, \Delta_1 \uplus \Delta_2)$.

Transducers. A *transducer*[4] over two alphabets Σ, Γ is a tuple $\mathcal{T} = (\mathcal{A}, \rho, \tau)$ such that $\mathcal{A} = (\Sigma, Q, I, F, \Delta)$ is an automaton over Σ, called the *input automaton*, $\rho : \Delta \to \Gamma^*$ is a mapping, called the output function, associating with every transition an output word, and $\tau : F \to \Gamma^*$ is a terminal function associating with every accepting state an output word. Given a run r of $\mathcal{A}$ on a word w, its output $\text{out}(r) \in \Gamma^*$ is defined by ϵ if $w = \epsilon$, and by $\rho(t_1) \dots \rho(t_n)$ if $r = t_1 \dots t_n$ for some $n \geq 1$. We write $p \xrightarrow{u|v}_{\mathcal{T}} q$ whenever there exists a run r of $\mathcal{A}$ on u from p to q, such that $v = \text{out}(r)$, and say that r *produces* v. The *relation* defined by $\mathcal{T}$ is the set $[\![\mathcal{T}]\!]$ of pairs $(u, v\tau(q)) \in \Sigma^* \times \Gamma^*$ such that $p \xrightarrow{u|v}_{\mathcal{T}} q$ for $p \in I$ and $q \in F$. We define $\text{dom}(\mathcal{T})$ by $\text{dom}(\mathcal{T}) = \text{dom}([\![\mathcal{T}]\!]) = L(\mathcal{A})$.

A transducer is *trim* if its input automaton is trim. It is called *sequential* if its input automaton is deterministic, and *functional* if $[\![\mathcal{T}]\!]$ is a function, i.e. for all $u \in \text{dom}(\mathcal{T})$, there exists at most one pair $(u, v) \in [\![\mathcal{T}]\!]$. In that case we let $\mathcal{T}(u) = v$. Note that any sequential transducer is functional. A transducer $\mathcal{T} = (\mathcal{A}, \rho, \tau)$ is called *synchronous* (or sometimes *letter-to-letter* in the literature) if, whenever it reads an input symbol, it produces exactly one output symbol, i.e. for all transition t, $|\rho(t)| = 1$, and $\tau(q) = \epsilon$ for all accepting state q. For example, consider the transducer $\mathcal{D}_1$ on Fig. 1 (the terminal function is assumed to output ϵ and is not depicted). It is sequential and synchronous. Its domain is $L = b^*ab^*a(a + b)^*$.

Two transducers are said to be *equivalent* if they define the same relation. Finally, the disjoint union of transducers is naturally defined as the disjoint union of their input automata and the disjoint union of their output functions (seen as graphs). For all transducers $\mathcal{T}_1, \mathcal{T}_2$, we have $[\![\mathcal{T}_1 \uplus \mathcal{T}_2]\!] = [\![\mathcal{T}_1]\!] \cup [\![\mathcal{T}_2]\!]$.

Transducer Synthesis Problem. Let $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$ be two alphabets of input and output symbols respectively. They may not necessarily be disjoint. A *specification* is a subset of $\Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{o}}^*$, and an *implementation* is a function, possibly partial, from $\Sigma_{\mathtt{i}}^*$ to $\Sigma_{\mathtt{o}}^*$. The transducer *realisability* problem asks, given a specification S defined by a transducer $\mathcal{T}$, i.e. $S = [\![\mathcal{T}]\!]$, whether there exists a sequential transducer $\mathcal{I}$ such that (1) $\text{dom}(\mathcal{I}) = \text{dom}(\mathcal{T})$ and (2) for all $u \in \text{dom}(\mathcal{T})$, $(u, \mathcal{I}(u)) \in [\![\mathcal{T}]\!]$. In that case, we say that $\mathcal{I}$ realises S (or $\mathcal{T}$), and that S is realisable by a sequential transducer, or sequentially realisable. We also say that $\mathcal{I}$ is a *realiser* of S. The synthesis problem asks to output $\mathcal{I}$. The realisability problem is undecidable in general [4, 12], but decidable, in 3ExpTime, if $\mathcal{T}$ is finite-valued, i.e. there exists $k \in \mathbb{N}$ such that for all $u \in \text{dom}(\mathcal{T})$, $|\{v \mid (u, v) \in [\![\mathcal{T}]\!]\}| \leq k$ [12].

[4] Our definition is sometimes called *real-time* transducer in the literature, in contrast to transducers with ϵ-input transitions. For the purpose of this paper, this does not make a difference.

Multi-sequential specifications. A transducer $\mathcal{T}$ is called *k-sequential* if it is the disjoint union of k sequential transducers. It is called *multi-sequential* if it is k-sequential for some k. Observe that when the k sequential transducers have pairwise disjoint domains, then $\mathcal{T}$ is functional, but it may not be the case in general. Deciding whether given a transducer $\mathcal{T}$, there exists an equivalent multi-sequential transducer $\mathcal{T}'$, can be done in PTime; however, $\mathcal{T}'$ may be exponentially larger than $\mathcal{T}$ [19]. Minimising the number of sequential transducers of the disjoint union is also doable: deciding whether $\mathcal{T}$ is equivalent to some k-sequential transducer for k given in unary is decidable in PSpace [8]. In this paper, we consider *multi-sequential specification*, i.e. relations $S \subseteq \Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{o}}^*$ defined by multi-sequential transducers.

PSpace-hardness. In both the synchronous and asynchronous case, the realisability problem of multi-sequential specifications by (a)synchronous sequential transducers is PSpace-hard.

We build a reduction from the emptiness problem of the intersection of n DFA $\mathcal{A}_1, \dots, \mathcal{A}_n$ on some alphabet Σ, proven PSpace-c in [25]. We define a specification S over $\Sigma \cup \{\#, a, b\}$ by $S = \bigcup_{i=1}^{n}(S_i \cup N_i)$ where $S_i = \{(w\#^m\sigma, w\sigma\#^m) \mid \sigma \in \{a,b\}, m \geq 0, w \in L(\mathcal{A}_i)\}$ and $N_i = \{(w\#^m\sigma, w\#^m\sigma) \mid \sigma \in \{a,b\}, m \geq 0, w \notin L(\mathcal{A}_i)\}$. If there exists $w \in \bigcap_{i=1}^{n} L(\mathcal{A}_i)$, then on the domain $w\#^*\{a,b\}$, the specification is a function which is not definable by any sequential transducer, thus not sequentially realisable, since it would imply counting the #s (in the synchronous setting, it suffices to take $m = 1$ since a synchronous transducer would be forced to guess the future). Conversely, if $\bigcap_{i=1}^{n} L(\mathcal{A}_i) = \varnothing$, then the identity function (trivially definable by a synchronous sequential transducer) realises the specification.

It is readily seen that each S_i (resp. N_i) is definable by a 2-(resp. 1-)sequential transducer, hence S is multi-sequential, concluding the proof.

3 The synchronous setting

In this section, we consider first the synchronous setting, where the specification is given as a disjoint union of synchronous sequential transducers, and the target implementations are synchronous sequential transducers. Not only is this setting interesting in itself, but it helps to understand the asynchronous setting. First, we characterise the realisable specifications through a property called the *residual property*, then we show it is decidable in PSpace.

Residual property. Let $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$ be an n-sequential transducer on $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$. Intuitively, the residual property says that if on some input prefix u, two sequential transducers of the union disagree on their outputs, i.e. produce different outputs, then a synchronous realiser of $[\![\mathcal{T}]\!]$ must "drop" one of the two transducers. However, it must do so while preserving the residual domain $u^{-1}\mathrm{dom}(\mathcal{T})$, i.e., the realiser must still accept any word of $u^{-1}\mathrm{dom}(\mathcal{T})$. For example, consider again Fig. 1 and the specification defined by $\mathcal{D}_1 \uplus \mathcal{D}_2$. On input a, the two transducers disagree, hence, since we want a synchronous realiser, a choice has to be made and therefore one of the two transducers must be dropped. However, by doing so, the residual domain will not be fully covered by the remaining transducer. For example, if a realiser chooses to output a when reading a, the residual language b^* is not covered anymore. As a matter of fact, $\mathcal{D}_1 \uplus \mathcal{D}_2$ is not realisable by any sequential and synchronous transducer.

Formally, let $u \in \Sigma_{\mathtt{i}}^*$ and let r_i, r_j be runs of some $\mathcal{D}_i, \mathcal{D}_j$ respectively, on u. We say that r_i and r_j agree on their output if $\mathrm{out}(r_i) = \mathrm{out}(r_j)$. Now, u is called *smooth* if every $\mathcal{D}_i$ admits an initial run on input u, and all these runs agree on the corresponding output. The word u is called *critical* if it is not smooth.

We say that $\mathcal{T}$ satisfies the *residual property* if for every critical prefix $u \in \Sigma_{\mathtt{i}}^*$ of a word of $\mathrm{dom}(\mathcal{T})$, there exists a subset $P \subsetneq \{1, \dots, n\}$ satisfying:

1. All the transducers $\mathcal{D}_i$, $i \in P$, produce the same output $\phi(u)$ on u;
2. $u^{-1}\mathrm{dom}(\mathcal{T}) = \bigcup_{i \in P} u^{-1}\mathrm{dom}(\mathcal{D}_i)$;
3. $\biguplus_{i \in P}(u, \phi(u))^{-1}[\![\mathcal{D}_i]\!]$ is realisable by a synchronous and sequential transducer.

▶ **Theorem 1.** *A specification S defined by a synchronous multi-sequential transducer $\mathcal{T}$ is realisable by a synchronous sequential transducer iff $\mathcal{T}$ satisfies the residual property.*

Sketch. If $[\![\mathcal{T}]\!]$ is realised by a synchronous sequential transducer $\mathcal{U}$, for every critical prefix u, let P be the set of i such that $\mathcal{D}_i$ and $\mathcal{U}$ map the same output to u. Property 1 is satisfied by definition, and the other two follow from the fact that $\mathcal{U}$ is sequential and realises $[\![\mathcal{T}]\!]$.

Conversely, if the residual property is satisfied, we can construct a synchronous and sequential realiser. The idea is to make a synchronised product of all the transducers $\mathcal{D}_i$, and, whenever on some input symbol σ at least two of them disagree on the output, we know by the residual property that there exists a subset P of them having the good properties $1, 2, 3$. Then, the realiser just goes on simulating all the transducers $\mathcal{D}_i$ corresponding to P in parallel.

It also shows that if the property is satisfied, then we can synthesise a realiser, which might however be exponentially larger than $\mathcal{T}$. ◀

▶ **Theorem 2.** *The realisability problem of synchronous multi-sequential specifications by synchronous sequential transducers is PSpace-complete.*

Sketch. The PSpace-hardness is obtained by reducing the problem from the emptiness problem of the intersection of n DFAs (*cf* Section 2 p. 6).

To show membership to PSpace, given a transducer $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$, we show that the residual property can be tested by a non-deterministic algorithm running in polynomial space. First, we bound the size of witnesses of the negation of the property: roughly, if there is such witness, namely a critical prefix u, then there exists a critical prefix v of exponential length (in $\mathcal{T}$) such that for any subset $P \subsetneq \{1, \dots, n\}$, one of the conditions $1, 2, 3$ is falsified. Then, the algorithm guesses the prefix v on the fly, simulating all transducers $\mathcal{D}_i$ in parallel and keeping their states in memory (it also needs a counter for the length of v). As soon as the transducers disagree on an output symbol, for each subset $P \subsetneq \{1, \dots, n\}$ (they can obviously be enumerated using only polynomial space), the algorithm checks whether property 1, 2 or 3 is falsified. Checking property 1 is easy: it suffices to look at the symbols produced when reading the last input symbol. Checking property 2 can be done using the current set of states reached by the transducers on input v, and by using any PSpace algorithm for automata inclusion. Finally, to check property 3, it suffices to recursively apply the PSpace algorithm described so far on a smaller set of transducers. The stack of recursive calls is linear in n, hence the memory used by the whole procedure remains polynomial. ◀

4 The synthesis game

We now define a 2-player safety game from a transducer $\mathcal{T}$ such that if Eve wins the game then $\mathcal{T}$ is realisable by a sequential transducer. This game notion will prove useful to show the correctness of the characterisation of Theorem 5, and may also be used as a practical way to synthesise implementations, as winning strategies of this game. In the asynchronous setting, two different runs of a transducer on the same input word may not only produce different outputs, but also the same output at different rates (i.e. one run is ahead, output-wise, of the other for some time). This leads us to the notion of delays, a classical tool to compare outputs in transducer theory. Let us define this notion formally.

Delays. Given two words $u_1, u_2 \in \Sigma^*$, their longest common prefix ℓ is denoted by $u_1 \wedge u_2$. The *delay* between u_1 and u_2 is an element of $\Sigma^* \times \Sigma^*$ defined by $\mathrm{del}(u_1, u_2) = (\ell^{-1}u_1, \ell^{-1}u_2)$. Intuitively, if a transducer produces u_1 and another one produces u_2, then $u_1 \wedge u_2$ is what can safely be output by the two transducers and $\mathrm{del}(u_1, u_2)$ what remains to be produced by each of them respectively. This notion is naturally extended to tuples of words: $\mathrm{del}(u_1, \ldots, u_n) = (\ell^{-1}u_1, \ldots, \ell^{-1}u_n)$ where $\ell = \bigwedge_{i=1}^{n} u_i$.

We now introduce notations that are useful when comparing the outputs of different runs on the same input of a transducer $\mathcal{T} = (\mathcal{A}, \rho, \tau)$ over $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$ with $\mathcal{A} = (\Sigma_{\mathtt{i}}, Q, I, F, \Delta)$. Given a pair $(q, w) \in Q \times \Sigma_{\mathtt{o}}^*$, where w is intended to be some delay associated with state q, given a transition $t = (q, \sigma, q') \in \Delta$ and some output word u prefix of $w\rho(t)$, we denote by $\mathit{next}((q, w), t, u)$ the "next" pair (state,delay), assuming that u is output, i.e. $\mathit{next}((q, w), t, u) = (q', u^{-1}w\rho(t))$. More generally, given a (total) function $D : Q \to 2^{\Sigma_{\mathtt{o}}^*}$ associating each state with a set of delays, we let $\mathit{live}(D) = \{q \in Q \mid D(q) \neq \varnothing\}$. For $\sigma \in \Sigma_{\mathtt{i}}$, $\mathit{next}(D, \sigma)$ maps every state which can be reached from $\mathrm{dom}(D)$ by reading σ to the corresponding delays obtained by outputting the longest common prefix of the words that can be formed from the previous delays and the output on these transitions. Formally, we call *safe output of D for σ* the word $\ell = \bigwedge\{w\rho(t) \mid q \in \mathit{live}(D), w \in D(q), t = (q, \sigma, q') \in \Delta\}$. Then $\mathit{next}(D, \sigma) = \{\mathit{next}((q, w), t, \ell) \mid q \in \mathit{live}(D), w \in D(q), t = (q, \sigma, q') \in \Delta\}$.

The synthesis game. In the synchronous setting, synthesis problems are classically solved by reduction to two-player games in which the players alternately choose one input symbol (the adversary, whom we call Adam) and one output symbol (the protagonist, called Eve). Their interaction induces a pair of input and output words by concatenating their respective symbols, and the protagonist wins if such pair satisfies the specification, or if the input word is out of the domain. Then, a finite-memory winning strategy in the game corresponds to an implementation of the specification.

In the asynchronous setting, the protagonist may choose arbitrary output words at each round instead of a single symbol, and one needs to introduce output delays in the game in order to define the winning condition in a regular manner. The game we now present follows this idea. Given a transducer $\mathcal{T} = (\mathcal{A}, \rho, \tau)$ with $\mathcal{A} = (\Sigma_{\mathtt{i}}, Q, I, F, \Delta)$, $\rho : \Delta \to \Sigma_{\mathtt{o}}^*$ and $\tau : F \to \Sigma_{\mathtt{o}}^*$, we build a two-player safety game $G_{\mathcal{T}} = (V_\forall, V_\exists, A_\forall, A_\exists, T_\forall, T_\exists, \mathsf{Safe})$, called the *synthesis game*, whose vertices keep track of the runs in $\mathcal{T}$ and the associated delays. More precisely, it consists of two disjoint sets of vertices $V_\forall = 2^Q \times (Q \to 2^{\Sigma_{\mathtt{o}}^*})$ and $V_\exists = V_\forall \times \Sigma_{\mathtt{i}}$, respectively controlled by Adam and Eve. The initial vertex is $v_0 = (I, D_0) \in V_\forall$ where $D_0(q) = \varnothing$ if $q \notin I$, and $D_0(q) = \{\epsilon\}$ otherwise.

Eve's vertices are Adam's vertices extended with the last input symbol picked by Adam. Suppose now that the game has been played for some rounds and is currently in some vertex (C, D) of Adam. Along these rounds, Adam has chosen a sequence u of input symbols, and Eve has chosen a set of runs over u from the initial states. C is the set of states in which these runs end. Each run induces some delays compared to the longest common prefix of all the outputs they can produce. D maps each state to the delays of the runs ending in it. Eve's actions consist in selecting some of these runs to prevent some delays to grow too high, i.e., she can drop from any set $D(q)$ some of its elements. By restricting the set of possible runs, Eve can be in a situation where some state q of C is accepting while none of the states of $\mathit{live}(D)$ is, in which case she loses, as none of the runs she has selected accepts the input word chosen by Adam. Such vertices constitute the set of *unsafe* vertices she needs to avoid.

More precisely, the set of Adam's *transitions* $T_\forall$ and Eve's transitions $T_\exists$ are defined as follows. From any game position (C, D), Adam can pick a symbol $\sigma \in \Sigma_{\mathtt{i}}$ and the game evolves to the position (C, D, σ). From (C, D, σ), Eve's actions is a subset $\alpha \subseteq \mathit{next}(D, \sigma)$

(she can "drop" some pairs of $next(D, \sigma)$), and the game evolves to (C', D_α) where C' is the set of states reached from C by reading σ, and D_α maps any $q \in Q$ to the set $\{w \mid (q, w) \in \alpha\}$.

Given $K \in \mathbb{N}$, we define the *K-synthesis game* as the restriction of $G_{\mathcal{T}}$ to delays of length at most K: $G_{\mathcal{T},K} = (V_\forall^K, V_\exists^K, v_0^K, A_\forall^K, A_\exists^K, T_\forall^K, T_\exists^K, \mathsf{Safe}^K)$, where $V_\forall^K = 2^Q \times (Q \to 2^{\Sigma_\mathfrak{o}^{\leq K}})$, $A_\exists \subseteq Q \times \Sigma_\mathfrak{o}^{\leq K}$, etc. There is no deadlock in $G_{\mathcal{T},K}$ because Eve can always play $\varnothing$, at the risk of going to an unsafe position.

Example. First, note that by definition of the game, any reachable vertex (C, D) or (C, D, σ) satisfies $live(D) \subseteq C$. Figure 2 represents the 1-synthesis game for $\mathcal{D}_1 \cup \mathcal{D}_2$ (*cf* Figure 1). The states depicted in a vertex correspond to C, together with their values by D (thanks to the previous remark, there is no need to represent the values D assigns for the states outside C). The circle vertices are Eve's positions, whose labels are not depicted, as they are just the label of their predecessor vertex extended with Adam's action. Bold nodes correspond to the unsafe states. Let us now explain how the game proceeds in more detail. First, since both $\mathcal{D}_1$ and $\mathcal{D}_2$ are complete and sequential, for each state (C, D) of Adam, C contains exactly one state of $\mathcal{D}_1$ and one state of $\mathcal{D}_2$. Eve's actions in the synthesis game, which consist in dropping a subset of pairs (state,delay), actually correspond here to "dropping" one of the two sequential transducers: at any moment, she can choose to drop $\mathcal{D}_2$, which leads her into the red part of the game, or to drop $\mathcal{D}_1$, which leads her into the blue part of the game. Note that once she has dropped one of the transducers, she is stuck in the corresponding part.

The initial vertex, owned by Adam, corresponds to being in the initial states of both $\mathcal{D}_1$ and $\mathcal{D}_2$, with no delays. If Adam chooses to play a as the first input letter, Eve has four choices. Either she keeps both transducers, with a delay of length 1, or she drops one of them, or both (not depicted). If Eve chooses to drop $\mathcal{D}_2$, respectively $\mathcal{D}_1$, Adam can then play a b, respectively an a, which leads her into an unsafe state. Note that this proves that Eve cannot win the 0-synthesis game corresponding to $\mathcal{D}_1 \cup \mathcal{D}_2$. If Eve keeps both, she has to drop one of them once Adam plays a second letter, since otherwise the delay would grow larger than 1. However, in both cases she has a move which ensures her a win: if Adam plays a second a, Eve can safely drop $\mathcal{D}_2$ since the accepting state of $\mathcal{D}_1$ has been reached, and if Adam plays b, Eve can safely drop $\mathcal{D}_1$ since the accepting state of $\mathcal{D}_2$ has been reached. If Adam chooses to play a b in the first place, Eve can immediately drop $\mathcal{D}_1$ and win. Hence, Eve wins the 1-synthesis game associated to $\mathcal{D}_1 \cup \mathcal{D}_2$. The described strategy then directly induces a sequential transducer realising the specification.

▶ **Proposition 3.** *Let S be a specification defined by some transducer $\mathcal{T}$. If Eve wins the K-synthesis game $G_{\mathcal{T},K}$ for some K, then S is realisable by a sequential transducer.*

Sketch. If Eve wins the K-synthesis game, then, since it is a safety game, she can win with a *positional* strategy. Thus, her actions only depend on the last visited vertex. This allows to reconstruct a realiser for S, whose states are the possible vertices of Adam visited by the strategy. Then, when Adam chooses an input symbol σ in a vertex (C, D) and Eve decides to go to some vertex (E, F) from (C, D, σ), then in the realiser, we add a transition from (C, D) to (E, F) on σ, outputting the safe output of D for σ. ◀

Synthesis algorithm. It is worth noting that the synthesis game allows for a synthesis procedure: for ascending values of K, test whether Eve wins the K-synthesis game (this can be done in PTime in the size of the game). If it is the case, then by Proposition 3 the specification is realisable, and we can even extract an implementation corresponding to a winning strategy of Eve. If it is not the case, then increment K and try again, until K

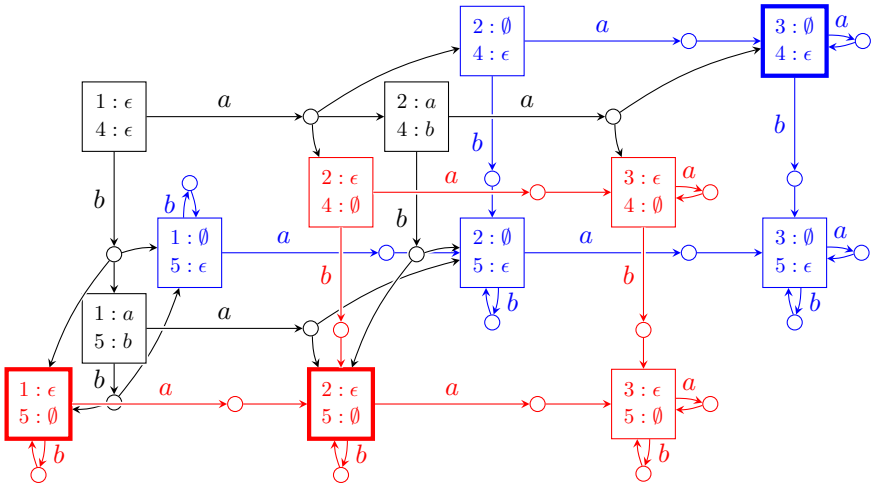

Figure 2 The 1-synthesis game corresponding to the union of $\mathcal{D}_1$ and $\mathcal{D}_2$ (*cf* Figure 1).

reaches some given upper bound B. The K-synthesis game is exponentially large in general (in the transducer defining the specification, and in K). Solving this game efficiently, using for instance symbolic methods, as done for LTL synthesis in the synchronous case [10, 13], is beyond the scope of this paper, but is an interesting research direction.

This algorithm is not complete in general: it is shown for instance in [12] that some specifications defined by transducers are realisable by sequential transducers while Eve has no winning strategy in $G_{\mathcal{T},K}$ for any K. Still, the converse of Proposition 3 holds for some subclasses of specifications $[\![\mathcal{T}]\!]$. For example, in the synchronous setting, where we want to synthesise a synchronous sequential transducer, it suffices to take $K = 0$. This gives an ExpTime procedure to check the realisability of $[\![\mathcal{T}]\!]$ by a synchronous sequential transducer. If $\mathcal{T}$ is *finite-valued*, then by taking K large enough (triply exponential in $\mathcal{T}$), we get completeness [12]. Finally, if $\mathcal{T}$ is functional, then Eve wins $G_{\mathcal{T},K}$ for some K iff $\mathcal{T}$ is equivalent to a sequential transducer, and a polynomial K (in $\mathcal{T}$) suffices [2].

In this paper, we obtain completeness for multi-sequential specifications by taking K exponential in $\mathcal{T}$ (Proposition 6). While this allows us to decide realisability using the game approach, the time complexity will not be optimal (2ExpTime). We indeed devise, in Section 5, a PSpace realisability-checking procedure based on an effective characterisation of realisable multi-sequential specifications. If the PSpace procedure concludes that the specification is realisable, one can run the former game-solving procedure to synthesise a realiser, for ascending values of K. This way, one may hope to synthesise a "small" realiser.

5 The asynchronous setting

We first characterise recursively the multi-sequential specifications which are sequentially realisable (Theorem 5). Then, we provide an equivalent characterisation, non-recursive and easier to check algorithmically, but more technical.

Similarly to the synchronous case, we define a notion of critical situation to which a realiser must react. In the synchronous case, it was just a prefix on which at least two sequential transducers were producing different outputs. In the asynchronous case, two sequential transducers may produce different outputs on the same prefix, but this may not be problematic in the case where one is ahead of the other, i.e., the output of one run is

a prefix of the output of the other. A critical situation is rather a prefix where the delays between all the outputs of the sequential transducers are too large. Since no bound is known a priori to define "too large", we formalise a critical situation as a prefix of the form uv, such that at least two sequential transducers loop on v, and have a different delay before and after the loop. By iterating this loop, i.e. by taking a prefix uv^n, the delay between these two transducers will grow unboundedly when n increases. For such loops, the situation will get critical if a realiser does not react.

▶ **Definition 4** (critical loop). Let $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$ be an n-sequential transducer. A *critical loop* for $\mathcal{T}$ is a triple $(u, v, \mathcal{X}) \in \Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{i}}^* \times 2^{\{1,\dots,n\}}$ such that

1. for all $i \in \mathcal{X}$, there exists an initial run $p_i \xrightarrow{u|\alpha_i} q_i \xrightarrow{v|\beta_i} q_i$ of $\mathcal{D}_i$ on uv;
2. for all $i \in \{1, \dots, n\} \setminus \mathcal{X}$, there is no run of $\mathcal{D}_i$ on u;
3. There exists $i, j \in \mathcal{X}$ such that $\mathrm{del}(\alpha_i, \alpha_j) \neq \mathrm{del}(\alpha_i\beta_i, \alpha_j\beta_j)$.

Our characterisation echoes the one of the synchronous setting. It says that whenever there is a critical situation (a critical loop), a realiser must be able to drop some of the sequential transducers, in order to prevent the delays to grow unboundedly, while preserving the residual domain. Formally:

▶ **Theorem 5** (recursive characterisation). *Let $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$ be a multi-sequential transducer over $\Sigma_{\mathtt{i}}, \Sigma_{\mathtt{o}}$. Then $[\![\mathcal{T}]\!]$ is realisable by a sequential transducer, iff, for all critical loops $(u, v, \mathcal{X})$, there exists $\mathcal{Y} \subsetneq \mathcal{X}$ such that*

1. *$\forall i, j \in \mathcal{Y}$, $del(\alpha_i, \alpha_j) = del(\alpha_i\beta_i, \alpha_j\beta_j)$ (following the notations of Definition 4),*
2. *$u^{-1}dom(\mathcal{T}) = \bigcup_{i\in\mathcal{Y}} u^{-1}dom(\mathcal{D}_i)$,*
3. *$\bigcup_{i\in\mathcal{Y}} (u, \ell)^{-1}[\![\mathcal{D}_i]\!]$ is realisable by a sequential transducer, where $\ell = \bigwedge_{i\in\mathcal{X}} \alpha_i$.*

Sketch. $\Rightarrow$ Let $\mathcal{U}$ be a sequential transducer realising $[\![\mathcal{T}]\!]$. For every critical loop $(u, v, \mathcal{X})$ of $\mathcal{T}$, the corresponding set $\mathcal{Y}$ is obtained by getting rid of all the transducers that stray arbitrarily far from $\mathcal{U}$ on the input words of the form uv^*. Then, the first property is immediate, and the other two follow from the fact that $\mathcal{U}$ is sequential and realises $[\![\mathcal{T}]\!]$.

$\Leftarrow$ Conversely, assuming that whenever a critical loop is met there exists a set $\mathcal{Y}$ satisfying the three conditions, we prove by induction on the degree n of sequentiality of $[\![\mathcal{T}]\!]$ that Eve has a winning strategy in the $K_{\mathcal{T}}$-synthesis game for $\mathcal{T}$, for some well-chosen value $K_{\mathcal{T}}$ depending on $\mathcal{T}$. By Proposition 3, this entails the existence of a sequential realiser.

Note that in the synthesis game, since $\mathcal{T}$ is a union of sequential transducers, for each accessible vertex (C, D) of Adam, and for every $i \in \{1, \dots, n\}$, there is at most one state q_i of $\mathcal{D}_i$ occurring in $live(D)$, and if there exists such a state, $|D(q_i)| = 1$. As a consequence, Eve's actions in the synthesis game, which consist in dropping a subset of pairs (state,delay), actually correspond here to "dropping" a subset of sequential transducers.

If $n = 1$, then $\mathcal{T}$ is sequential, and the strategy of Eve that consists in never dropping $\mathcal{T}$ is winning. Now, suppose that $n > 1$. In order to demonstrate that Eve has a winning strategy, we show that for every input word chosen by Adam, either Eve can keep track of all the transducers in the $K_{\mathcal{T}}$-synthesis game, which ensures her a win, or she can drop some transducers on the way, while reaching a state from which she has a winning strategy.

Let $u \in \Sigma_{\mathtt{i}}^*$, and let (C_0, D_0) be the state reached by Eve on input u if she drops nothing. If (C_0, D_0) is not part of the $K_{\mathcal{T}}$-synthesis game, i.e., for some $q \in C_0$, $D_0(q) = \{w\}$ with $|w| > K_{\mathcal{T}}$, this implies the existence of a decomposition $u_1u_2u_3$ of u such that $(u_1, u_2, \mathcal{X})$ is a critical loop for some $\mathcal{X} \subseteq \{1, \dots, n\}$. Then, by hypothesis, there exists a subset $\mathcal{Y} \subsetneq \mathcal{X}$ which satisfies the three conditions of the theorem, hence $\mathcal{T}' = \biguplus_{i\in\mathcal{Y}} (u_1, \ell)^{-1}[\![\mathcal{D}_i]\!]$ is realisable by a sequential transducer. In particular, $\mathcal{T}'$ satisfies the conditions on critical loops (implication

$\Rightarrow$ shown before), and, by the induction hypothesis (since $\mathcal{T}'$ is $|\mathcal{Y}|$-sequential and $|\mathcal{Y}| < n$), Eve has a winning strategy in the $K_{\mathcal{T}'}$-synthesis game for $\mathcal{T}'$ from the initial vertex. Lifting this strategy to the $K_{\mathcal{T}}$-synthesis game for $\mathcal{T}$ yields a winning strategy for Eve from the state (C, D'), where (C, D) is the state reached by Eve on input u_1u_2 if she drops nothing, and D' is obtained from D by dropping all the transducers that are not part of $\mathcal{Y}$. ◀

The proof of Theorem 5 shows that if a multi-sequential specification is realisable, Eve wins the K-synthesis game for K computable from the specification, as stated in Proposition 6. As explained in Section 4, solving the k-synthesis game for ascending values of k then provides a practical way to synthesise a realiser, but the complexity is not optimal.

▶ **Proposition 6** (bounded delay). *Let S be a specification defined by an n-sequential transducer $\mathcal{T}$. Then S is realisable by some sequential transducer iff Eve wins the K-synthesis game for $K = L(6M)^{n^2}$, where L is the longest output occurring on a transition of $\mathcal{T}$, and M is the maximal number of states of a sequential transducer of $\mathcal{T}$.*

▶ **Theorem 7.** *A realisable specification S defined by a multi-sequential transducer $\mathcal{T}$ with m states admits a realiser of size doubly exponential in m. Moreover, there exists a family $(S_n)_{n\in\mathbb{N}}$ of realisable specifications such that for every $n \in \mathbb{N}$, S_n is definable by a multi-sequential transducer of size polynomial in n, and every sequential transducer realising S_n has a size that is doubly exponential in n.*

Proof. Let $S \subset \Sigma^* \times \Gamma^*$ be a realisable specification defined by an n-sequential transducer $\mathcal{T}$ with a set of states Q of size m. Note that $n \leq m$, hence, by Proposition 6, Eve wins the K-synthesis game for some K exponential in m. Then, the construction presented in the proof of Proposition 3 exposes a realiser whose set of states Q' consists of Adam's vertices that are reachable in the K-synthesis game. For every such vertex $(C, D) \in 2^Q \times (Q \to 2^{\Gamma^*})$, since $\mathcal{T}$ is n-sequential, there is at most n sates $q \in Q$ satisfying $D(q) \neq \emptyset$. Moreover, for every such state we have $D(q) = \{w\}$ for some $w \in \Gamma^*$ satisfying $|w| \leq K$. Therefore, the size of Q' is bounded by $2^m(m(|\Gamma|^{K+1}))^n$, which is doubly exponential in m.

In order to expose the family $(S_n)_{n\in\mathbb{N}}$, we use the notion of j-pairs, presented in [22]. For every $n \in \mathbb{N}$, let us consider the alphabet $I_n = \{1, \ldots, n\}$. A *bad j-pair* of a word $u = i_1 \ldots i_m \in I_n^*$ is a pair of positions $1 \leq k < k' \leq m$ such that $i_k = i_{k'} = j$, and for all $k < \ell < k'$, $i_\ell \leq j$. Then every $u \in I_n^*$ satisfying $|u| \geq 2^n$ admits a bad j-pair for some $1 \leq j \leq n$, and there exists a word, denoted by ψ_n, that has size $2^n - 1$, and contains no j-pair (see [22]). We now consider the finite alphabet $\Sigma = \{a, b\}$. For every $n \in \mathbb{N}$, let Σ_n denote the alphabet $I_n \times \Sigma$. We denote by $\pi_1 : \Sigma_n^* \to I_n^*$ and $\pi_2 : \Sigma_n^* \to \Sigma^*$ the projections on the first, respectively second component. Let $f : \Sigma_n^* \to \Sigma^*$ be the function mapping $w \in \Sigma_n^*$ to the word obtained by taking the last letter of $\pi_2(w)$ and putting it at the beginning, i.e., $f(w) = \sigma v$ where $\sigma \in \Sigma$ and $v \in \Sigma^*$ satisfy $\pi_2(w) = v\sigma$. We consider the specification

$$S_n = \{(w, f(w)) | w \in \Sigma_n^*\} \cup \{(w, \epsilon) | w \in \Sigma_n^* \text{ contains a bad } j\text{-pair for some } 1 \leq j \leq n\}.$$

Then S_n is definable by an $(n+2)$-sequential transducers with $3(n+2)$ states, since the function f is definable by the union of 2 sequential transducers of size 3, and for every $1 \leq j \leq n$, the set of words $w \in I_n^*$ containing a bad j-pair is recognisable by a deterministic automaton of size 3. Moreover, since every word $u \in I_n^*$ of size greater than 2^n admits a bad j-pair for some j, S_n is realised by the sequential transducer mapping every word $w \in \Sigma_n^*$ satisfying $|w| < 2^n$ to $f(w)$, and every $w \in \Sigma_n^*$ satisfying $|w| \geq 2^n$ to ϵ.

We now show that every sequential transducer $\mathcal{D}$ realising S_n has at least 2^{2^n-1} states. Let $\mathcal{D} = ((\Sigma, Q, I, F, \Delta), \rho, \tau)$ be a sequential transducer realising S_n. For every $v \in \Sigma^*$ such that $|v| = 2^n - 1$, let $\psi_v \in \Sigma_n^*$ denote the word satisfying $\pi_1(\psi_v) = \psi_n$ and $\pi_2(\psi_v) = v$. We

now show that for every pair of distinct words $v_1, v_2 \in \Sigma^*$ of size $2^n - 1$, the states reached by $\mathcal{D}$ on input ψ_{v_1} and ψ_{v_2} are distinct. This allows us to conclude the proof, since Σ^* contains 2^{2^n-1} such words. Given $v \in \Sigma^*$ satisfying $|v| = 2^n - 1$, let $\rho_v : p_0 \xrightarrow{\psi_v|v'} p_v$ denote the accepting run of $\mathcal{D}$ on input ψ_v. Then $v' = \epsilon$, since if the first letter of v' was an a, $\mathcal{D}$ would not be able to produce an acceptable output on input $\psi_v \cdot (1, b)$, and a similar contradiction would be reached if the first letter of v' was a b. Therefore, the output associated to ψ_v is produced by the terminal function of $\mathcal{D}$, i.e., $\tau(p_v) = f(\psi_v)$. Since f is injective, for every pair of distinct words $v_1, v_2 \in \Sigma^*$ of size $2^n - 1$, $p_{v_1} \neq p_{v_2}$. ◀

We are now ready to show how to decide the realisability of multi-sequential specifications in PSpace. Consider the characterisation given in Theorem 5. We rely on the notion of witness for the non-satisfaction of this characterisation, and we show how to decide the existence of a witness, using a reduction to the emptiness of reversal-bounded counter machines.

The notion of witness intuitively consists in the following ingredients: (1) an unfolding (modeled as a tree) of the recursive characterisation of Theorem 5 and (2) an explicit formulation of delay differences using simple properties of words. Formally, given an n-sequential transducer $\mathcal{T} = \biguplus_{i=1}^{n} \mathcal{D}_i$, where each $\mathcal{D}_i$ is sequential, a *witness* for $\mathcal{T}$ is a finite tree t whose nodes are labelled in $\Sigma_{\mathtt{i}}^* \times \Sigma_{\mathtt{i}}^* \times (2^{\{1,\dots,n\}} \setminus \{\varnothing\})$. For any node x of t, we denote by (u_x, v_x, S_x) its label. For all nodes x, y, z of t, it is required that:

1. (maximality) if x is the root, $S_x = \{1, \dots, n\}$;
2. (consistency) S_x can be split into two disjoint sets N_x, L_x such that for all $i \in N_x$ there is no run of $\mathcal{D}_i$ on u_x, and for all $i \in L_x$ there is a run of $\mathcal{D}_i$ on $u_x v_x$ from its initial state q_0^i, of the form $q_0^i \xrightarrow{u_x|\alpha_{x,i}} p_{x,i} \xrightarrow{v_x|\beta_{x,i}} p_{x,i}$;
3. (monotonicity) if y is a child of x, then $S_y \subsetneq L_x$ and u_x is a prefix of u_y;
4. (partition) if Y is the set of children of x, then $\{S_y \mid y \in Y\}$ partitions L_x;
5. (delays) if y and z are different children of x, for all $i \in S_y$ and $j \in S_z$, either $|\beta_{x,i}| \neq |\beta_{x,j}|$ or, $\beta_{x,i}\beta_{x,j} \neq \epsilon$ and, $\alpha_{x,i}$ and $\alpha_{x,j}$ mismatch[5];
6. (leaves) if x is a leaf, then there is $w \in \Sigma_{\mathtt{i}}^*$ such that $u_x w \in \mathrm{dom}(\mathcal{T})$ and $u_x w \notin \mathrm{dom}(\mathcal{D}_i)$ for all $i \in S_x$.

Intuitively, conditions 2 and 5 require that the words u_x, v_x are critical loops. The delay difference required in the definition of critical loops is not explicit here, but rather replaced by simple properties of words (condition 5), which are easier to check algorithmically. These properties are not strictly equivalent to delay difference, but up to iterating the loop on v_x a sufficient number of times, they are. Conditions $1, 3, 4$ correspond to properties of the subsets met when unfolding the recursive characterisation of Theorem 5. They also allow us to bound linearly the number of nodes of a witness. As announced, all these conditions characterise the unrealisable multi-sequential specifications:

▶ **Lemma 8.** *A multi-sequential specification defined by a trim transducer $\mathcal{T}$ is not realisable by a sequential transducer if and only if there exists a witness for $\mathcal{T}$.*

▶ **Theorem 9.** *The realisability problem by some sequential transducer of a specification defined by a multi-sequential transducer is PSpace-c.*

Sketch. PSpace-hardness has been shown in Section 2. To show PSpace-easyness, we reduce the problem to deciding the emptiness of the language of a counter machine, whose counters make at most 1 reversal (i.e. move from increasing to decreasing mode). This is known to

[5] Two words u, v mismatch if there is a position i such that $i \leq |u|, |v|$ and the ith letter of u differs from the ith letter of v.

be in NLogSpace [16]. Our machine is exponentially large (in the transducer defining the specification), but can be constructed on the fly, hence we get PSpace.

A bit more precisely, we first define the notion of *skeleton* s, which is a witness without the words u_x, v_x, hence there are finitely many skeletons, each one of polynomial size. Given an enumeration $x_1 \dots x_n$ in depth-first order of the nodes of s, we construct a counter machine M_s which recognises sequences of the form $x_1 w_{x_1} \# v_{x_1} \dots x_n w_{x_n} \# v_{x_n}$ such that if we extend any label of a node x in s with the pair of words $(w_{y_1} \dots w_{y_k}, v_x)$, where $y_1 \dots y_k$ is the path from the root to x, we get a witness. Hence, there exists a witness iff there exists a skeleton s such that $L(M_s)$ is non-empty. Our algorithm non-deterministically guesses a skeleton and runs a procedure to check in PSpace the emptiness of M_s.

Let us intuitively explain how M_s works. Conditions $1, 3, 4$ and 6 are regular, so no counter is needed there. Counters are only necessary to check Condition 5, for instance to compute the length of the words $\beta_{x,i}$, and to check the existence of a mismatch between a word $\alpha_{x,i}$ and a word $\alpha_{x,j}$. First, a mismatch position m is guessed, by incrementing for some time two counters $c_{i,x}$ and $c_{j,x}$ in parallel. Then, they are decremented according to the length of outputs produced by simulating the transitions of $\mathcal{D}_i$ and $\mathcal{D}_j$ respectively. When one of them reaches 0, say $c_{i,x}$, we store the mth symbol of the output of $\mathcal{D}_i$ on u_x in memory. We do the same for $c_{j,x}$ and later on check that the two stored symbols are different. ◀

6 Conclusion

We have identified a class of specifications (whose membership is decidable in PTime), for which the sequential realisability problem is PSpace-c, both in the asynchronous and synchronous settings. This is in contrast to the general case, which is ExpTime-c for synchronous specifications, and undecidable in the asynchronous case. Our procedure allows to synthesise a sequential transducer whenever the specification is realisable, and allows for incremental testing, via the solvability of a two-player game parameterised by the longest output allowed to be queued by a realiser before being output.

While the class of multi-sequential specifications is natural, as the closure of graphs of sequential functions under finite unions, we believe that it may also be interesting for practical applications. In particular, Vardi and Lustig have defined the concept of synthesis from component libraries [26], in the synchronous setting, over infinite words. In this setting, given a set of components (synchronous sequential transducers over finite words), a specification S over infinite words, the question is whether the components can be arranged in such a way which realises the specification (by linking the final states of the components to the initial state of another component). This problem was shown to be decidable. We would like to investigate another way of reusing existing components, which is tightly related to multi-sequential specifications: given components $C_1, \dots, C_n$ represented as sequential transducers and a specification S, decide whether there exists a sequential function f such that f and S have the same domain, $f \subseteq \bigcup_i C_i$ and f satisfies S. This is beyond the scope of this paper but we plan to investigate further this question in the near future.

References

1 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

2 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292(1):45–63, 2003.

3 Jean Berstel and Luc Boasson. Transductions and context-free languages. *Ed. Teubner*, pages 1–278, 1979.

4 Arnaud Carayol and Christof Löding. Uniformization in Automata Theory. In *Proceedings of the 14th Congress of Logic, Methodology and Philosophy of Science Nancy, July 19-26, 2011*, pages 153–178, London, 2014. College Publications.

5 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Information and Computation*, 208(6):677–693, 2010. doi:10.1016/j.ic.2009.07.004.

6 Christian Choffrut and Marcel Paul Schützenberger. Décomposition de fonctions rationnelles. In *2nd Annual Symposium on Theoretical Aspects of Computer Science, STACS*, pages 213–226, 1986.

7 Church, Alonzo. Logic, arithmetic and automata. In *International Congress of Mathematics*, pages 23–35, Stockholm, 1962.

8 Laure Daviaud, Ismaël Jecker, Pierre-Alain Reynier, and Didier Villevalois. Degree of sequentiality of weighted automata. In Javier Esparza and Andrzej S. Murawski, editors, *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures, FOSSACS 2017, Uppsala, Sweden, April 22-29*, pages 215–230. Springer Berlin Heidelberg, 2017. doi:10.1007/978-3-662-54458-7_13.

9 Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Symbolic algorithms for infinite-state games. In *Proceedings of the 12th International Conference in Concurrency Theory, CONCUR 2001, Aalborg, Denmark, August 20-25*, pages 536–550, 2001. doi:10.1007/3-540-44685-0_36.

10 Rüdiger Ehlers. Symbolic bounded synthesis. In *Proceedings of the 22nd International Conference on Computer Aided Verification, CAV 2010, Edinburgh, UK, July 15-19*, volume 6174 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2010.

11 Samuel Eilenberg. *Automata, Languages, and Machines*. Academic Press, 1974.

12 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, Rome, Italy*, pages 125:1–125:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.125.

13 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Exploiting structure in LTL synthesis. *International Journal on Software Tools for Technology Transfer*, 2011.

14 Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Antichains and compositional algorithms for LTL synthesis. *Formal Methods in System Design*, 39(3):261–296, 2011.

15 Wladimir Fridman, Christof Löding, and Martin Zimmermann. Degrees of lookahead in context-free infinite games. In *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, pages 264–276, 2011. doi:10.4230/LIPIcs.CSL.2011.264.

16 Eitan M. Gurari and Oscar H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Science*, 22(2):220–229, 1981. doi:10.1016/0022-0000(81)90028-3.

17 Michael Holtmann, Lukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. In C.-H. Luke Ong, editor, *Proceedings of the 13th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2010, Paphos, Cyprus, March 20-28*, volume 6014 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2010.

18 Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The first reactive synthesis competition (SYNTCOMP 2014). *STTT*, 19(3):367–390, 2017. doi:10.1007/s10009-016-0416-3.

19 Ismaël Jecker and Emmanuel Filiot. Multi-sequential word relations. In *Proceedings of the 19th International Conference on Developments in Language Theory, DLT 2015, Liverpool, UK, July 27-30*, pages 288–299, 2015. doi:10.1007/978-3-319-21500-6_23.

20 B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Computer Aided Verification, CAV*, pages 258–262, 2007.

21 J.R. Büchi and L.H. Landweber. Solving sequential conditions finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

22 Felix Klein and Martin Zimmermann. How much lookahead is needed to win infinite games? *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:4)2016.

23 Felix Klein and Martin Zimmermann. Prompt delay. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, Chennai, India*, pages 43:1–43:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.43.

24 Kojiro Kobayashi. Classification of formal languages by functional binary transductions. *Information and Control*, 15(1):95–109, 1969.

25 Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266. IEEE Computer Society, 1977. URL: http://dblp.uni-trier.de/db/conf/focs/focs77.html#Kozen77.

26 Yoad Lustig and Moshe Y. Vardi. Synthesis from component libraries. *STTT*, 15(5-6):603–618, 2013.

27 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM Symposium on Principles of Programming Languages, POPL*. ACM, 1989.

28 Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer Berlin Heidelberg, 2007.

29 Wolfgang Thomas. Church's problem and a tour through automata theory. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 635–655. Springer, 2008.

30 Martin Zimmermann. Delay games with WMSO+U winning conditions. *RAIRO - Theoretical Informatics and Applications*, 50(2):145–165, 2016. doi:10.1051/ita/2016018.

31 Martin Zimmermann. Finite-state strategies in delay games. In *Proceedings 8th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017, Roma, Italy, 20-22 September*, pages 151–165, 2017. doi:10.4204/EPTCS.256.11.

32 Martin Zimmermann. Games with costs and delays. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005125.