Sequential Relational Decomposition*

Dror Fried
Department of Computer Science
Rice University, USA
dror.fried@rice.edu

Axel Legay
Inria, France
axel.legay@inria.fr

Joël Ouaknine
Max Planck Institute for Software Systems
Saarland Informatics Campus, Germany
Department of Computer Science
Oxford University, UK
joel@mpi-sws.org

Moshe Y. Vardi
Department of Computer Science
Rice University, USA
vardi@cs.rice.edu

Abstract

The concept of *decomposition* in computer science and engineering is considered a fundamental component of *computational thinking* and is prevalent in design of algorithms, software construction, hardware design, and more. We propose a simple and natural formalization of *sequential decomposition*, in which a task is decomposed into two sequential sub-tasks, with the first sub-task to be executed out before the second sub-task is executed. These tasks are specified by means of input/output relations. We define and study *decomposition problems*, which is to decide whether a given specification can be sequentially decomposed. Our main result is that decomposition itself is a difficult computational problem. More specifically, we study decomposition problems in three settings: where the input task is specified explicitly, by means of Boolean circuits, and by means of automatic relations. We show that in the first setting decomposition is NP-complete, in the second setting it is NEXPTIME-complete, and in the third setting there is evidence to suggest that it is undecidable. Our results indicate that the intuitive idea of decomposition as a system-design approach requires further investigation. In particular, we show that adding human to the loop by asking for a decomposition hint lowers the complexity of decomposition problems considerably.

*Full version of the paper appears in [9].

LICS '18, July 9–12, 2018, Oxford, United Kingdom

ACM ISBN 978-1-4503-5583-4/18/07…$15.00
https://doi.org/10.1145/3209108.3209203

1 Introduction

Over the past decade, it became apparent that the conceptual way of analyzing problems through computer-science techniques can be considered as a general approach to analysis, design, and problem solving, known as *computational thinking* [31, 32]. A key element of this approach is the taming of complexity by decomposing a complex problem into simpler problems. Quoting Wing [31]: "Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system." While abstraction helps to tame complexity by simplifying away irrelevant details of a complex problem, decomposition helps complexity by breaking down complex problems into simpler ones. In fact, decomposition is a generic project-management technique: project managers quite often face challenges that require decomposition – a large project that is divided among team members to make the problem less daunting and easier to solve as a set of smaller tasks, where team members work on tasks that are in their specific fields of expertise. As computer scientists and engineers, the concept of decomposition is prevalent in the design of algorithms, in software construction, in hardware design, and so on. For example, a classical paper in software engineering studies criteria to be used in decomposing systems into modules [24]. Yet, in spite of the centrality of the concept of decomposition in computational thinking, it has yet to be studied formally in a general theoretical setting (see related work). Such a study is the focus of this work.

There are many different types of decomposition that can be considered. Based on her understanding of the problem, the *decomposer* has to make a decision on how to decompose a given problem, for example, by meeting certain constraints on the size of the sub-problems, or constraints on the way that solved sub-problems ought to be recomposed. A simple and natural way of decomposition is *sequential* decomposition in which a task is decomposed into two sub-tasks, where the first sub-task

is to be carried out before the second sub-task can be executed. A formal model for sequential decomposition is the subject of this work. We assume that the given problem is specified by means of an input/output relation. It is widely accepted that such relations are the most general way to specify programs, whether for terminating programs [13], where input and output should be related values, or for nonterminating programs [25], where input and output are streams of values. The *decomposition problem* is to decompose a given input/output relation R, between an input domain $\mathcal{I}$ and an output domain $\mathcal{O}$, into two relations R_1 and R_2, such that R can be reconstructed from R_1 and R_2 using relational composition (defined in Section 3). To avoid trivial solutions, where either R_1 or R_2 is the identity relation, we assume that the intermediate domain, that is, the co-domain of R_1, which is also the domain of R_2, is specified. Intuitively, specifying the intermediate domain amounts to constraining the manner in which the first task can "communicate" with the second task. Such a restriction can be viewed as a form of *information hiding*, which is one of the major criteria for decomposition in [24]. As we show, sequential decomposition is nontrivial only when the channel of communication between the first and second task has a small "bandwidth", appropriately defined.

We study sequential decomposition in three settings: *explicit*, *symbolic*, and *automatic*. In the explicit setting, the input/output relation R is specified explicitly. In the symbolic setting, the domains and R are finite but too large to be specified explicitly, so R is specified symbolically as a Boolean circuit. In the automatic setting, the domains and R may be infinite, so the domains are specified by means of an alphabet, over which R is specified by means of a deterministic finite-state automaton.

Our general finding is that sequential decomposition, viewed as a computational problem, is itself a challenging problem. In the explicit setting, the decomposition problem is NP-complete. This escalates to NEXPTIME-complete for the symbolic setting. For the automatic setting the decomposition problem is still open, but we provide evidence and conjecture that it is undecidable. Specifically, we show that even a very simple variant of the automatic setting can be viewed as equivalent to the Positivity Problem, whose decidability is well known to be open [21, 29]. We do show, however, that a "strategic" variant of the automatic setting, in which the required relations are described as transducers is in EXPTIME. These findings, that decomposition is an intractable problem, can be viewed as a "No-Free-Lunch" result, as it says that decomposition, which is a tool to combat complexity, is itself challenged by computational complexity. This means that while decomposition is an essential tool, the application of decomposition is an art, rather than science, and requires human intuition.

As such, we explore decomposition with "a human in the loop", where the role of the human is to offer a hint, suggesting one of the terms of the decomposition, and the role of the decomposition algorithm is to check if a valid decomposition can be found based on the hint. We show that decomposition with a hint is easier than decomposition with no hint; it is in PTIME in the explicit setting, in Π_3^P in the symbolic setting, and in EXPTIME in the automatic setting.

2 Related work

In this paper we introduce a new framework to study *decomposition* of system-level specifications into component-level specifications. In contrast, most existing approaches in software engineering focus on *composition*, that is, developing systems from independently developed components, or proving system-level properties from component-level properties, for example [8]. This compositional approach is a fundamental approach in computer science to taming complexity.

A major weakness of composition-based approaches to system development is that they consider only the problem of simplifying the implementation work, but ignore the task of decomposing the often very complex system-level technical specifications into smaller/simpler ones. For example, there is no technique to explain how smaller assumption/guarantee contracts can be obtained from larger ones. This operation has to be conducted manually by developers using their intuition and understanding of complex system-level specification. Thus, our work here on decomposition complements existing approaches on composition-based development. In addition, we believe that our work is also relevant to *architectural design*, e.g., as a complement of [4].

A classical paper of Parnas in software engineering studies criteria for to be used in decomposing systems into modules [24]. Parnas's framework, however, assumes that the starting point for decomposition consists of architectural specification of the system, while our starting point is quite more abstract, as we assume that we are provided with relational specification. Another related work is that of [27], which describe an approach for extracting sequential components from system specification. Unlike, however, our work here, which starts from a highly abstract relational specification, the approach of [27] assumes that the system's specification is provided by means of interface specification of the components. Thus, this approach is of factorization rather then decomposition.

Decomposition has been studied in the context of linear algebra. A matrix decomposition or matrix factorization is a factorization of a matrix into a product of matrices. There are many different matrix factorizations. Certain Boolean matrix-factorization problems are known to be NP-complete [15]. Our NP-completeness result for explicit relations can be viewed as a special case of Boolean matrix-factorization. In addition, our formulation for the explicit case can be viewed as a reformulation of the combinatorial definition of the nondeterministic communication complexity (see Chapters 1-2 in [19]). In that sense, this paper extends these works to more general representations of relations.

3 Preliminaries

3.0.1 Relations

Let A, B, C be finite sets. For a binary relation $R \subseteq A \times B$, let $Dom(R)$, and $Img(R)$ be the domain of R, and the image (sometimes called co-domain) of R, defined as follows. $Dom(R) = \{a \in A | \exists b \in B \text{ s.t } (a, b) \in R)\}$, and $Img(R) = \{b \in B | \exists a \in A \text{ s.t } (a, b) \in R)\}$. For $a \in A$, let $Img_R(a) = \{b \in B | (a, b) \in R\}$. R is called a function if for every $a \in A, b, b' \in B$ we have $(a, b), (a, b') \in R \implies b = b'$. Given binary relations $R_1 \subseteq A \times B$, and $R_2 \subseteq B \times C$, the *composition* of R_1 and R_2 is a binary relation $R_1 \circ R_2 \subseteq A \times C$ where $R_1 \circ R_2 = \{(a, c) | \exists b \in B \text{ s.t. } (a, b) \in R_1 \text{ and } (b, c) \in R_2\}$.

3.0.2 Automata

A Nondeterministic Finite Automaton (NFA) is a tuple $\mathcal{A} = (\Sigma, Q, q, \delta, F)$, where Σ is a finite alphabet, Q is a finite state set with an initial state q, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and F is an accepting-state set. A run of $\mathcal{A}$ over a word $w = a_1 a_2 \cdots a_n$ for some n is a state sequence $r = q_0, q_1, \cdots, q_n$ such that $q_{i+1} \in \delta(q_i, a_i)$ for $i \geq 0$, where $q_0 = q$ is the initial state. A run is *accepting* if its final state is accepting. A word is *accepted* if it has an accepting run. The language of $\mathcal{A}$, $L(\mathcal{A})$, is the set of all accepted words of $\mathcal{A}$ and is called a *regular language*. A language is also regular if and only if it can be described by a *regular expression*. We define the size of the automaton $\mathcal{A}$ as $|Q| + |\Sigma| + |\delta|$ and denote this size by $|\mathcal{A}|$. For NFAs $\mathcal{A}_1 = (\Sigma_1, Q_1, q^1, \delta_1, F_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, q^2, \delta_2, F_2)$, we define the *product automaton* of $\mathcal{A}_1$ and $\mathcal{A}_2$ as an automaton $\mathcal{A}_1 \times \mathcal{A}_2 = (\Sigma_1 \times \Sigma_2, Q_1 \times Q_2, q^1 \times q^2, \delta, F_1 \times F_2)$ where $(p, p') \in \delta((q, q'), (l, l'))$ iff $p \in \delta_1(q, l)$ and $p' \in \delta_2(q', l')$. An NFA $\mathcal{A}$ is deterministic (called DFA) if for every state q and letter a, $|\delta(q, a)| \leq 1$. Every NFA can be determinized to a DFA that describes the same language by using the *subset construction*, possibly with an exponential blow-up [14]. It is often more convenient for users to specify regular languages by means of regular expressions, which can be converted to DFA, possibly with a doubly exponential blow-up [14]. We assume here that all regular languages are specified by means of DFAs, as we wish to study the inherent complexity of decomposition.

Finally a *transducer* (also called a Moore machine) is a DFA with no accepting states, but with additional output alphabet and an additional function from the set of states to the output alphabet. Transducers describe automatic functions from input to output.

4 Problem definition

The concept of decompositions that we explore here is related to systems that can be defined by their given input and produced output. We model these as an input domain $\mathcal{I}$ and an output domain $\mathcal{O}$, not necessarily finite. Our description of a system is a specification that associates inputs to outputs, and is modeled as a relation $R \subseteq \mathcal{I} \times \mathcal{O}$ [13, 25]. In addition we assume a constraint in form of a domain with a specific size that directs the decomposition to be more concise and is given as an intermediate domain $\mathcal{B}$. The objective is to decompose R into relations $R_1 \subseteq \mathcal{I} \times \mathcal{B}$, and $R_2 \subseteq \mathcal{B} \times \mathcal{O}$ such that the composition $R_1 \circ R_2$ has either of the following properties: (i) no input-output association is added or lost - this problem is called the *Total Decomposition Problem (TDP)*, or (ii) a more relaxed version, called the *Partial Decomposition Problem (PDP)* in which no input-output association is added, but we are allowed to lose some of the output as long as each input can be resolved.

To make the paper more fluent to read we use the notation TDP/PDP for statements that are valid to the TDP and the PDP variants respectively. Since the size of the intermediate domain $\mathcal{B}$ can be significantly smaller than the size of the input or output domains, the problem becomes non-trivial as some sort of compression is required in order to enable TDP/PDP. The actual problems of TDP/PDP are appropriately defined for each section as decision problems. We first define the TDP/PDP conditions as follows.

Definition 4.1. (TDP/PDP conditions) Given binary relations $R \subseteq \mathcal{I} \times \mathcal{O}$, $R_1 \subseteq \mathcal{I} \times \mathcal{B}$, and $R_2 \subseteq \mathcal{B} \times \mathcal{O}$ for some domains $\mathcal{I}, \mathcal{O}, \mathcal{B}$, we say that (R_1, R_2) meet the TDP condition if $Img(R_1) \subseteq Dom(R_2)$ and $R_1 \circ R_2 = R$. We say that (R_1, R_2) meet the PDP condition if $Img(R_1) \subseteq Dom(R_2)$, $Dom(R_1 \circ R_2) = Dom(R)$, and $R_1 \circ R_2 \subseteq R$.

The decision problem of TDP/PDP, formally defined for diverse settings, is: given a description of domains and a relevant relation R, find whether there exist (R_1, R_2) that meet the TDP/PDP condition.

Since the type of domains and relations that we explore varies between an explicit and a more implicit

description of relations, the formal definitions of the problem change according to these representations, and so are the sought decomposed relations. It is important to note that the TDP/ PDP conditions are properties of the actual relations, and not of the description in which the relations are represented.

Note that PDP without the restriction of $Dom(R_1 \circ R_2) = Dom(R)$ becomes trivial, as one can take the empty sets as R_1 and R_2. Also note that $Img(R_1) \subseteq Dom(R_2)$ implies that $Dom(R_1 \circ R_2) = Dom(R_1)$ (see proper claim in [9]), a technical fact that can help prove TDP/PDP conditions in various settings. Finally see that the decomposed relations that meet the TDP conditions, also meet the PDP conditions on the same input, therefore a positive answer to TDP implies a positive answer to PDP.

5 Decomposition is hard

Decomposition has been advocated as the first step in the design of complex systems, with the intuition that it is easier to design components separately, rather than design a complex monolithic system. We show in this section several settings in which finding a sequential decomposition is computationally hard. This means that while sequential decomposition could be used to simplify the complexity of the initial specification, such decomposition itself is intractable, thus can be viewed as a "No-Free-Lunch".

5.1 Explicit relations

The simplest case of decomposition is when the domains $\mathcal{I}, \mathcal{O}$ and $\mathcal{B}$ are finite and given explicitly as a part of the input, and the relation R is given explicitly as a table in $\mathcal{I} \times \mathcal{O}$.

Problem 1. *(TDP/PDP on explicit relations) We are given a tuple $I = (\mathcal{I}, \mathcal{O}, \mathcal{B}, R)$ where $\mathcal{I}, \mathcal{O}, \mathcal{B}$ are finite domains and $R \subseteq \mathcal{I} \times \mathcal{O}$. The problem is whether there exist relations $R_1 \subseteq \mathcal{I} \times \mathcal{B}$, and $R_2 \subseteq \mathcal{B} \times \mathcal{O}$ such that (R_1, R_2) meet the TDP/PDP conditions.*

Claim 1. *If $|\mathcal{B}| \geq |\mathcal{I}|$ or $|\mathcal{B}| \geq |\mathcal{O}|$ then TDP has a positive solution (and therefore PDP as well) and the relations that solve TDP can be found in a linear time to the size of the input.*

Proof. Suppose w.l.o.g. that $|\mathcal{B}| \geq |\mathcal{O}|$. Then we can define $R_2 = \{(g(o), o) | o \in \mathcal{O}\}$ where $g : \mathcal{O} \to \mathcal{B}$ is any injection. Then for TDP, $R_1 = \{(i, b) | (i, o) \in R \text{ and } b = g(o)\}$ is a relation that satisfies $R_1 \circ R_2 = R$. □

Therefore TDP/PDP become non-trivial when $|\mathcal{B}|$ is strictly smaller than $|\mathcal{I}|$ and $|\mathcal{O}|$.

Example 5.1. Let $\mathcal{I} = \{i_1, i_2\}, \mathcal{B} = \{b\}$, and $\mathcal{O} = \{o_1, o_2\}$. Let $R = \{(i_1, o_1), (i_2, o_2)\}$. Then the answer to TDP is negative as every non-empty composition of relations $R_1 \circ R_2$ with $Dom(R_1 \circ R_2) = Dom(R)$, must also include (i_1, o_2) or (i_2, o_1).

We next show that even for explicit setting, TDP/PDP are computationally hard, that is NP-complete. On a positive note, being in NP, solutions for TDP/PDP can be sought by various techniques such as reduction to SAT, then using SAT solvers. We give proof outline, full details appear in [9].

Theorem 5.2. *TDP/PDP on explicit relations are NP-complete.*

Proof. To see that TDP/PDP is in NP, guess R_1, R_2 and verify conditions of TDP/PDP.

For TDP, NP-hardness is shown by a reduction from the NP-complete problem: Covering by Complete Bipartite Subgraphs (CCBS) (Problem GT18 at[12]). In the CCBS we are given a bipartite graph G and $k > 0$, and the problem is whether G can be covered by k complete bipartite subgraphs. The idea of the proof is that all the $\mathcal{I}$ and $\mathcal{O}$ elements that are related to a specific element $b \in \mathcal{B}$ via R_1 and R_2, must be a part of a bipartite clique in the bipartite $(\mathcal{I}, \mathcal{O})$ graph that R describes.

For PDP where only a part of R needed to be "covered" although the solution should still cover the domain of R, NP-hardness follows by a reduction from Set Cover (Problem SP5 in [12]). □

5.2 Symbolic relations

In Section 5.1, the input relation is described explicitly. In many cases, however, although finite, the relation is too large to be described explicitly, and it makes more sense to describe it symbolically. Specifically, the domains are given as the set of all truth assignments over sets of Boolean variables, and the relation is described symbolically. Such representations have been studied in the literature, where they are often referred to as *succinct* representations, since they allow for a polynomial-size description of exponential-size domains and relations. In this section we explore a standard encoding in which the relation is described as a Boolean circuit, as in [3, 6]. Other symbolic encodings studied in the literature are Boolean formulas [30], and BDDs [7].

Let D, D' be finite domains of size $|D| = 2^n$, and $|D'| = 2^{n'}$ for some n, n'. A (succinct) *circuit* description of a relation $R \subseteq D \times D'$ is a circuit C_R of size polynomial in $max(n, n')$ with Boolean variables $d_1, \cdots d_n, d'_1, \cdots, d'_{n'}$ such that $C_R(d_1, \cdots, d_n, d'_1, \cdots, d'_{n'}) = 1$ iff $((d_1, \cdots, d_n), (d'_1, \cdots, d'_{n'})) \in R$. For more about succinct circuit representation, see [3, 11, 23].

Problem 2. *(TDP/PDP for symbolic relations) We are given a tuple $I = (n_{\mathcal{I}}, n_{\mathcal{O}}, n_{\mathcal{B}}, C_R)$ where $n_{\mathcal{I}}, n_{\mathcal{O}}, n_{\mathcal{B}}$ are logarithmic in the size of some domains $\mathcal{I}, \mathcal{O}$, $\mathcal{B}$*

respectively and C_R is a circuit that describes a relation $R \subseteq \mathcal{I} \times \mathcal{O}$. The problem is whether there exist circuits C_{R_1}, C_{R_2} that describe relations $R_1 \subseteq \mathcal{I} \times \mathcal{B}$, and $R_2 \subseteq \mathcal{B} \times \mathcal{O}$ such that (R_1, R_2) meet the TDP/PDP conditions.

For TDP/PDP, as in Claim 1, the problem becomes trivial when $n_{\mathcal{B}} \geq \min\{n_{\mathcal{I}}, n_{\mathcal{O}}\}$. Note that a variant of PDP, in which the required relations in the solution are functions can be viewed as an instance of the problem of Boolean functional synthesis, e.g. [10].

We next show that TDP/PDP are NEXPTIME-complete. We obtain this result by applying the computational-complexity theory of succinct-circuit representations for "logtime" reductions [3] to the NP-hardness reductions described in Section 5.1. We describe this in details.

A reduction from languages $A \subseteq \Sigma^*$ to $B \subseteq \Sigma^*$, for a finite alphabet Σ, is a function $f : A \to B$ such that $x \in A$ iff $f(x) \in B$. The function f is called a *logtime reduction* if the i-th symbol of $f(x)$ can be computed in a time logarithmic of the size of x. This is done by using a so called "direct-input-access" Turing machine, which has a specific "index" tape in which a binary index i is written and then the i-th symbol of the input string x is computed. See [3], which also states a generalization of the following:

Theorem 5.3. (Balcazar, Lozano, Toran [3]) *For every language $B \subseteq \Sigma^*$, if B is NP-hard under logtime reducibility, then the circuit representation of B is NEXPTIME-hard under polynomial-time reducibility.*

From Theorem 5.3 we get:

Theorem 5.4. *TDP/PDP for symbolic relations are NEXPTIME-complete.*

We give a proof sketch; the full proof appears in [9].

Proof Sketch. For membership in NEXPTIME, we construct the relation R from C_R in time exponential in n, then use the explicit solution (in NP), as in Section 5.1. For NEXPTIME hardness we first retrace the chain of log-time reductions from a Turing machine description of an NP language, via SAT to either TDP or PDP on explicit relations. Most of these reductions are found in [12], in addition to Theorem 5.2 from Section 5.1. All these reductions, which are functions from strings to strings, have the property that only a few symbols are required from the input string, as well as a local comparison, in order to determine the identity of the j'th symbol of the output string: the property that is required for logtime reductions. Having obtained that these reductions are logtime, NEXPTIME hardness follows from the Conversion Lemma from [3] and by using transitivity of polynomial-time reductions. □

5.3 Automatic relations

In many applications we need to consider input and output as streams of symbols, with some desired relation between the input stream and the output stream. The most basic description for such systems, the one that we explore in this work, is when the domains are (possibly infinite) sets of finite words over finite alphabets, and the given relation is a regular language, given as a deterministic finite automaton (DFA), over the product alphabet of the input and output domains. The setting that we consider in this paper is of *automatic relations*. Automatic relations provide a context for a rich theory of automatic structures, cf. [16, 28], with a solvable decision for first-order logic. We follow here the convention in *regular model checking*, cf. [5], where *length-preserving* automatic relations, with input and output symbols interleaved, are used as input/output specifications for each step of reactive systems. (Thus, unlike the definitions in [16, 28], we do not allow padding.)

Given finite alphabets Σ, Σ' with domains $D \subseteq \Sigma^*$, $D' \subseteq \Sigma'^*$, and a relation $R \subseteq D \times D'$, we say that a DFA $\mathcal{A}_R$ over alphabet $(\Sigma \times \Sigma')$ describes R if: $(\vec{d}, \vec{d'}) \in L(\mathcal{A}_R)$ if and only if $(\vec{d}, \vec{d'}) \in R$. Note the slight abuse of notation as $L(\mathcal{A}_R)$ describes words in $(\Sigma \times \Sigma')^*$ while R describes words in $\Sigma^* \times \Sigma'^*$. Thus, we assume that input and output streams have the same lengths, that is, $(\vec{d}, \vec{d'}) \in R$ implies that $|\vec{d}| = |\vec{d'}|$.

Problem 3. *(TDP/PDP for automatic relations) We are given a tuple $I = (\Sigma_{\mathcal{I}}, \Sigma_{\mathcal{B}}, \Sigma_{\mathcal{O}}, \mathcal{A}_R)$, where $\Sigma_{\mathcal{I}}$, $\Sigma_{\mathcal{B}}$, and $\Sigma_{\mathcal{O}}$ are finite alphabets, and $\mathcal{A}_R$ is a DFA that describes a relation $R \subseteq \Sigma_{\mathcal{I}}^* \times \Sigma_{\mathcal{O}}^*$. The problem is whether there exist DFAs $\mathcal{A}_{R_1}, \mathcal{A}_{R_2}$ that describe relations $R_1 \subseteq \Sigma_{\mathcal{I}}^* \times \Sigma_{\mathcal{B}}^*$ and $R_2 \subseteq \Sigma_{\mathcal{B}}^* \times \Sigma_{\mathcal{O}}^*$ such that (R_1, R_2) meet the TDP/PDP conditions.*

As in the explicit and symbolic cases, the problem becomes non trivial for TDP/PDP only when $|\Sigma_{\mathcal{B}}| < \min\{|\Sigma_{\mathcal{I}}|, |\Sigma_{\mathcal{O}}|\}$. While TDP/PDP in the symbolic setting can be solved by reduction to the explicit setting, this cannot be done here as the domains are possibly infinite. Indeed, automatic-relation TDP/PDP seems to be a challenging problem. We next conjecture that automatic-relation TDP/PDP are undecidable and explain the motivation for this conjecture and why an automatic-theoretic approach may not be helpful for TDP/PDP. We then show that even for the most basic case, in which the given relation is the equality relation, TDP/PDP can already be viewed as an algorithmic problem in automata that is equivalent to the Positivity Problem whose decidability is still open [21, 29]. We next show that for intermediate alphabet that is of exponential size, TDP/PDP on automatic relations can

be reduced to TDP/PDP on binary intermediate alphabet. Finally we show by an automata-theoretic approach that a "strategic" variant of PDP, in which the required relations are in form of transducers, is decidable, and in fact is in EXPTIME.

5.3.1 An undecidability conjecture for TDP/PDP

A notable positive result about automatic relations is the decidability of their first-order theories [16, 28]. On the other hand, most second-order problems over automatic relations are undecidable; for example, checking the existence of an Eulerian cycle in an automatic graph is highly undecidable [20]. TDP/PDP are essentially second-order problems–we ask for the existence of R_1 and R_2 under the TDP/PDP conditions. Our conjecture is that this problem is undecidable. We provide here intuition to justify this conjecture.

We show in Section 6 below that TDP/PDP are decidable for automatic relations when a hint, in the form of a DFA for R_1 or R_2 is given. The idea is that given one component, say R_1, there is a maximal second component R_2 that can be obtained from R_1 so that if R_1 is the first component of some proper decomposition, then R_2 is a proper second component. When R_1 is given as a DFA, we can then construct a DFA for R_2. Can we leverage this idea towards solving the problem in full? That is, can we search for, say, an automaton for R_1 that together with the maximal R_2 constructed below, forms a solution for TDP/PDP?

It is tempting to try to use an automata-theoretic approach similar to the strategic PDP (see Section 5.3.4 below) in which we consider representing $R_1 \subseteq (\Sigma_I \times \Sigma_B)^*$ as a labeled tree $\tau_1 : (\Sigma_I \times \Sigma_B)^* \mapsto \{0, 1\}$. Then we can try to define a tree automaton $\mathcal{A}_1$ that accepts a tree τ_1 iff it is a correct hint for the total or partial decomposition of an input/output relation $R \subseteq (\Sigma_I \times \Sigma_B)^*$. The difficulty is that such an automaton has to check two properties of τ_1: (1) The domain of R_1 has to be equal to the domain of R. This is essentially a requirement on the projection of τ_1 on Σ_I. (2) The composition of R_1 with the maximal R_2 is contained in or equal to R. This is essentially a requirement on the projection of τ_1 on Σ_B. Known automata-theoretic techniques exist for dealing with projection of trees, see for example [17]. We currently, however, have no known technique to deal with two orthogonal projections, as we have here.

Of course, this argument only shows that a particular technique to attack the problem is unlikely to be successful. To justify the conjecture, we note that the dual-projection problem is reminiscent to the problem of distributed temporal synthesis, which was shown to be undecidable [26] (though there is no obvious formal connection between the decomposition problem and the distributed-synthesis problem). There we are representing an overall system strategy as a tree where tree labels correspond to actions of system components and tree edges correspond to environment actions. When different systems components are expected to act independently, without knowledge of actions by other components, overall systems strategies has to be decomposed into separate strategies as projections of the system strategy. The similarity between the two problems motivates us to formulate the following conjecture.

Conjecture 5.5. *TDP/PDP for automatic relations is undecidable.*

5.3.2 Decomposing equality

To support the claim of how non-trivial the decomposition problem is, we consider a more simple and abstract variant of automatic TDP. We consider a very simple case in which the given automatic relation is trivial, and the intermediate alphabet is binary. We do not even require the decomposed relation R_1, R_2 to be realized by automata, although we do require the length of every matching words in R_1 and R_2 to be the same. Specifically, given a regular language L over $\Sigma_{\mathcal{I}}^*$, let $R_L^= \subseteq \Sigma_{\mathcal{I}} \times \Sigma_{\mathcal{I}}$ be the *equality* relation over L. i.e. $R = \{(w, w) | w \in L\}$. Given L over $\Sigma_{\mathcal{I}}^*$ and $\Sigma_{\mathcal{B}} = \{0, 1\}$, we ask whether there are relations (R_1, R_2) that meet the TDP properties with respect to $R_L^=$.

First see that we can assume wlog that (R_1, R_2) are functions since existence of such relations leads to existence of such functions. Next, note that R_1 cannot relate two distinct words in L to the same word in Σ_B^*. This is because otherwise a word from Σ_B^* has to meet two distinct $\Sigma_{\mathcal{I}}^*$ elements, thus either break the equality property or break the TDP property. Therefore we have that R_1 is an injection from $\Sigma_{\mathcal{I}}^*$ to $\Sigma_{\mathcal{B}}^*$. As such, finding an R_1 that is such an injection also gives us $R_2 = R_1^{-1}$.

Let L_n be the words in L of size n. Note that if there is n for which $|L_n| > 2^n$ then no such R_1 can be found. If, however, for every n we have $|L_n| \leq 2^n$ then a function R_1 can be simply realized by ordering the words in every L_n in lexicographic order and relating each one to the $\Sigma_{\mathcal{B}}^n$ word that encodes the index in binary.

Therefore the problem of TDP in this setting is reduced to the following problem: given a regular language L over a finite alphabet Σ, where for every n, L_n is the set of words in L of size n, does $|L_n| \leq 2^n$ for every n? We call this problem the *The Exponential-Bound Problem* (EBP). The following theorem, however, shows that EBP is equivalent to the problem of *Positivity*, described below, whose decidability has been famously open for decades [21, 29]. Although this is not a direct reduction

to automatic TDP, this relation indicates to the hardness of solving automatic TDP on even a simple relation.

A *linear recurrence sequence* (LRS) is a sequence of integers $\langle u_n \rangle_{n=0}^{\infty}$ satisfying a recurrence relation: there exist constants integers $a_1, a_2, \ldots, a_k$ such that, for all $n \geq 0$, $u_{n+k} = a_1 u_{n+k-1} + a_2 u_{n+k-2} + \ldots + a_k u_n$. If the initial values $u_0, \ldots, u_{k-1}$ of the sequence are provided, the recurrence relation defines the rest of the sequence uniquely. Given a linear recurrence sequence (LRS) $\langle u_n \rangle_{n=0}^{\infty}$, the *Positivity Problem* asks whether all terms of the sequence are non-negative.

Theorem 5.6. *EBP is Equivalent to Positivity.*

Proof. We first show that Positivity reduces to EBP. Let $\langle u_n \rangle_{n=0}^{\infty}$ be an LRS; we show how positivity for the sequence $\langle -u_n \rangle_{n=0}^{\infty}$ can be formulated as an EBP problem. To this end, we invoke Corollary 4 from [2] to obtain a rational stochastic matrix M of size $c \times c$ (denoted $\widetilde{Q}$ in the proof of Proposition 2 from *op. cit.*, where the value of c is also stated) such that, for all $n \geq 0$,

$$(M^{2n+1})_{1,2} \leq \frac{1}{4} \quad \text{iff} \quad u_n \leq 0 \quad \text{iff} \quad -u_n \geq 0\,, \qquad (1)$$

in other words the positivity of $\langle -u_n \rangle_{n=0}^{\infty}$ is violated iff there is some n such that the $(1,2)$-*th* entry of M^{2n+1} is strictly larger than $1/4$.

In fact, as noted in the comments following Corollary 4 in [2], M can be chosen so that its entries are dyadic rationals, i.e., having denominator some power of 2. Let us therefore assume this to be the case, and let 2^p be the largest such power. Write $J = 2^p M$ and $N = (2^p M)^2$. Then J and N are square matrices with non-negative integer coefficients, and hence there is some DFA $\mathcal{A}$ such that $(J \cdot N^n)_{1,2}$ is the number of words of length $n+1$ accepted by $\mathcal{A}$. More precisely, $\mathcal{A}$ has initial state s, and c further states $q_1, \ldots, q_c$. The single accepting state is q_2. To define the transition function, if the $(1,j)$-*th* entry of J is ℓ, then we postulate ℓ transitions going from state s to state q_j, each labelled with a new (fresh) letter. Likewise, if the (i,j)-*th* entry of N is ℓ, then we include ℓ transitions going from q_i to q_j, again for each one using a new letter as label. In this way, J and N can be viewed as the adjacency matrices of the underlying directed multigraph of $\mathcal{A}$, and $(J \cdot N^n)_{1,2}$ counts the number of paths in $\mathcal{A}$ going from s to q_2 in $n+1$ steps. Since by construction, different paths give rise to different words, $(J \cdot N^n)_{1,2}$ does indeed correspond to the number of words of length $n+1$ accepted by $\mathcal{A}$.

Writing $L(\mathcal{A}) = L$, Eq. (1) becomes, for all $n \geq 0$,

$$L_{n+1} \leq \frac{2^p 2^{2pn}}{4} = \frac{2^{2p(n+1)}}{2^{p+2}} \quad \text{iff} \quad -u_n \geq 0\,. \qquad (2)$$

We now modify automaton $\mathcal{A}$ by lengthening every transition in $\mathcal{A}$ by a factor of $2p$; more precisely, for every transition $q \to q'$, create $2p-1$ fresh non-accepting states $r_1, \ldots, r_{2p-1}$ and replace $q \to q'$ by the sequence $q \to r_1 \to r_2 \to \ldots \to r_{2p-1} \to q'$, all labelled with the same letter as the original transition. The initial and accepting states otherwise remain unchanged. Let us call the resulting DFA $\mathcal{A}'$, with accepted language $L(\mathcal{A}') = L'$. In moving from $\mathcal{A}$ to $\mathcal{A}'$, the net effect has been to increase the length of every accepted word by a factor of $2p$; note also that if m is not a multiple of $2p$, then $L'_m = 0$.

Combining the above with Eq. (2), we conclude that the LRS $\langle -u_n \rangle_{n=0}^{\infty}$ is positive iff for all $m \geq 0$, $L'_m \leq \frac{2^m}{2^{p+2}}$, i.e., $2^{p+2} L'_m \leq 2^m$.

Inflating the alphabet size of $\mathcal{A}'$ by a factor of 2^{p+2}, we can easily manufacture a DFA $\mathcal{A}''$ with accepted language $L(\mathcal{A}'') = L''$ having the property that, for all $m \geq 0$, $L''_m = 2^{p+2} L'_m$. It therefore follows that the LRS $\langle -u_n \rangle_{n=0}^{\infty}$ is positive iff for all $m \geq 0$, $L''_m \leq 2^m$, which completes the reduction of Positivity to EBP.

Finally, since the series obtained from the number of distinct words of length n in an automaton makes an LRS [22], we have that EBP also reduces to Positivity, so that the two problems are in fact equivalent. □

5.3.3 Reduction to binary alphabet

Since the size of the intermediate alphabet plays a crucial role in the solution of TDP/PDP, one may ask whether it suffices to search for solutions for only the binary case. We show that the answer is positive for intermediate alphabet that is of size of exponent. Specifically we show by reduction that for every automatic TDP/PDP instance $I = (\Sigma_{\mathcal{I}}, \Sigma_{\mathcal{B}}, \Sigma_{\mathcal{O}}, \mathcal{A}_R)$, where $|\Sigma_{\mathcal{B}}| = 2^m$ for some $m > 0$, there is a TDP/PDP instance $I' = (\Sigma_{\mathcal{I}}, \{0,1\}, \Sigma_{\mathcal{O}}, \mathcal{A}_{R'})$ such that I has a solution if and only if I' has a solution. The idea behind the reduction is to encode every letter in $\Sigma_{\mathcal{B}}$ in binary, and use this encoding for the construction of I'. $\mathcal{A}_{R'}$ is therefore constructed from $\mathcal{A}_R$ by replacing every edge labeled (i,o) with a path of m edges, each with the same label (i,o). We give below an outline of the reduction proof; see [9] for details.

Theorem 5.7. *There is a TDP/PDP solution to $I = (\Sigma_{\mathcal{I}}, \Sigma_{\mathcal{B}}, \Sigma_{\mathcal{O}}, \mathcal{A}_R)$ where $|\Sigma_{\mathcal{B}}| = 2^m$ for some $m > 0$, if and only if there is a TDP/PDP solution to $I' = (\Sigma_{\mathcal{I}}, \{0,1\}, \Sigma_{\mathcal{O}}, \mathcal{A}_{R'})$.*

Proof. Let *bin* be a function in which $bin(b)$ is a binary encoding of the letter $b \in \Sigma_{\mathcal{B}}$ in m bits, and let $bin_j(b)$ be the j'th bit in $bin(b)$. Since Σ_B is of size 2^m, the function *bin* is a bijection. Assume that I has a solution $(\mathcal{A}_{R_1}, \mathcal{A}_{R_2})$. We first construct an automaton $\mathcal{A}_{R'_1}$ over alphabet $(\Sigma_{\mathcal{I}} \times \{0,1\})$ from $\mathcal{A}_{R_1}$ by replacing every edge labeled (i,b) with an m-edges path with edge labels $(i, bin_0(b)), \cdots (i, bin_{m-1}(b))$, thus such a path describes

the word $(i^m, bin(b))$ (where i^m is the letter i concatenated m times). Same, we construct an automaton $\mathcal{A}_{R'_2}$ over alphabet $(\{0,1\} \times \Sigma_{\mathcal{O}})$ from $\mathcal{A}_{R_2}$ by replacing every edge labeled (b, o) with an m-edges path with edge labels that describes the word $(bin(b), o^m)$. Then since (R_1, R_2) solve TDP/PDP for the instance I, we have that (R'_1, R'_2) solve TDP/PDP for the instance I'.

The other side of the proof is more involved since suppose we assume $(\mathcal{A}_{R'_1}, \mathcal{A}_{R'_2})$ solve I'. Then we cannot guarantee that these automata have an "m-path" structure that can reverse the construction we have just described. Therefore to reach a solution, we need to reason about the automata as regular expressions. We give the proof in [9]. □

Theorem 5.7 is stated only for alphabet that are of exponent size. We do not yet know the solution for the general case. We thus have the following conjecture.

Conjecture 5.8. *There is a reduction for TDP/PDP from an intermediate alphabet of arbitrary size to the binary alphabet.*

Corollary 6.8 shows that TDP/PDP on automatic relations with unary intermediate alphabet is solvable.

5.3.4 Strategic PDP is decidable

A specific version of PDP, that captures essential concepts in synthesis [10, 18, 26], is when we require the solution to be strategies. For that, we define *Strategic PDP* as PDP in which the required relations R_1 and R_2 are functions (defined in Section 3) that are represented by transducers T_1 and T_2, respectively.

Strategic PDP can be viewed as a game of incomplete information. Since the information "flows" in one direction, however, the key to proving decidability for this problem is to view the problem as a one-way chain communication of distributed synthesis from [18] to synthesize the required transducers.

Theorem 5.9. *Strategic PDP is in EXPTIME.*

We give a proof outline below. Full proof appears in [9].

Proof. Note that T_1 is a finite-state realization of a function $f_1 : \Sigma^*_{\mathcal{I}} \to \Sigma_{\mathcal{B}}$, and T_2 is a finite-state realization of a function $f_2 : \Sigma^*_{\mathcal{B}} \to \Sigma_{\mathcal{O}}$. We first consider trees of the form $\tau_1 : \Sigma^*_{\mathcal{I}} \to \Sigma_{\mathcal{B}} \times \Sigma_{\mathcal{O}}$. Such a tree represents simultaneously both f_1 and f_2. A branch of this tree can be viewed as a word $w \in (\Sigma_I \times \Sigma_B \times \Sigma_O)^\omega$. By running the DFA $\mathcal{A}_R$ on w we can check that every prefix of w is consistent with R. Thus, we can construct a deterministic automaton $\mathcal{A}^t_R$ on infinite trees that checks that all prefixes of all branches in τ_1 are consistent with R.

Note, however, that in τ_1 the $\Sigma_{\mathcal{O}}$ values depend not only on the $\Sigma_{\mathcal{B}}$ values but also on the $\Sigma_{\mathcal{I}}$ values, while in f_2 the domain is $\Sigma^*_{\mathcal{B}}$. So now we consider a tree $\tau_2 : \Sigma^*_{\mathcal{B}} \to \Sigma_{\mathcal{O}}$. We now simulate $\mathcal{A}^t_R$ on τ_2 (cf., [18]). The idea is to match each branch of τ_1 with a branch of τ_2 according to the $\mathcal{B}$-values. This means that we have several branches of the run of $\mathcal{A}^t_R$ running on one branch of τ_2. We thus obtain an alternating tree automaton $\mathcal{A}_2$, whose size is linear in the size of $\mathcal{A}_R$. We can now check in exponential time non-emptiness of $\mathcal{A}_2$, to see if a function f_2 exists. If $\mathcal{A}_2$ is non-empty, then we obtain a witness transducer T_2 whose size is exponential in $\mathcal{A}_R$.

Finally, we consider a tree $\tau'_1 : \Sigma^*_{\mathcal{I}} \to \Sigma_{\mathcal{B}}$ that represents f_1. We run both $\mathcal{A}_R$ and T_2 on each branch of τ'_1, where T_2 generates the $\mathcal{O}$-values that were present in τ_1 but not in τ'_1. Using these values $\mathcal{A}_R$ checks that each prefix is consistent with R. We have obtained a deterministic tree automaton $\mathcal{A}_1$, whose size is exponential in $\mathcal{A}_R$. We can now check non-emptiness of $\mathcal{A}_1$, and obtain a witness transducer T_1. Thus, we can solve Strategic PDP for automatic relations in exponential time. □

6 Decomposition with a hint

Our results above indicate that fully automated decomposition is a hard problem. Can we ameliorate this difficulty by including a "human in the loop"? Indeed, in some decomposition scenarios, a part of a decomposition is already given, and the challenge is to find the complementary component. This can be thought of as a partial solution that is offered by a human intuition, e.g. a domain expert. In the context of our framework we have that a candidate to either R_1 or R_2 is given (in the relevant description formalism) as a "hint". The question then is whether, given one possible component, a complementary component indeed exists, and can be constructed, such that both relations together meet the TDP/PDP conditions. To avoid needless repetition, we discuss below only TDP/PDP problems with a hint R_1. Unless mentioned otherwise, all the statements in this section can be applied to R_2 as a hint by almost identical arguments. The results that concern R_2 being a hint are discussed in [9].

Definition 6.1. In the TDP/PDP problem with a hint R_1 we are given input, output and intermediate domain $\mathcal{I}, \mathcal{O}, \mathcal{B}$, relations $R \subseteq \mathcal{I} \times \mathcal{O}$ and $R_1 \subseteq \mathcal{I} \times \mathcal{B}$. The goal is to find whether there is a relation $R_2 \subseteq \mathcal{B} \times \mathcal{O}$ such that (R_1, R_2) meet the TDP/PDP conditions.

The exact nature of the domains and relations varies according to the problem setting (explicit, etc.). As we see, such a hint as a partial solution relaxes the computational difficulty of TDP/PDP, shown in previous sections, considerably. To that end, we show the following *maximum property* that is relevant for all TDP/PDP settings. Given a TDP/PDP instance with a hint R_1, define a relation $R'_2 \subseteq (\mathcal{B} \times \mathcal{O})$ to be $R'_2 = \{(b, o) | \forall i \in$

$\mathcal{I}((i,b) \in R_1 \rightarrow (i,o) \in R)\}$. Note that $Dom(R_2')$ can strictly contain $Img(R_1)$.

Lemma 6.2. *Every solution for TDP/PDP with a hint R_1 is contained in R_2' and if there exists such a solution then R_2' is a solution as well.*

Proof. We prove for TDP, we skip the proof for PDP that is almost identical. Let R_2 be a solution to TDP with a hint R_1. We first see that $R_2 \subseteq R_2'$. Let $(b,o) \in R_2$. Suppose there exists i such that $(i,b) \in R_1$ and $(i,o) \notin R$. Then we have that $R_1 \circ R_2 \not\subseteq R$ which means R_2 is not a solution, a contradiction. Therefore we have that for all i, if $(i,b) \in R_1$ then $(i,o) \in R$, hence $(b,o) \in R_2'$. To see that R_2' is a solution, first see that since R_2 is a solution contained in R_2' then $Dom(R_1 \circ R_2') = Dom(R)$ and $Img(R_1) \subseteq Dom(R_2')$. Let $(i,o) \in R_1 \circ R_2'$. Then there is a b such that $(i,b) \in R_1$ and $(b,o) \in R_2'$, which means that by the definition of R_2' and since $(i,b) \in R_1$, it must be that $(i,o) \in R$. Finally, assume that $(i,o) \in R$. Then there is b such that $(i,b) \in R_1$, and $(b,o) \in R_2''$ for some solution R_2'' contained in R_2'. Therefore $(b,o) \in R_2'$ so $(i,o) \in R_1 \circ R_2'$. □

From Lemma 6.2 we get a simple method to solve TDP/PDP with a hint R_1 for the explicit settings as follows: Construct R_2' and check that (R_1, R_2') meet the TDP/PDP conditions. If R_2' meets these conditions then (R_1, R_2') is a solution for TDP/PDP. Otherwise, there is no solution with R_1 as a hint. We thus obtain:

Theorem 6.3. *TDP/PDP with a hint R_1 for explicit relations are in PTIME.*

The definition of R_2' can also solve the symbolic setting with a hint. Thus the following result allows the use of QBF solvers for finding solution for R_1, see proof in [9].

Theorem 6.4. *TDP/PDP with a hint C_{R_1} for symbolic relations are in Π_3^P.*

For the automatic relation setting we have the following result for TDP/PDP with a hint R_1, where we assume that R_1 is given as a DFA (though users are more likely to use regular expressions). A similar detailed result for hint R_2 is given in [9].

Theorem 6.5. *TDP/PDP with a hint $\mathcal{A}_{R_1}$ for automatic relations are in EXPTIME.*

Proof. Assume that $\mathcal{A}_R = (\Sigma_\mathcal{I} \times \Sigma_\mathcal{O}, Q, q^0, \delta, F)$ is a DFA that describes R and $\mathcal{A}_{R_1} = (\Sigma_\mathcal{I} \times \Sigma_\mathcal{B}, Q_1, q_1^0, \delta_1, F_1)$ is a DFA hint. We show how to construct a DFA $\mathcal{A}_2 = (\Sigma_\mathcal{B} \times \Sigma_\mathcal{O}, Q_2, q_2^0, \delta_2, F_2)$ that describes the maximum relation R_2'. We define $\mathcal{A}' = (\Sigma_\mathcal{I} \times \Sigma_\mathcal{B} \times \Sigma_\mathcal{I} \times \Sigma_\mathcal{O}, Q_1 \times Q, (q_1^0, q^0), \delta_1 \times \delta, F')$ as the product automaton of $\mathcal{A}_{R_1}$ and $\mathcal{A}_R$. We re-define the accepting states $F' = F \times F_1$ by setting $F' = \{(q,q') | q \in F_1 \rightarrow q' \in F\}$. For the transition function, we first extract from $\mathcal{A}'$ all the (i,b,i',o) edges in which $(i \neq i')$. Next we delete the alphabets $\Sigma_\mathcal{I} \times \Sigma_\mathcal{I}$ from all the labels in $\mathcal{A}'$. This results in a non-deterministic automaton, that we determinize (with possibly an exponential blow-up) in the standard way through the subset construction, with the one exception that we set every super-state (in the subset construction) to be accepting if and only if *all* of its elements are accepting states. This results in a DFA $\mathcal{A}_2$ over $\Sigma_\mathcal{B} \times \Sigma_\mathcal{O}$ that describes the maximum relation R_2'.

Finally, we check that $(\mathcal{A}_{R_1}, \mathcal{A}_2)$ meet the condition for TDP/PDP by standard emptiness checks on projections, and intersections of $\mathcal{A}_{R_1}$ and $\mathcal{A}_2$; More details are found in [9]. □

For Strategic PDP in the automatic setting, we assume that the hint is given in the form of a transducer.

Theorem 6.6. *Strategic PDP with a hint T_1 is in EXPTIME.*

Proof. For Strategic PDP with a hint T_1 as a transducer we simplify the construction of Section 5.3.4. We directly construct A_2 by taking the product of A_R and T_1, then solve non-emptiness for A_2 and obtain a witness transducer T_2. Since however the resulting A_2 is still an alternating automaton, this construction is still in an exponential time. □

A complete proof appears in [9], where we also show that Strategic PDP with a hint T_2, in which no alternation is required, can be done in PTIME.

Theorem 6.5 provides an elegant solution for TDP/PDP in the automatic setting, in case one of the component, say, $\mathcal{A}_{R_1}$ is required to be bounded by a given (unary) size k: guess an automaton $\mathcal{A}_{R_1}$ of size k as a hint R_1, then use Theorem 6.5. This gives us the following.

Corollary 6.7. *TDP/PDP on automatic relations when one of the component DFAs is bounded by a given (unary) size is in NEXPTIME.*

As another corollary of Theorem 6.5, we can say the following on unary intermediate alphabet.

Corollary 6.8. *TDP/PDP for automatic relations with a unary intermediate alphabet is in EXPTIME.*

Proof. For a unary intermediate alphabet $\Sigma_\mathcal{B} = \{b\}$, there is a unique solution R_1 that can be used as a hint. Indeed, since for a solution R_1 we have that $Dom(R_1) = Dom(R)$ (see remark in Section 4 and proof in [9]), then every word of length n in $Dom(R_1)$ must be paired with the word b^n. Therefore $\mathcal{A}_{R_1}$ is constructed from $\mathcal{A}_R$ by simply replacing every $\Sigma_\mathcal{O}^*$ word in $\mathcal{A}_R$ with the unique word of the same length over $\Sigma_\mathcal{B}$ (this projection yields an NFA, but determinizing in order to get a DFA as a hint is not necessary, since the proof of Theorem 6.5 can work with $\mathcal{A}_{R_1}$ given as an NFA as well without changing the upper bound). □

7 Discussion

We studied here a formal model of sequential decomposition, a fundamental concept in computational thinking. We showed that while decomposition is viewed as an approach to tame design complexity, complexity is not so easily tamed and decomposition can be quite difficult when viewed as a computational problem. Human intuition, used to offer hints to the decomposition algorithm, is therefore necessary to tame the complexity of the decomposition problem. The complexity of TDP/PDP in the automatic-relation setting, conjectured to be undecidable, is still open. It is related to other decision problems for automatic relations, cf. [20], and is a subject of future work.

Acknowledgments

We thank Orna Kupferman, Nir Piterman, Lucas M. Tabajara, and Jacobo Toran for useful discussions, and the anonymous reviewers for their suggestions. This work is supported in part by NSF grants CCF-1319459, and by by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering". Joël Ouaknine was supported by ERC grant AVS-ISS (648701).

References

[1] M. Abadi and L. Lamport. 1994. Decomposing specifications of concurrent systems. In *Proc. IFIP Working Conference on Programming Concepts, Methods and Calculi*. 327–340.

[2] S. Akshay, T. Antonopoulos, J. Ouaknine, and J. Worrell. 2015. Reachability problems for Markov chains. *Inf. Process. Lett.* 115, 2 (2015), 155–158.

[3] J. L. Balcázar, A. Lozano, and J. Torán. 1992. The complexity of algorithmic problems on succinct instances. In *Computer Science*. Springer, 351–377.

[4] S. Bliudze, J. Sifakis, M. Bozga, and M. Jaber. 2014. Architecture internalisation in BIP. In *Proc. 17th International Symposium on Component-Based Software Engineering*. 169–178.

[5] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. 2000. Regular model checking. In *Int'l Conf. on Computer-Aided Verification*. Springer, 403–418.

[6] B. Das, P. Scharpfenecker, and J. Torán. 2016. CNF and DNF succinct graph encodings. *Information and Computation* (2016).

[7] J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. 1999. The Complexity of Problems on Graphs Represented as OBDDs. *Chicago J. Theor. Comput. Sci.* 1999 (1999).

[8] L. Fix, N. Francez, and O. Grumberg. 1991. Program composition and modular verification. In *Proc. 18th Int. Colloq. on Automata, Languages, and Programming*. 93–114.

[9] D. Fried, A. Legay, J. Ouaknine, and M.Y. Vardi. 2018. Sequential Relational Decomposition. Full version. (2018). https://www.cs.rice.edu/~vardi/papers/index.html

[10] D. Fried, L.M. Tabajara, and M.Y. Vardi. 2016. BDD-based Boolean functional synthesis. In *Proc. 28th Int'l Conf. on Computer Aided Verification*. 402–421.

[11] H. Galperin and A. Wigderson. 1983. Succinct representations of graphs. *Information and Control* 56, 3 (1983), 183–198.

[12] M. R. Garey and D. S. Johnson. 1979. *Computers and intractability: A guide to the theory of NP-Completeness*. W. H. Freeman.

[13] C. A. R. Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (1969), 576–580.

[14] J. E. Hopcroft, R. Motwani, and J. D. Ullman. 2003. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley.

[15] D.S. Johnson. 1987. The NP-completeness column: An ongoing guide. *Journal of algorithms* 8, 2 (1987), 285–303.

[16] B. Khoussainov and A. Nerode. 1995. Automatic presentations of structures. In *Proc. Workshop on Logic and computational complexity (LNCS 960)*. Springer, 367–392.

[17] O. Kupferman and M.Y. Vardi. 2000. Synthesis with incomplete informatio. *Advances in Temporal Logic* 16 (2000), 109–127.

[18] O. Kupferman and M.Y. Vardi. 2001. Synthesizing distributed Systems. In *Proc. 16th IEEE Symp. on Logic in Computer Science*. 389–398.

[19] E. Kushilevitz and N. Nisan. 1997. *Communication complexity*. Cambridge University Press.

[20] D. Kuske and M. Lohrey. 2010. Some natural decision problems in automatic graphs. *J. of Symbolic Logic* (2010), 678–710.

[21] J. Ouaknine and J. Worrell. 2014. On the Positivity problem for simple linear recurrence sequences. In *Proc. ICALP (LNCS)*, Vol. 8573. Springer.

[22] J. Ouaknine and J. Worrell. 2015. On linear recurrence sequences and loop termination. *SIGLOG News* 2, 2 (2015), 4–13.

[23] C. H. Papadimitriou and M. Yannakakis. 1986. A note on succinct representations of graphs. *Information and Control* 71, 3 (1986), 181–185.

[24] D.L. Parnas. 1972. On the criteria to be used in decomposing systems into modules. *Commun. of the ACM* 15, 12 (1972), 1053–1058.

[25] A. Pnueli. 1977. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science*. 46–57.

[26] A. Pnueli and R. Rosner. 1990. Distributed reactive systems are hard to synthesize. In *Proc. 31st Annual Symposium on Foundations of Computer Science*. 746–757.

[27] K. Rath, V. Choppella, and S.D. Johnson. 1995. Decomposition of sequential behavior using interface specification and complementation. *VLSI Design* 3, 3-4 (1995), 347–358.

[28] S. Rubin. 2004. *Automatic structures*. Ph.D. Dissertation. University of Auckland, New Zealand.

[29] M. Soittola. 1976. On D0L Synthesis Problem. In *Automata, Languages, Development*. North-Holland.

[30] H. Veith. 1997. Languages represented by Boolean formulas. *Inf. Process. Lett.* 63, 5 (1997), 251–256.

[31] J. M. Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.

[32] J. M. Wing. 2011. Computational thinking. In *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing*. 3.