Theoretical Computer Science 712 (2018) 50–72

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Synthesis from component libraries with costs ☆

Guy Avni [a,*], Orna Kupferman [b]

[a] IST Austria, Austria
[b] Hebrew University, Israel

ARTICLE INFO

Article history:
Received 4 May 2016
Received in revised form 23 September 2017
Accepted 6 November 2017
Available online 9 November 2017
Communicated by J.-F. Raskin

Keywords:
Synthesis
Component libraries
Cost-sharing games
Lattice automata

ABSTRACT

Synthesis is the automated construction of a system from its specification. In real life, hardware and software systems are rarely constructed from scratch. Rather, a system is typically constructed from a library of components. Lustig and Vardi formalized this intuition and studied LTL synthesis from component libraries. In real life, designers seek optimal systems. In this paper we add optimality considerations to the setting. We distinguish between quality considerations (for example, size – the smaller a system is, the better it is), and pricing (for example, the payment to the company who manufactured the component). We study the problem of designing systems with minimal quality-cost and price. A key point is that while the quality cost is individual – the choices of a designer are independent of choices made by other designers that use the same library, pricing gives rise to a resource-allocation game – designers that use the same component share its price, with the share being proportional to the number of uses (a component can be used several times in a design). We study both closed and open settings, and in both we solve the problem of finding an optimal design. In a setting with multiple designers, we also study the game-theoretic problems of the induced resource-allocation game.

1. Introduction

Synthesis is the automated construction of a system from its specification. The classical approach to synthesis is to extract a system from a proof that the specification is satisfiable. In the late 1980s, researchers realized that the classical approach to synthesis is well suited to *closed* systems, but not to *open* (also called *reactive*) systems [1,32]. A reactive system interacts with its environment, and a correct system should have a *strategy* to satisfy the specification with respect to all environments. It turns out that the existence of such a strategy is stronger than satisfiability, and is termed *reliability*.

In spite of the rich theory developed for synthesis, in both the closed and open settings, little of this theory has been reduced to practice. This is in contrast with verification algorithms, which are extensively applied in practice. We distinguish between algorithmic and conceptual reasons for the little impact of synthesis in practice. The algorithmic reasons include the high complexity of the synthesis problem (PSPACE-complete in the closed setting [37] and 2EXPTIME-complete in the

☆ This research was supported in part by the European Research Council (ERC) under grant 267989 (QUAREM) and by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award). The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 278410, and from The Israel Science Foundation (grant no. 1229/10).

* Corresponding author.
E-mail addresses: guy.avni@ist.ac.at (G. Avni), orna@cs.huji.ac.il (O. Kupferman).

https://doi.org/10.1016/j.tcs.2017.11.001
0304-3975/

open setting [32], for specifications in LTL) as well as the intricacy of the algorithms in the open setting – the traditional approach involves determinization of automata on infinite words [36] and a solution of parity games [24].

We find the argument about the algorithmic challenge less compelling. First, experience with verification shows that even nonelementary algorithms can be practical, since the worst-case complexity does not arise often. For example, while the model-checking problem for specifications in second-order logic has nonelementary complexity, the model-checking tool MONA [17] successfully verifies many specifications given in second-order logic. Furthermore, in some sense, synthesis is not harder than verification: the complexity of synthesis is given with respect to the specification only, whereas the complexity of verification is given with respect to the specification and the system, which is typically much larger than the specification. About the intercity of the algorithms, in the last decade we have seen quite many alternatives to the traditional approach – Safraless algorithms that avoid determinization and parity games, and reduce synthesis to problems that are simpler and are amenable to optimizations and symbolic implementations [20,27,29].

The arguments about the conceptual and methodological reasons are more compelling. We see here three main challenges, relevant in both the closed and open settings. First, unlike verification, where a specification can be decomposed into sub-specifications, each can be checked independently, in synthesis the starting point is one comprehensive specification. This inability to decompose or evolve the specification is related to the second challenge. In practice, we rarely construct systems from scratch or from one comprehensive specification. Rather, systems are constructed from existing components. This is true for both hardware systems, where we see IP cores or design libraries, and software systems, where web APIs and libraries of functions and objects are common. Third, while in verification we only automate the check of the system, automating its design is by far more risky and unpredictable – there are typically many ways to satisfy a satisfiable or realizable specification, and designers will be willing to give up manual design only if they can count on the automated synthesis tool to construct systems of comparable quality. Traditional synthesis algorithms do not attempt to address the quality issue.

In this paper we continue earlier efforts to cope with the above conceptual challenges. Our contribution extends both the setting and the results of earlier work. The realization that design of systems proceeds by composition of underlying components is not new to the verification community. For example, [21] proposed a framework for component-based modeling that uses an abstract layered model of components, and [14] initiated a series of works on interface theories for component-based design, possibly with a reuse of components in a library [15]. The need to consider components is more evident in the context of software, where, for example, recursion is possible, so components have to be equipped with mechanisms for call and return [4]. The setting and technical details, however, are different from these in the synthesis problem we consider here. The closer to our work here is [30], which studied LTL synthesis from reusable component libraries. Lustig and Vardi studied two notions of component composition. In the first notion, termed data-flow composition, components are cascaded so that the outputs of one component are fed to other components. In the second notion, termed control-flow composition, the composition is flat and control flows among the different components. The second notion, which turns out to be the decidable one [30], is particularly suitable for modeling web-service orchestration, where users are typically offered services and interact with different parties [3].

Let us turn now to the quality issue. Traditional formal methods are based on a Boolean satisfaction notion: a system satisfies, or not, a given specification. The richness of today's systems, however, calls for a *multi-valued* approach, where different systems are evaluated not just according to their correctness but also according to different quality measures. In order to capture a wide set of scenarios in practice, we associate with each component in the library two costs: a *quality cost* and a *construction cost*. The quality cost concerns the structural quality of the component and is paid each time the component is used. It refers to properties like the size of the component or its security level. The construction cost is the cost of adding the component to the library. Thus, a design that uses a component pays its construction cost once. When several designs use the same component, they share its construction cost. This corresponds to real-life scenarios, where users pay, for example, for web-services, and indeed their price is influenced by the market demand.

In [5], the authors study the problem of synthesizing a *hierarchical* system from a library of components that satisfies a specification while attempting to find a succinct system. They assume that rather than one specification, the input is a sequence of specifications $\phi_1, \ldots, \phi_m$ that attempt to guide the synthesis. The construction is then incremental. At step i, a component that satisfies ϕ_i is added to the library. The component C_m is then output as the final system.

We study synthesis from component libraries with costs in the closed and open settings. In both settings, the specification is given by means of a deterministic automaton $\mathcal{S}$ on finite words (DFA).[1] In the closed setting, the specification is a regular language over some alphabet Σ and the library consists of box-DFAs (that is, DFAs with exit states) over Σ. In the open setting, the specification $\mathcal{S}$ is over sets I and O of input and output signals, and the library consists of box-I/O-transducers. The boxes are black, in the sense that a design that uses components from the library does not see Σ (or $I \cup O$) nor it sees the behavior inside the components. Rather, the mode of operation is as in the control-flow composition of [30]: the design gives control to one of the components in the library. It then sees only the exit state through which the component completes its computation and relinquishes control. Based on this information, the design decides which component gets control next, and so on.

[1] It is possible to extend our results to specifications in LTL. We prefer to work with deterministic automata, as this setting isolates the complexity and technical challenges of the design problem and avoids the domination of the doubly-exponential complexity of going from LTL to deterministic automata.

In more technical details, the synthesis problem gets as input the specification $\mathcal{S}$ as well as a library $\mathcal{L}$ of components $\mathcal{B}_1, \ldots, \mathcal{B}_n$. The goal is to return a correct design – a transducer $\mathcal{D}$ that reads the exit states of the components and outputs the next component to gain control. In the closed setting, correctness means that the language over Σ that is generated by the composition defined by $\mathcal{D}$ is equal to the language of $\mathcal{S}$. In the open setting, correctness means that the interaction of the composition defined by $\mathcal{D}$ with all input sequences generates a computation over $I \cup O$ that is in the language of $\mathcal{S}$.

We first study the problem without cost and reduce it to the solution of a two-player safety game $\mathcal{G}_{\mathcal{S}}$. In the closed setting, the game is of full information and the problem can be solved in polynomial time. In the open setting, the flexibility that the design have in responding to different input sequences introduces partial information to the game, and the problem is EXPTIME-complete. We note that in [30], where the open setting was studied and the specification is given by means of an LTL formula, the complexity is 2EXPTIME-complete, thus one could have expected our complexity to be only polynomial. We prove, however, hardness in EXPTIME, showing that it is not just the need to transfer the LTL formula to a deterministic formalism that leads to the high complexity.

We then turn to integrate cost to the story. As explained above, there are two types of costs associated with each component $\mathcal{B}_i$ in $\mathcal{L}$. The first type, quality cost, can be studied for each design in isolation. We show that even there, the combinatorial setting is not simple. While for the closed setting an optimal design can be induced from a memoryless strategy of the designer in the game $\mathcal{G}_{\mathcal{S}}$, making the problem of finding an optimal design NP-complete, seeking designs of optimal cost may require sophisticated compositions in the open setting. In particular, we show that optimal designs may be exponentially larger than other correct designs,[2] and that an optimal design may not be induced by a memoryless strategy in $\mathcal{G}_{\mathcal{S}}$. We are still able to bound the size of an optimal transducer by the size of $\mathcal{G}_{\mathcal{S}}$, and show that the optimal synthesis problem is NEXPTIME-complete.

The second type of cost, namely construction cost, depends not only on choices made by the designer, but also on choices made by designers of other specifications that use the library. Indeed, recall that the construction cost of a component is shared by designers that use this component, with the share being proportional to the number of uses (a component can be used several times in a design). Hence, the setting gives rise to a *resource-allocation game* [34,35,18]. Unlike traditional resource-allocation games, where players' strategies are sets of resources, here each strategy is a multiset – the components a designer needs. As has been the case in [8], the setting of multisets makes the game less stable. We show that the game is not guaranteed to have a *Nash Equilibrium* (NE), and that the problem of deciding whether an NE exists is Σ_2^P-complete. We then turn to the more algorithmic related problems and show that the problems of finding an optimal design given the choices of the other designers (a.k.a. the *best-response* problem, in algorithmic game theory) and of finding designs that minimize the total cost for all specifications (a.k.a. the *social optimum*) are both NP-complete.

Recently, in [9], the setting of synthesis from component libraries by multiple users has been considered also for a setting in which the costs of the components have *congestion* effects rather than *cost-sharing* as we study here. For example, components might model processors and cost can model performance. When many users use the same component, congestion is increased and performance is decreased.

While the cost model we describe above is suited for capturing some aspects of quality of systems, e.g., when the goal is to minimize the number of states in the system, many other aspects are *computation-based*, and they refer to the performance of the system. For example, in a system that issues grants upon requests, a goal of the designer can be to design a system that minimizes the waiting time for a grant once a request is received. In recent years, we see many efforts to extend the Boolean setting of specification formalisms to a multi-valued one, allowing the user to associate rich satisfaction values with computations [13,19,12,2,11]. A standard model for reasoning about such costs of computations is *weighted automata* [16]. The rich semantics of weighted automata makes reasoning about them very difficult. For example, not all weighted automata can be determinized, and in fact the problem of deciding whether a given weighted automaton has an equivalent deterministic one is open [31]. Likewise, the weighted-containment problem for weighted automata is known to be undecidable [28].

We extend our study to *lattice automata* [26], which assign to each word a value that is an element of some finite lattice. Specifically, we study an extension of the closed synthesis problem from component libraries to a setting in which the specification is given by a deterministic lattice automaton (LDFA, for short) and the components are box LDFAs. Thus, our goal is to compose the components in the library to an LDFA that is equivalent to the specification LDFA, where equivalence amounts to agreement on the values assigned to each word. While the set of possible values that an LDFA assigns is finite, circumventing many of the technical difficulties in general weighted automata, handling box LDFAs involves other technical difficulties. In particular, there is no canonical minimal LDFA for a given language [22]. This is problematic, as minimal DFAs play a key role in our solution to the design problem in the closed setting. We introduce a new type of LDFAs, namely *separable LDFAs*, in which every two states have a word that separates between them. We show how the synthesis problem in the latticed setting can then be solved in polynomial time using a similar ideas to these used in the Boolean setting.

[2] Recall that "optimal" here refers to the quality-cost function.

2. Preliminaries

Automata, transducers, and boxes A *deterministic finite automaton* (DFA, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$, where Σ is an alphabet, Q is a set of states, $\delta : Q \times \Sigma \to Q$ is a partial transition function, $q_0 \in Q$ is an initial states, and $F \subseteq Q$ is a set of accepting states. We extend δ to words in an expected way, thus $\delta^* : Q \times \Sigma^* \to Q$ is such that for $q \in Q$, we have $\delta^*(q, \epsilon) = q$ and for $w \in \Sigma^*$ and $\sigma \in \Sigma$, we have $\delta^*(q, w \cdot \sigma) = \delta(\delta^*(q, w), \sigma)$. When $q = q_0$, we sometimes omit it, thus $\delta^*(w)$ is the state that $\mathcal{A}$ reaches after reading w. We assume that all states are reachable from q_0, thus for every $q \in Q$ there exists a word $w \in \Sigma^*$ such that $\delta^*(w) = q$. We refer to the *size* of $\mathcal{A}$, denoted $|\mathcal{A}|$, as the number of its states.

The *run* of $\mathcal{A}$ on a word $w = w_1, \ldots w_n \in \Sigma^*$ is the sequence of states $r = r_0, r_1, \ldots, r_n$ such that $r_0 = q_0$ and for every $0 \leq i \leq n-1$ we have $r_{i+1} = \delta(r_i, w_{i+1})$. The run r is accepting iff $r_n \in F$. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words $w \in \Sigma^*$ such that the run of $\mathcal{A}$ on w is accepting, or, equivalently, $\delta^*(w) \in F$. For $q \in Q$, we denote by $L(\mathcal{A}^q)$ the language of the DFA that is the same as $\mathcal{A}$ only with initial state q. Note that since $\mathcal{A}$ is deterministic and δ is partial, there is at most one run of $\mathcal{A}$ on each word.

A *transducer* models an interaction between a system and its environment. It is similar to a DFA except that in addition to Σ, which is referred to as the input alphabet, denoted Σ_I, there is an output alphabet, denoted Σ_O, and rather than being classified to accepting or rejecting, each state is labeled by a letter from Σ_O.[3] Formally, a transducer is a tuple $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \delta, \nu \rangle$, where Σ_I is an input alphabet, Σ_O is an output alphabet, Q, q_0, and $\delta : Q \times \Sigma_I \to Q$ are as in a DFA, and $\nu : Q \to \Sigma_O$ is an output function. We require $\mathcal{T}$ to be *receptive*. That is, δ is complete, so for every input word $w \in \Sigma_I^*$, there is a run of $\mathcal{T}$ on w. Consider an input word $w = w_1, \ldots, w_n \in \Sigma_I^*$. Let $r = r_0, \ldots, r_n$ be the run of $\mathcal{T}$ on w. The *computation of $\mathcal{T}$ in w* is then $\sigma_1, \ldots, \sigma_n \in (\Sigma_I \times \Sigma_O)^*$, where for $1 \leq i \leq n$, we have $\sigma_i = \langle w_i, \nu(r_{i-1}) \rangle$. We define the language of $\mathcal{T}$, denoted $L(\mathcal{T})$, as the set of all its computations. For a specification $L \subseteq (\Sigma_I \times \Sigma_O)^*$, we say that $\mathcal{T}$ *realizes* L iff $L(\mathcal{T}) \subseteq L$. Thus, no matter what the input sequence is, the interaction of $\mathcal{T}$ with the environment generates a computation that satisfies the specification. For two words $u \in \Sigma_I^*$ and $v \in \Sigma_O^*$ of length n we define the *product* of the two words, denoted $u \oplus v$, as $w = w_1 \ldots w_n \in (\Sigma_I \times \Sigma_O)^*$, where, for $1 \leq i \leq n$, we have $w_i = \langle u_i, v_i \rangle$.

By adding *exit states* to DFAs and transducers, we can view them as components from which we can compose systems. Formally, we consider two types of components. Closed components are modeled by *box-DFAs* and open components are modeled by *box-transducers*. A box-DFA augments a DFA by a set of exit states. Thus, a box-DFA is a tuple $\langle \Sigma, Q, \delta, q_0, F, E \rangle$, where $E \subseteq Q$ is a nonempty set of exit states. There are no outgoing transitions from an exit state. Also, the initial state cannot be an exit state and exit states are not accepting. Thus, $q_0 \notin E$ and $F \cap E = \emptyset$. Box-transducers are defined similarly, and their exit states are not labeled, thus $\nu : Q \setminus E \to \Sigma_O$.

Component libraries A *component library* is a collection of boxes $\mathcal{L} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$. We say that $\mathcal{L}$ is a *closed library* if the boxes are box-DFAs, and is an *open library* if the boxes are box-transducers. Let $[n] = \{1, \ldots, n\}$. In the first case, for $i \in [n]$, let $\mathcal{B}_i = \langle \Sigma, C_i, \delta_i, c_i^0, F_i, E_i \rangle$. In the second case, $\mathcal{B}_i = \langle \Sigma_I, \Sigma_O, C_i, \delta_i, c_i^0, \nu_i, E_i \rangle$. Note that all boxes in $\mathcal{L}$ share the same alphabet (input and output alphabet, in the case of transducers). We assume that the states of the components are disjoint, thus for every $i \neq j \in [n]$, we have $C_i \cap C_j = \emptyset$. We use the following abbreviations $\mathcal{C} = \bigcup_{i \in [n]} C_i$, $\mathcal{C}_0 = \bigcup_{i \in [n]} \{c_i^0\}$, $\mathcal{F} = \bigcup_{i \in [n]} F_i$, and $\mathcal{E} = \bigcup_{i \in [n]} E_i$. We define the *size* of $\mathcal{L}$ as $|\mathcal{C}|$.

We start by describing the intuition for composition of closed libraries. A *design* is a recipe to compose the components of a library $\mathcal{L}$ (allowing multiple uses) into a DFA. A run of the design on a word starts in an initial state of one of the components in $\mathcal{L}$. We say that this component has the initial *control*. When a component is in control, the run uses its states, follows its transition function, and if the run ends, it is accepting iff it ends in one of the components' accepting states. A component relinquishes control when the run reaches one of its exit states. It is then the design's duty to assign control to the next component, which gains control through its initial state.

Formally, a design is a transducer $\mathcal{D}$ with input alphabet $\mathcal{E}$ and output alphabet $[n]$. We can think of $\mathcal{D}$ as running beside the components. When a component reaches an exit state e, then $\mathcal{D}$ reads the input letter e, proceeds to its next state, and outputs the index of the component to gain control next. Note that $\mathcal{D}$ does not read the alphabet Σ and has no information about the states that the component visits. It only sees which exit state has been reached.

Consider a design $\mathcal{D} = \langle \mathcal{E}, [n], D, \delta, d^0, \nu \rangle$ and a closed library $\mathcal{L}$. We formalize the behavior of $\mathcal{D}$ by means of the *composition DFA* $\mathcal{A}_\mathcal{D}$ that simulates the run of $\mathcal{D}$ along with the runs of the box-DFAs. Formally, $\mathcal{A}_\mathcal{D} = \langle \Sigma, Q_\mathcal{D}, \delta_\mathcal{D}, q_\mathcal{D}^0, F_\mathcal{D} \rangle$ is defined as follows. The set of states $Q_\mathcal{D} \subseteq (\mathcal{C} \setminus \mathcal{E}) \times D$ consists of pairs of a *component state* from $\mathcal{C}$ and an *design state* from S. The component states are consistent with ν, thus $Q_\mathcal{D} = \bigcup_{i \in [n]} (C_i \setminus E_i) \times \{d : \nu(d) = i\}$. In exit states, the composition immediately moves to the initial state of the next component, which is why the component states of $\mathcal{A}_\mathcal{D}$ do not include $\mathcal{E}$. Consider a state $\langle c, d \rangle \in Q_\mathcal{D}$ and a letter $\sigma \in \Sigma$. Let $i \in [n]$ be such that $c \in C_i$. When a run of $\mathcal{A}_\mathcal{D}$ reaches the state $\langle c, d \rangle$, the component $\mathcal{B}_i$ is in control. Recall that c is not an exit state. Let $c' = \delta_i(c, \sigma)$. If $c' \notin E_i$, then $\mathcal{B}_i$ does not relinquish control after reading σ and $\delta_\mathcal{D}(\langle c, d \rangle, \sigma) = \langle c', d \rangle$. If $c' \in E_i$, then $\mathcal{B}_i$ relinquishes control through c', and it is the design's task to choose the next component to gain control. Let $d' = \delta(d, c')$ and let $j = \nu(d')$. Then, $\mathcal{B}_j$ is the next component to gain control (possibly $j = i$). Accordingly, we advance $\mathcal{D}$ to d' and continue to the initial state of $\mathcal{B}_j$. Formally, $\delta_\mathcal{D}(\langle c, d \rangle, \sigma) = \langle c_j^0, d' \rangle$. (Recall that $c_j^0 \notin E_j$, so the new state is in $Q_\mathcal{D}$.) Note also that a visit in c' is skipped. The component

[3] These transducers are sometimes referred to as *Moore machines*.

that gains initial control is chosen according to $\nu(d^0)$. Thus, $q^0_{\mathcal{D}} = \langle c^0_j, d^0\rangle$, where $j = \nu(d^0)$. Finally, the accepting states of $\mathcal{A}_{\mathcal{D}}$ are these in which the component state is accepting, thus $F_{\mathcal{D}} = \mathcal{F} \times D$.

The definition of a composition for an open library is similar. There, the composition is a transducer $\mathcal{T}_{\mathcal{D}} = \langle \Sigma_I, \Sigma_O, Q_{\mathcal{D}}, \delta_{\mathcal{D}}, q^0_{\mathcal{D}}, \nu_{\mathcal{D}}\rangle$, where $Q_{\mathcal{D}}$, $q^0_{\mathcal{D}}$, and $\delta_{\mathcal{D}}$ are as in the closed setting, except that $\delta_{\mathcal{D}}$ reads letters in Σ_I, and $\nu_{\mathcal{D}}(\langle c, d\rangle) = \nu_i(c)$, for $i \in [n]$ such that $c \in C_i$.

Consider a closed-library $\mathcal{L}$, a design $\mathcal{D}$, and the run r of $\mathcal{A}_{\mathcal{D}}$ on $w = w_0 \cdots w_l$. We partition w according to positions in which control is transferred among components. Equivalently, positions in which r skips visits in exit states. Thus, $w = y_0 \cdots y_k$ is such that for all $0 \leq i < k$, we have that $y_i \in \Sigma^+$ and the composition $\mathcal{A}_{\mathcal{D}}$ takes a transfer transition exactly when it reads the last letter of y_i. An exception is y_k, which may be empty (this happens when r ends upon entering the last component to gain control). We then say that w is *suffix-less*. The definitions in the open setting are similar.

3. The design problem

The *design problem* gets as input a component library $\mathcal{L}$ and a specification that is given by means of a DFA $\mathcal{S}$. The problem is to decide whether there exists a correct design for $\mathcal{S}$ using the components in $\mathcal{L}$. In the closed setting, a design $\mathcal{D}$ is correct if $L(\mathcal{A}_{\mathcal{D}}) = L(\mathcal{S})$. In the open setting, $\mathcal{D}$ is correct if the transducer $\mathcal{T}_{\mathcal{D}}$ realizes $\mathcal{S}$. Our solution to the design problem reduces it to the problem of finding the winner in a turn-based two-player game, defined below.

A *turn-based two-player game* is played on an arena $\langle V, \Delta, V_0, \alpha\rangle$, where $V = V_1 \cup V_2$ is a set of vertices that are partitioned between Player 1 and Player 2, $\Delta \subseteq V \times V$ is a set of directed edges, $V_0 \subseteq V$ is a set of initial vertices, and α is an objective for Player 1, specifying a subset of V^ω. We consider here *safety games*, where $\alpha \subseteq V$ is a set of vertices that are *safe* for Player 1. The game is played as follows. Initially, Player 1 places a token on one of the vertices in V_0. Assume the token is placed on a vertex $v \in V$ at the beginning of a round. The player that owns v is the player that moves the token to the next vertex, where the legal vertices to continue to are $\{v' \in V : \langle v, v'\rangle \in \Delta\}$. The outcome of the game is a *play* $\pi \in V^\omega$. The play is winning for Player 1 if for every $i \geq 1$, we have $\pi_i \in \alpha$. Otherwise, Player 2 wins.

A *strategy* for Player i, for $i \in \{1, 2\}$, is a recipe that, given a prefix of a play, tells the player what his next move should be. Thus, it is a function $f_i : V^* \cdot V_i \to V$ such that for every play $\pi \cdot v \in V^*$ with $v \in V_i$, we have $\langle v, f_i(\pi \cdot v)\rangle \in \Delta$. Since Player 1 moves first, we require that $f_1(\epsilon)$ is defined and is in V_0. For strategies f_1 and f_2 for players 1 and 2 respectively, the play $out(f_1, f_2) \in V^\omega$ is the unique play that is the outcome of the game when the players follow their strategies. A strategy f_i for Player i is *memoryless* if it depends only in the current vertex, thus it is a function $f_i : V_i \to V$.

A strategy is *winning* for a player if by using it he wins against every strategy of the other player. Formally, a strategy f_1 is winning for Player 1 iff for every strategy f_2 for Player 2, Player 1 wins the play $out(f_1, f_2)$. The definition for Player 2 is dual. It is well known that safety games are *determined*, namely, exactly one player has a winning strategy, and admits *memoryless* strategies, namely, Player i has a winning strategy iff he has a memoryless winning strategy. Deciding the winner of a safety game can done in linear time.

Solving the design problem We describe the intuition of our solution for the design problems. Given a library $\mathcal{L}$ and a specification $\mathcal{S}$ we construct a safety game $\mathcal{G}_{\mathcal{S}}$ such that Player 1 wins $\mathcal{G}_{\mathcal{S}}$ iff there is a correct design for $\mathcal{S}$ using the components in $\mathcal{L}$. Intuitively, Player 1's goal is to construct a correct design, thus he chooses the components to gain control. Player 2 challenges the design that Player 1 chooses, thus he chooses a word (over Σ in the closed setting and over $\Sigma_I \times \Sigma_O$ in the open setting) and wins if his word is a witness for the incorrectness of Player 1's design.

Closed designs The input to the closed-design problem is a closed-library $\mathcal{L}$ and a DFA $\mathcal{S}$ over the alphabet Σ. The goal is to find a correct design $\mathcal{D}$. Recall that $\mathcal{D}$ is correct if the DFA $\mathcal{A}_{\mathcal{D}}$ that is constructed from $\mathcal{L}$ using $\mathcal{D}$ satisfies $L(\mathcal{A}_{\mathcal{D}}) = L(\mathcal{S})$. We assume that $\mathcal{S}$ is the minimal DFA for the language $L(\mathcal{S})$ as if it is not minimal, we can minimize it in polynomial time.

Theorem 3.1. *The closed-design problem can be solved in polynomial time.*

Proof. Given a closed-library $\mathcal{L}$ and a DFA $\mathcal{S} = \langle \Sigma, S, \delta_{\mathcal{S}}, s^0, F_{\mathcal{S}}\rangle$, we describe a safety game $\mathcal{G}_{\mathcal{S}}$ such that Player 1 wins $\mathcal{G}_{\mathcal{S}}$ iff there is a design of $\mathcal{S}$ using components from $\mathcal{L}$. Recall that $\mathcal{L}$ consists of box-DFAs $\mathcal{B}_i = \langle \Sigma, C_i, \delta_i, c^0_i, F_i, E_i\rangle$, for $i \in [n]$, and that we use $\mathcal{C}$, $\mathcal{C}_0$, $\mathcal{E}$, and $\mathcal{F}$ to denote the union of all states, initial states, exit states, and accepting states in all the components of $\mathcal{L}$. The number of vertices in $\mathcal{G}_{\mathcal{S}}$ is $|(\mathcal{C}_0 \cup \mathcal{E}) \times S|$ and it can be constructed in polynomial time. Since solving safety games can be done in linear time, the theorem follows.

We define $\mathcal{G}_{\mathcal{S}} = \langle V, E, V_0, \alpha\rangle$. First, $V = (\mathcal{C}_0 \cup \mathcal{E}) \times S$ and $V_0 = \mathcal{C}_0 \times \{s^0\}$. Recall that Player 1 moves when it is time to decide the next (or first) component to gain control. Accordingly, $V_1 = \mathcal{E} \times S$. Also, Player 2 challenges the design suggested by Player 1 and chooses the word that is processed in a component that gains control, so $V_2 = \mathcal{C}_0 \times S$.

Consider a vertex $\langle e, s\rangle \in V_1$. Player 1 selects the next component to gain control. This component gains control through its initial state. Accordingly, E contains edges $\langle\langle e, s\rangle, \langle c^0_i, s\rangle\rangle$, for every $i \in [n]$. Note that since no letter is read when control is passed, we do not advance the state in $\mathcal{S}$. Consider a vertex $v = \langle c^0_i, s\rangle \in V_2$. Player 2 selects the word that is read in the component $\mathcal{B}_i$, or equivalently, he selects the exit state from which $\mathcal{B}_i$ relinquishes control. Thus, E contains an edge $\langle\langle c^0_i, s\rangle, \langle e, s'\rangle\rangle$ iff there exists a word $u \in \Sigma^*$ such that $\delta^*_i(u) = e$ and $\delta^*_{\mathcal{S}}(s, u) = s'$.

We now turn to define the winning condition. All the vertices in V_1 are in α. A vertex $v \in V_2$ is not in α if it is possible to extend the word traversed for reaching v to a witness for the incorrectness of $\mathcal{D}$. Accordingly, a vertex $\langle c_i^0, s\rangle$ is not in α if one of the following holds. First ("the suffix witness"), there is a finite word that is read inside the current component and witnesses the incorrectness. Formally, there is $u \in \Sigma^*$ such that $\delta_i^*(u) \in F_i$ and $\delta_{\mathcal{S}}^*(s,u) \notin F_{\mathcal{S}}$, or $\delta_i^*(u) \in C_i \setminus (F_i \cup E_i)$ and $\delta_{\mathcal{S}}^*(s,u) \in F_{\mathcal{S}}$. Second ("the infix witness"), there are two words that reach the same exit state of the current component yet the behavior of $\mathcal{S}$ along them is different. Formally, there exist words $u, u' \in \Sigma^*$ such that $\delta_i^*(u) = \delta_i^*(u') \in E_i$ and $\delta_{\mathcal{S}}^*(s,u) \neq \delta_{\mathcal{S}}^*(s,u')$. Intuitively, the minimality of $\mathcal{S}$ enables us to extend either u or u' to an incorrectness witness. Note that if there is no suffix witness from $\langle c_i^0, s\rangle$, Player 2 cannot show that $\mathcal{D}$ is incorrect without $\mathcal{B}_i$ relinquishing control, thus his only hope is to choose a word u after which $\mathcal{B}_i$ relinquishes control. Moreover, if there is no infix witness, then Player 1 is able to track the run of $\mathcal{S}$ on u, which leads us to a Player 1 vertex of the form $\langle e, s'\rangle$. Given $\mathcal{L}$ and $\mathcal{S}$, the game $\mathcal{G}_{\mathcal{S}}$ can be constructed in polynomial time.

We claim that there is a correct design $\mathcal{D}$ iff Player 1 wins $\mathcal{G}_{\mathcal{S}}$. Assume first that there is a correct design $\mathcal{D} = \langle \mathcal{E}, [n], D, \delta, d^0, \nu\rangle$, thus $L(\mathcal{A}_{\mathcal{D}}) = L(\mathcal{S})$. We construct a winning strategy $f_{\mathcal{D}}$ for Player 1. The strategy $f_{\mathcal{D}}$ proceeds like $\mathcal{D}$. First, $f_{\mathcal{D}}(\epsilon) = \langle c_i^0, s^0\rangle$, with $i = \nu(d^0)$. Then, for a finite play π, let $\langle e_0, s_0\rangle, \langle e_1, s_1\rangle \ldots \langle e_m, s_m\rangle$ be its projection on V_1. Thus, $e_0, \ldots, e_m \in \mathcal{E}$ and $s_0, \ldots, s_m \in S$. We define $f_{\mathcal{D}}(\pi) = \langle c_i^0, s_m\rangle$, for $i = \nu(\delta_{\mathcal{D}}^*(e_0, \ldots, e_m))$.

We claim that $f_{\mathcal{D}}$ is a winning strategy. Assume towards contradiction that there is a Player 2 strategy f_2 that wins against $f_{\mathcal{D}}$. Let $\pi = out(f_{\mathcal{D}}, f_2)$, let $\langle c_i^0, s\rangle$ be its first vertex that is not in α, and let w be the word in Σ^* that Player 2 follows along the prefix of π that reaches $\langle c_i^0, s\rangle$. Finally, let $d \in D$ be such that $\langle c_i^0, d\rangle$ is the state that $\mathcal{A}_{\mathcal{D}}$ reaches when it reads w. Since we define $f_{\mathcal{D}}$ to agree with $\mathcal{D}$, then the component state of this state in $\mathcal{A}_{\mathcal{D}}$ is indeed c_i^0.

We distinguish between two cases. First, if $\langle c_i^0, s\rangle$ exits α because of a suffix witness, let $u \in \Sigma^*$ be such that $\delta_i^*(u) \in F_i$ and $\delta_{\mathcal{S}}^*(s,u) \notin F_{\mathcal{S}}$. (The case where $\delta_i^*(u) \notin (F_i \cup E_i)$ and $\delta_{\mathcal{S}}^*(s,u) \in F_{\mathcal{S}}$ is similar). Let $c = \delta_i^*(u)$. The run of $\mathcal{A}_{\mathcal{D}}$ on $w \cdot u$ ends in the state $\langle \delta_i^*(u), s\rangle$. Since $\delta_i^*(u) \in F_i$, we have $w \cdot u \in L(\mathcal{A}_{\mathcal{D}})$. By the definition of E, we have that $\delta_{\mathcal{S}}^*(w) = s$. Since $\delta_{\mathcal{S}}^*(s,u) \notin F_{\mathcal{S}}$, we have $w \cdot u \notin L(\mathcal{S})$. Thus, $L(\mathcal{A}_{\mathcal{D}}) \neq L(\mathcal{S})$, and we reach a contradiction to the correctness of $\mathcal{D}$.

In the second case, of an infix witness, there exist words $u, u' \in \Sigma^*$ such that $\delta_i^*(u) = \delta_i^*(u') \in E_i$ and $\delta_{\mathcal{S}}^*(s,u) = p \neq p' = \delta_{\mathcal{S}}^*(s,u')$. Since $\mathcal{S}$ is a minimal DFA for $L(\mathcal{S})$, we have $L(\mathcal{S}^p) \neq L(\mathcal{S}^{p'})$. Thus, wlog, there is a word $z \in L(\mathcal{S}^p) \setminus L(\mathcal{S}^{p'})$. Recall that $\delta_{\mathcal{D}}^*(w) = \langle c_i^0, d\rangle$. Let $d' = \delta_{\mathcal{D}}(d, e)$ and $j = \nu(d')$. Since $\delta_{\mathcal{D}}^*(w \cdot u) = \delta_{\mathcal{D}}^*(w \cdot u') = \langle c_j^0, d'\rangle$, we have $\delta_{\mathcal{D}}^*(w \cdot u \cdot z) = \delta_{\mathcal{D}}^*(w \cdot u' \cdot z)$. Thus, $w \cdot u \cdot z \in L(\mathcal{A}_{\mathcal{D}})$ iff $w \cdot u' \cdot z \in L(\mathcal{A}_{\mathcal{D}})$. However, $w \cdot u \cdot z \in L(\mathcal{S})$ and $w \cdot u' \cdot z \notin L(\mathcal{S})$. Thus, we reach a contradiction to the correctness of $\mathcal{D}$, and we are done.

Assume now that Player 1 wins the game $\mathcal{G}_{\mathcal{S}}$. Let f be a memoryless winning strategy for Player 1. We construct a correct design $\mathcal{D}_f$ from f. Note that all the successors of a vertex in V_1 are in V_2. Thus, $f : V_1 \to V_2$. We define $\mathcal{D}_f = \langle \mathcal{E}, [n], D, \delta, s^0, \nu\rangle$ as follows. First, $D = V_2 = \mathcal{C}_0 \times S$. Consider a state $v = \langle c_i^0, s\rangle \in V_2 \cap \alpha$. Recall that c_i^0 is the initial state of the component $\mathcal{B}_i$. Since $v \in \alpha$, the lack of an infix witness implies that for every exit state $e \in E_i$ there is exactly one state $s' \in S$ such that $\langle\langle c_i^0, s\rangle, \langle e, s'\rangle\rangle \in E$. We define $\delta(v, e) = f(\langle e, s'\rangle)$. Note that if $v \notin \alpha$ or $e \notin E_i$, then we can define $\delta(v, e)$ arbitrarily. The labeling function ν is defined as expected, with $\nu(\langle c_i^0, s\rangle) = i$.

We prove that $\mathcal{D}_f$ is a correct design. Assume towards contradiction that there is a word $w \in L(\mathcal{A}_{\mathcal{D}_f}) \setminus L(\mathcal{S})$. The case where $w \in L(\mathcal{S}) \setminus L(\mathcal{A}_{\mathcal{D}_f})$ is similar. Consider the run r of $\mathcal{A}_{\mathcal{D}_f}$ on w. Let $\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_m} \in \mathcal{L}^*$ be sequence of components that r traverses and $e_{i_1}, \ldots, e_{i_{m-1}} \in \mathcal{E}^*$ be the corresponding exit states. Let $y_1, \ldots, y_m$ be the partition of w according to $\mathcal{D}$. Thus, for $1 \leq j < m$, we have $y_i \in \Sigma^+$ and $\delta_{i_j}^*(y_i) = e_{i_j} \in E_{i_j}$, and $y_m \in \Sigma^*$. Since $w \in L(\mathcal{A}_{\mathcal{D}_f})$, we have $\delta_{i_m}^*(w_m) \in F_{i_m}$. Note that the word $y_1 \cdots y_{m-1} \in \Sigma^*$ is suffix-less, thus $\delta_{\mathcal{D}_f}^*(y_1 \cdot \ldots \cdot y_{m-1}) = \langle c_{i_m}^0, d\rangle$ for some $d \in D$ with $\nu(d) = i_m$. Since we defined $\mathcal{D}_f$ to agree with f on the components that gain control, the finite play π that is the outcome of the game when Player 1 plays f and Player 2 chooses the exit states $e_{i_1}, \ldots, e_{i_{m-1}}$ reaches the Player 2 vertex $v = \langle c_{i_m}^0, s\rangle \in V_2$, for $s \in S$ such that $\delta_{\mathcal{S}}^*(y_1 \cdots y_{m-1}) = s$. We claim that $v \notin \alpha$. Indeed, $\delta_i^*(y_m) \in F_i$ and since $w \notin L(\mathcal{S})$, we have $y_m \notin L(\mathcal{S}^s)$. Thus, π is a winning play for Player 2, contradicting our assumption that f is a winning strategy, and we are done. □

We show a tight lower bound.

Theorem 3.2. *The closed-design problem is PTIME-hard.*

Proof. We show a reduction from the problem of deciding the winner in a safety game, which is known to be PTIME-complete [23]. Given a safety game $\mathcal{G} = \langle V, \Delta, V_0, \alpha\rangle$, we construct a component library $\mathcal{L}$ and a specification $\mathcal{S}$ such that Player 1 wins $\mathcal{G}$ iff there is a correct design $\mathcal{D}$; i.e., $L(\mathcal{A}_{\mathcal{D}}) = L(\mathcal{S})$. For simplicity, we assume that $\mathcal{G}$ has only one initial vertex, i.e., $|V_0| = 1$, that it belongs to Player 1, and that the players alternate turns, thus $\Delta \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$. The problem of deciding the winner is PTIME-hard already for this fragment.

We define $\Sigma = \Delta$. For each edge $e = \langle v_1, v_2\rangle \in (V_1 \times V_2)$, we have a component $\mathcal{B}_e$. Intuitively, a design corresponds to a Player 1 strategy and assigning control to $\mathcal{B}_e$ corresponds to a Player 1 move from v_1 to v_2. We describe the specification before showing how to simulate Player 2's moves.

The language of $\mathcal{S}$ consists of the finite paths of $\mathcal{G}$. This is achieved by constructing an automaton from $\mathcal{G}$ by labeling each edge $e \in \Delta$ with the letter e, setting the initial state to be the initial vertex of $\mathcal{G}$, and setting all states to be accept-

ing. Note that $\mathcal{S}$ is deterministic since edges have unique names. Further note that the run on a word w that does not correspond to a path in $\mathcal{G}$ gets stuck and thus w is rejected.

We construct a component $\mathcal{B}_e$ so that it simulates a move of Player 2. The moves of Player 2 are modeled by a choice of word that a component processes, where we think of Player 2's goal as trying to show that the design is incorrect. Let $e = \langle v_1, v_2\rangle \in (V_1 \times V_2)$. There are two special vertices in $\mathcal{B}_e$, denoted v_1^e and v_2^e, where v_1^e is the initial state of $\mathcal{B}_e$. Suppose $\mathcal{B}_e$ gains control after reading a word $w \in \Sigma^*$. The run of $\mathcal{S}$ on w reaches the state v_1. If Player 2 chooses the letter $e' \in \Delta$ such that e' is not an outgoing edge from v_1, both the run in $\mathcal{S}$ and in $\mathcal{B}_e$ get stuck and reject. If he chooses an outgoing edge e' from v_1 such that $e' \neq e$, then $\mathcal{B}_e$ proceeds to a state from which the words that are accepted are exactly the finite paths that start from v_2 in $\mathcal{G}$. In other words, by selecting e', Player 2 has no way of winning. This is achieved by placing, for every such e', a copy $\mathcal{S}_{e'}$ of $\mathcal{S}$ in $\mathcal{B}_e$, where the initial state of $\mathcal{S}_{e'}$ is the target of e'. Finally, if Player 2 chooses e, then $\mathcal{B}_e$ moves to the state v_2^e that corresponds to the Player 2 vertex v_2. From there, Player 2 can select any letter $e' \in \Delta$ such that $e' = \langle v_2, v_1'\rangle \in (V_2 \times V_1)$. Upon reading e', $\mathcal{B}_e$ relinquishes control from the exit state v_1'. In other words, Player 2 moves the token from v_2 to the vertex v_1'.

We show a correspondence between outcomes in $\mathcal{G}$ and runs of a compositional system constructed from $\mathcal{L}$. Consider strategies f_1 and f_2 for Players 1 and 2, respectively, and let $out(f_1, f_2) = v_1, v_2, \ldots, v_m$ for an even m. There is a clear one-to-one correspondence between designs and Player 1 strategies in $\mathcal{G}$. Let $\mathcal{D}$ be the design that corresponds to f_1. We construct a word w such that the run of $\mathcal{A}_\mathcal{D}$ on w visits the states $v_1^{e_1}, v_2^{e_1}, \ldots, v_{m-1}^{e_{m/2}}, v_m^{e_{m/2}}$, for some $e_1, \ldots, e_{m/2}$. We describe w inductively. Recall that Player 1 moves first. Suppose w is defined before $\mathcal{B}_e$ gains control, where $e = \langle v_1, v_2\rangle$. Then, the next letter is e followed by $\langle v_2, f(v_2)\rangle$. Our definition of the components implies that the run of $\mathcal{A}_\mathcal{D}$ indeed has the form above. For the other direction, consider a design $\mathcal{D}$. Every run of $\mathcal{A}_\mathcal{D}$ that has the form $v_1^{e_1}, v_2^{e_1}, \ldots, v_{m-1}^{e_{m/2}}, v_m^{e_{m/2}}$, for some $e_1, \ldots, e_{m/2}$, easily corresponds to a Player 2 strategy.

In order to conclude the construction, we need to take care of the safety objective of Player 1. Consider a vertex v in $\mathcal{G}$ that is not safe. Let $e \in \Delta$ be such that v^e is in the component $\mathcal{B}_e$. Then, the state v^e in $\mathcal{B}_e$ is rejecting. All other states in the components are accepting. The correspondence above implies that a run that visits v^e corresponds to an outcome that visits v. The latter is losing for Player 1 and the former corresponds to a word that is rejected by $\mathcal{A}_\mathcal{D}$ and is accepted by $\mathcal{S}$, and we are done. □

Open designs We continue to study the open setting. Recall that there, the input is a DFA $\mathcal{S}$ over the alphabet $\Sigma_I \times \Sigma_O$ and an open library $\mathcal{L}$. The goal is to find a correct design $\mathcal{D}$ or return that no such design exists, where $\mathcal{D}$ is correct if the composition transducer $\mathcal{T}_\mathcal{D}$ realizes $L(\mathcal{S})$.

Lustig and Vardi [30] studied the design problem in a setting in which the specification is given by means of an LTL formula. They showed that the problem is 2EXPTIME-complete. Given an LTL formula one can construct a deterministic parity automaton that recognizes the language of words that satisfy the formula. The size of the automaton is doubly-exponential in the size of the formula. Thus, one might guess that the design problem in a setting in which the specification is given by means of a DFA would be solvable in polynomial time. We show that this is not the case and that the problem is EXPTIME-complete. As in [30], our upper bound is based on the ability to "summarize" the activity inside the components. Starting with an LTL formula, the solution in [30] has to combine the complexity involved in the translation of the LTL formula into an automaton with the complexity of finding a design, which is done by going throughout a universal word automaton that is expanded to a tree automaton. Starting with a deterministic automaton, our solution directly uses games. The interesting contribution, however, is the lower bound, showing that problem is EXPTIME-hard even when the specification is given by means of a deterministic automaton. We start with the upper bound.

Theorem 3.3. *The open-design problem is in EXPTIME.*

Proof. Given an open-library $\mathcal{L}$ and a DFA $\mathcal{S} = \langle \Sigma_I \times \Sigma_O, S, \delta_\mathcal{S}, s^0, F_\mathcal{S}\rangle$, we describe a safety game $\mathcal{G}_\mathcal{S}$ such that Player 1 wins $\mathcal{G}_\mathcal{S}$ iff there is a design for $\mathcal{S}$ using components from $\mathcal{L}$. The number of vertices in $\mathcal{G}_\mathcal{S}$ is exponential in S. Since solving safety games can be done in linear time, membership in EXPTIME follows.

We define $\mathcal{G}_\mathcal{S} = \langle V, E, V_0, \alpha\rangle$ as follows.[4] Recall that $\mathcal{C}$, $\mathcal{C}_0$, $\mathcal{E}$, and $\mathcal{F}$ are the union of all states, initial states, exit states, and accepting states in all the components of $\mathcal{L}$. We define $V = (\mathcal{C}_0 \cup \mathcal{E}) \times 2^S$ with $V_1 = \mathcal{E} \times 2^S$ and $V_2 = \mathcal{C}_0 \times 2^S$. Also, $V_0 = \mathcal{C}_0 \times \{\{s^0\}\}$. As in the closed-setting, Player 1 selects the components that gain control, thus for a vertex $\langle e, T\rangle \in V_1$ we have $\langle\langle e, T\rangle, \langle c_i^0, T\rangle\rangle \in E$, for every $c_i^0 \in \mathcal{C}_0$. Player 2 selects the word that is processed in the component, or equivalently, the exit state from which it relinquishes control, thus for a vertex $v = \langle c_i^0, T\rangle \in V_2$ we have $\langle\langle c_i^0, T\rangle, \langle e, T'\rangle\rangle \in E$ iff for every $s' \in T'$ there is a state $s \in T$ and a word $u \in \Sigma_I^*$ such that $\delta_i^*(u) = e$ and, assuming $w \in (\Sigma_I \times \Sigma_O)^*$ is the computation of $\mathcal{B}_i$ that corresponds to u, we have $\delta_\mathcal{S}^*(s, w) = s'$. Intuitively, Player 1 tracks the computation that takes place in $\mathcal{B}_i$ as best as he can. Taking the edge $\langle\langle c_i^0, T\rangle, \langle e, T'\rangle\rangle$ models the fact that, assuming the run of $\mathcal{S}$ can be in any one of the

[4] A different way to construct $\mathcal{G}_\mathcal{S}$ would be to go through a *partial-information* game (see Theorem 3.4) with vertices in $\mathcal{C} \times S$, where Player 1 cannot distinguish between vertices $\langle e, d\rangle$ and $\langle e, d'\rangle$ for $e \in \mathcal{E}$ and $d, d' \in S$. The game $\mathcal{S}$ is the corresponding game. We describe it directly, which also shows that the exponential dependency is only in S.

states in T, then Player 2 can choose an input word such that $\mathcal{B}_i$ relinquishes control from e and the run of $\mathcal{S}$ is at any one of the states in T'. Note that for $c_i^0 \in \mathcal{C}_0$, $T \in 2^S$, and $e \in \mathcal{E}$, there is at most one, nonempty, subset $T' \in 2^S$ such that $\langle\langle c_i^0, T\rangle, \langle e, T'\rangle\rangle \in E$. The set of vertices that are losing for Player 1 consists of states $\langle c_i^0, T\rangle \in V_2$ from which Player 2 can generate a suffix-witness to the incorrectness of the design. Formally, $\langle c_i^0, T\rangle \in \alpha$ iff there exists $u \in L(\mathcal{B}_i)$ and $s \in T$ such that $u \notin L(\mathcal{S}^s)$.

We claim that there is a correct design $\mathcal{D}$ iff Player 1 wins $\mathcal{G}_\mathcal{S}$. For the first direction, consider a correct design $\mathcal{D}$, thus $L(\mathcal{T}_\mathcal{D}) \subseteq L(\mathcal{S})$. We construct a winning Player 1 strategy $f_\mathcal{D}$. Recall that a design reads exit states and outputs components. Further recall that assuming the game does not end, a Player 2 move is a choice of an exit state. We define $f_\mathcal{D}$ so that it responds to Player 2's choice the same way $\mathcal{D}$ responds. We define $f_\mathcal{D}(\epsilon) = \langle c_i^0, \{\{s^0\}\}\rangle$ for $i = \nu(d^0)$, thus $f_\mathcal{D}$ and $\mathcal{D}$ assign initial control to the same component. Consider a finite play π and let $\langle e_1, T_1\rangle, \ldots, \langle e_m, T_m\rangle$ be the projection of π on V_2, thus $e_1, \ldots, e_m \in \mathcal{E}$ and $T_1, \ldots, T_m \in 2^S$. We define $f_\mathcal{D}(\pi) = \langle c_i^0, T_m\rangle$ where $\nu(\delta_\mathcal{D}^*(e_1, \ldots, e_m)) = i$.

We claim that $f_\mathcal{D}$ is a winning strategy. Assume towards contradiction that there is a Player 2 strategy f_w that wins against $f_\mathcal{D}$. Let $\pi = \langle c_{i_1}^0, \{s^0\}\rangle, \langle e_{i_1}, T_1\rangle, \langle c_{i_2}^0, T_1\rangle, \langle e_{i_2}, T_2\rangle, \ldots, \langle e_{i_{m-1}}, T_m\rangle, \langle c_{i_m}^0, T_m\rangle$ be the finite losing prefix of $out(f_\mathcal{D}, f_w)$, thus $\langle c_{i_m}^0, T_m\rangle \notin \alpha$. Since $\langle c_{i_m}^0, T_m\rangle \notin \alpha$ there is a state $s_m \in T_m$ and a word $w_m \in L(\mathcal{B}_i)$ such that $w_m \notin L(\mathcal{S}^{s_m})$. It is not hard to see that there are words $w_1, \ldots, w_{m-1} \in (\Sigma_I \times \Sigma_O)^*$ and states $s_1, \ldots, s_{m-1} \in S$ such that for $1 \leq j \leq m-1$ we have $w_j \in L(\mathcal{B}_{i_j})$, $\delta_{i_j}^*(c_{i_j}^0, w_j) = e_{i_j}$, $\delta_\mathcal{S}^*(w_1) = s_1$, and $\delta_\mathcal{S}^*(s_j, w_j) = s_{j+1}$. It is not hard to see that since we defined $f_\mathcal{D}$ to agree with $\mathcal{D}$, the components that gain control in the run of $\mathcal{T}_\mathcal{D}$ on $w = w_1 \cdot \ldots w_m$ are $\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_m}$, thus it is possible to prove by induction that $w \in L(\mathcal{T}_\mathcal{D})$. Moreover, the run of $\mathcal{S}$ on w is not accepting, thus $w \in L(\mathcal{T}_\mathcal{D}) \setminus L(\mathcal{S})$, and we reach a contradiction to the correctness of $\mathcal{D}$.

We continue to the second direction. Assume Player 1 wins the game $\mathcal{G}_\mathcal{S}$. Thus, he has a memoryless winning strategy $f_\mathcal{D}$ from which we construct a design $\mathcal{D}$. Intuitively, in $\mathcal{D}$, we skip exit states and proceed according to $f_\mathcal{D}$. Thus, the states of $\mathcal{D}$ are $V_2 = \mathcal{C}_0 \times 2^S$. Consider a state $\langle c_i^0, T\rangle \in V_2$, where recall that c_i^0 is the initial state of the component $\mathcal{B}_i \in \mathcal{L}$, and an exit state $e \in E_i$ of $\mathcal{B}_i$. Recall that there is a unique subset $T' \in 2^S$ such that $\langle\langle c_i^0, T\rangle, \langle e, T'\rangle\rangle \in E$. We define $\delta_\mathcal{D}(\langle c_i^0, T\rangle, e) = f_\mathcal{D}(\langle e, T'\rangle)$.

We claim that $\mathcal{D}$ is a correct design. Assume towards contradiction that there is a word $w \in L(\mathcal{T}_\mathcal{D}) \setminus L(\mathcal{S})$. Consider the run r of $\mathcal{T}_\mathcal{D}$ on w. Let $\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_m} \in \mathcal{L}^*$ be sequence of components that r traverses. Let $w_1, \ldots, w_m$ be the partition of w according to $\mathcal{D}$. That is, for $1 \leq j \leq m$, the subword w_j is induced while r is in component $\mathcal{B}_{i_j}$ and $\delta_{i_j}^*(w_j) \in E_{i_j}$. Let $\pi_w = e_{i_1}, \ldots, e_{i_{m-1}} \in \mathcal{E}^*$ be the exit states that r visits. Note that since B_{i_m} gains control last the word $w_1 \cdot \ldots \cdot w_{m-1}$ is suffix-less, thus $\delta_\mathcal{D}^*(w_1 \cdot \ldots \cdot w_{m-1}) = \langle c_{i_m}^0, d\rangle$ for some $d \in D$ having $\nu(d) = i_m$. Moreover, $w_m \in L(\mathcal{B}_{i_m})$. Since we defined $\mathcal{D}$ to agree with $f_\mathcal{D}$ on the components that gain control, the finite play π that is the outcome of the game when Player 1 plays $f_\mathcal{D}$ and Player 2 chooses the exit states $e_{i_1}, \ldots, e_{i_{m-1}}$ reaches the Player 2 vertex $v = \langle c_{i_m}^0, T\rangle \in V_2$, for some $T \in 2^S$. We claim that $v \notin \alpha$. Indeed, the definition of E implies that there is a vertex $s \in T$ such that $\delta_\mathcal{S}^*(w_1 \cdot \ldots \cdot w_{m-1}) = s$. Since $w \notin L(\mathcal{S})$, we have $w_m \notin L(\mathcal{S}^s)$ and $w_m \in L(\mathcal{B}_{i_m})$. Thus, π is a winning play for Player 2, contradicting our assumption that $f_\mathcal{D}$ is a winning strategy, and we are done. □

We continue to study the lower bound.

Theorem 3.4. *The open-design problem is EXPTIME-hard.*

Proof. We describe a reduction from the problem of deciding whether Player 1 has a winning strategy in a *partial-information safety game*, known to be EXPTIME-complete [10].

Partial-information games (PI games, for short) are a variant of the *full-information* games (FI games, for short) defined above in which Player 1 has imperfect information [33]. The vertices, which we refer to as *locations*, denoted L, are partitioned into *observations*, denoted $\mathcal{O}$. Player 1 is unaware of the location on which the token is placed and is only aware of the observation it is in. Accordingly, in his turn, Player 1 cannot select the next location to move the token to. Instead, the edges in the game are labeled with actions, denoted Γ. In each round, Player 1 selects an action and Player 2 resolves nondeterminism and chooses the next location the token moves to. Initially, the token is placed on $l^0 \in L$. The set of labeled edges in the game is $\Upsilon \subseteq L \times \Gamma \times L$, and the safety objective α is given with respect to the observations, thus $\alpha \subseteq \mathcal{O}$.

Formally, a PI game is played on an arena $\langle \Gamma, L, \mathcal{O}, l_0, \Upsilon, \alpha\rangle$, where Γ is a set of actions, L is a set of locations, $\mathcal{O} \subseteq 2^L$ is a set of observations that form a partition of L (that is, $\mathcal{O} = \{L_1, \ldots, L_k\}$ where for all $i \neq j \in [k]$, we have that $L_i \cap L_j = \emptyset$, and $\bigcup_{i\in[k]} L_i = L$), $l_0 \in L$ is an initial location, $\Upsilon \subseteq L \times \Gamma \times L$ are labeled edges, and α is a winning condition defined with respect to $\mathcal{O}$. In particular, in safety games $\alpha \subseteq \mathcal{O}$ are safe observations for Player 1. We require that for every location $l \in L$ and action $a \in \Gamma$, there is a location $l' \in L$ such that $\langle l, a, l'\rangle \in \Upsilon$.

The game proceeds similarly to FI games. At the beginning of the game, a token is placed on the initial location l_0. Assume the token is on a location $l \in L$. Player 1 moves first and selects an action $a \in \Gamma$. Player 2 resolves non-determinism and selects a location $l' \in L$ such that $\langle l, a, l'\rangle \in \Upsilon$. Since Player 1 only observes the member in $\mathcal{O}$ in which l is, a strategy for Player 1 is a function $f_1 : (\mathcal{O} \cdot \Gamma)^* \cdot \mathcal{O} \to \Gamma$. Since Player 2 has complete information, a strategy for him is a function $f_2 : L^+ \cdot \Gamma \to L$ such that for a play $\pi = l_0, \ldots, l_m$ and an action $a \in \Gamma$, we have $\langle l_m, a, f_2(\pi, a)\rangle \in \Upsilon$. The definition of winning strategies are as in the full-information setting.

Consider a PI safety game $\mathcal{G} = \langle \Gamma, L, \mathcal{O}, l^0, \Upsilon, \alpha \rangle$. We construct a library $\mathcal{L}$ and a DFA $\mathcal{S}$ such that there is a correct design for $\mathcal{S}$ using the components of $\mathcal{L}$ iff Player 1 wins $\mathcal{G}$. Recall that a design reads an exit state and outputs the component that gets control next. Also, a Player 1 strategy in $\mathcal{G}$ reads observations and outputs actions. Accordingly, the library $\mathcal{L}$ consists of box-transducers $\mathcal{B}_a$, one for every action $a \in \Gamma$. The exit states of the components correspond to the observations in $\mathcal{O}$. That is, when a component exits through an observation $L_i \in \mathcal{O}$, the design decides which component $\mathcal{B}_a \in \mathcal{L}$ gains control, which corresponds to a Player 1 strategy that chooses the action $a \in \Gamma$ from the observation L_i.

Next, we define words to correspond to Player 2 strategies. Recall that Player 2 resolves nondeterminism in $\mathcal{G}$. That is, when the token is in location $l \in L$ and Player 1 selects an action $a \in \Gamma$, Player 2 selects an edge $\langle l, a, l' \rangle \in \Upsilon$ and the token moves to the location l'. Accordingly, $\Sigma_I = \Upsilon$, thus a word in Σ_I^* is a sequence of edges. We define the specification $\mathcal{S}$ so that a word witnesses the incorrectness of the design only if it corresponds to a *losing path* in $\mathcal{G}$, where $\langle l_0, a_1, l'_1 \rangle, \langle l_1, a_2, l'_2 \rangle, \langle l_2, a_2, l'_3 \rangle, \ldots, \langle l_{m-1}, a_{m-1}, l'_m \rangle \in \Upsilon^*$ is a losing path if for all $1 \leq i \leq m-1$ we have that $l'_i = l_i$ and $l'_m \notin \alpha$.

Finally, we define the components so that a correct design corresponds to a winning Player 1 strategy. For $a \in \Gamma$, the component $\mathcal{B}_a \in \mathcal{L}$ can *process* every edge that is labeled with a. When reading such an edge it relinquishes control, and when reading an edge that is labeled with $b \neq a$, the component enters a sink which is intuitively a rejecting sink. Thus, in order to avoid processing a word $w = t_1 \ldots t_m$ that corresponds to a losing path, a design must assign control to some component $\mathcal{B}_a$ with $a \neq a_i$, after reading the input prefix $t_1 \ldots t_{i-1}$, for $1 \leq i \leq m$.

Formally, for $a \in \Gamma$, we define the box-transducer $\mathcal{B}_a = \langle \Upsilon, \{\top, \bot\}, \{c_a^0, c_a^{rej}\} \cup \mathcal{O}, c_a^0, \delta_a, \nu_a, \mathcal{O} \rangle$, where $\nu_a(c_a^0) = \top$, $\nu_a(c_a^{rej}) = \bot$, and δ_a is defined as follows. Consider an edge $\langle l, a, l' \rangle \in \Upsilon$. We define $\delta_a(c_a^0, \langle l, a, l' \rangle) = P$, for the observation P with $l' \in P$. For an edge $\langle l, b, l' \rangle \in \Upsilon$, with $b \neq a$, we define $\delta_a(c_a^0, \langle l, b, l' \rangle) = c_a^{rej}$. The state c_a^{rej} is a sink, thus $\delta_a(c_a^{rej}, \sigma) = c_a^{rej}$ for all $\sigma \in \Sigma_I$. Note that the component $\mathcal{B}_a$ relinquishes control and outputs $\top$ when it reads a transition labeled a. Otherwise, it gets stuck in the rejecting sink.

The specification is given by the DFA $\mathcal{S} = \langle \Sigma_I \times \Sigma_O, S, \delta_{\mathcal{S}}, l^0, F_{\mathcal{S}} \rangle$, where $S = L \cup \{s_{acc}\}$, the accepting states $F_{\mathcal{S}}$ are the states that do not belong to observations in α, thus $F_{\mathcal{S}} = S \setminus \bigcup_{P \in \alpha} P$, and we describe $\delta_{\mathcal{S}}$ in the following. Consider a location $l \in L$. For every edge $t = \langle l, a, l' \rangle \in \Upsilon$ we define $\delta_{\mathcal{S}}(l, \langle t, \top \rangle) = l'$. For every other letter $\sigma \in \Sigma_I \times \Sigma_O$, we define $\delta_{\mathcal{S}}(l, \sigma) = s_{acc}$.

Note that a word $w \in (\Sigma_I \times \Sigma_O)^*$ is not in $L(\mathcal{S})$ if it is of the form $w = w_1 \oplus w_2 \in (\Sigma_I \times \Sigma_O)^*$ for $w_2 \in \top^*$ and $w_1 \in \Sigma_I^*$ corresponds to a *losing path* in $\mathcal{G}$. That is, $w = t_1 \ldots t_m$, where $t_1, \ldots, t_m \in \Upsilon$, $t_1 = \langle l^0, a_1, l_1 \rangle$, for $1 \leq i < m$, the target location of the transition t_i is the source location of t_{i+1}, and assuming $t_m = \langle t_{m-1}, a_m, t_m \rangle$, we have $t_m \notin \alpha$. Recall that a component $\mathcal{B}_a \in \mathcal{L}$ relinquishes control after reading an edge that the action $a \in \Gamma$ participates in. When reading an edge that a does not participate in, $\mathcal{B}_a$ enters a sink and outputs $\bot$. Thus, in order to avoid processing the word w as in the above, a design must assign control to some component $\mathcal{B}_a$ with $a \neq a_i$, after reading the input prefix $t_1 \ldots t_{i-1}$, for $1 \leq i \leq m$.

We claim that Player 1 wins $\mathcal{G}$ iff there is a correct design $\mathcal{D}$ for $\mathcal{S}$ using the components in $\mathcal{L}$. For the first direction, consider a Player 1 winning strategy f_1. We construct a design $\mathcal{D}$ inductively. Let $a_1 = f_1(P_0)$, where $P_0 \in \mathcal{O}$ is such that $l^0 \in P_0$. The first component to gain control is $\mathcal{B}_{a_1}$. Consider a run on some word, and let $\mathcal{B}_{a_1}, P_1, \mathcal{B}_{a_2}, P_2, \ldots \mathcal{B}_{a_m} P_m$ be the sequence of components and corresponding exit states that the run visits. Recall that $P_1, \ldots, P_m$ are observations in $\mathcal{O}$. Let $\pi = P_0, a_1, P_1, \ldots, P_m$, where $l^0 \in P_0$. Note that π might not correspond to a path in $\mathcal{G}$, in which case $\mathcal{D}$ assigns control to an arbitrary component. Otherwise, assuming $f_1(\pi) = a_{m+1}$, $\mathcal{D}$ assigns control to $\mathcal{B}_{a_{m+1}}$.

We claim that $\mathcal{D}$ is a correct design. Assume towards contradiction that there is a word $w \in L(\mathcal{T}_{\mathcal{D}}) \setminus L(\mathcal{S})$. Since $w \notin L(\mathcal{S})$, its projection on Σ_O is in $\top^*$ and the projection of w on Σ_I is a losing path $t_1, \ldots, t_m$. For $1 \leq i \leq m$, let $t_i = \langle l_{i-1}, a_i, l_i \rangle$, where $l_0 = l^0$. Let r be the run of $\mathcal{T}_{\mathcal{D}}$ on w. Since the projection of w on Σ_O is a sequence of $\top$'s, the sequence of components and exit states that r passes is $\mathcal{B}_{a_1}, P_1, \ldots, \mathcal{B}_{a_m}, P_m$. Indeed, if for $1 \leq i \leq m$ and $b \neq a_i$, the component $\mathcal{B}_b \in \mathcal{L}$ is in control when a_i is read, the component outputs $\bot$. Let π be the path that corresponds to $t_1, \ldots, t_m$, thus $\pi = l_0, \ldots, l_m$, where $l_0 = l^0$. Consider the Player 2 strategy f_2 that selects the edges $t_1, \ldots, t_m$. Since by our definition of the components in $\mathcal{L}$, for $1 \leq i \leq m$, we have $l_i \in P_i$, we can prove by induction that π is a prefix of $out(f_1, f_2)$. Since $l_m \notin \alpha$, the strategy f_2 is winning against f_1, which is a contradiction to the fact that it is a winning strategy, and we are done.

For the second direction, consider a correct design $\mathcal{D}$. We define a Player 1 strategy $f_{\mathcal{D}}$ inductively as follows. We abuse notation and refer to the labeling function ν of $\mathcal{D}$ as a function from its states D to $\mathcal{L}$. Assume the first component that $\mathcal{D}$ assigns control to is $\mathcal{B}_{a_1} \in \mathcal{L}$. For the observation P_0 such that $l^0 \in P_0$, we define $f_1(P_0) = a_1$. Consider a play $\pi = l_0, l_1, \ldots, l_m$, where $l_0 = l^0$. Let $\tau = P_0, a_1, P_1, \ldots, a_m, P_m$ such that for $1 \leq i \leq m$ we have $l_i \in P_i$. Let $a_1, \ldots, a_m \in \Gamma$ such that, for $1 \leq i \leq m$ we have $\langle l_{i-1}, a_i, l_i \rangle \in \Upsilon$. Let $\delta_{\mathcal{D}}^*(P_1 \ldots P_m) = d$ and let $\nu(d) = \mathcal{B}_{a_m} \in \mathcal{L}$. We define $f_{\mathcal{D}}(\tau) = a_m$.

We claim that $f_{\mathcal{D}}$ is a winning strategy. Assume towards contradiction that there is a Player 2 strategy f_2 and a finite prefix π of $out(f_{\mathcal{D}}, f_2)$ that is winning for Player 2. Let $\tau = t_1, \ldots, t_m$ be the edges that π traverses. Consider the run of $\mathcal{T}_{\mathcal{D}}$ on τ. We can prove by induction that by our definition of $f_{\mathcal{D}}$, for $1 \leq i \leq m$, when the letter $t_i = \langle l_{i-1}, a_i, l_i \rangle$ is read, the component $\mathcal{B}_{a_i}$ is in control. Thus, the output of $\mathcal{T}_{\mathcal{D}}$ when reading the word τ is a sequence w of $|\tau|$ $\top$'s. Since τ corresponds to a losing path, we have $\tau \oplus w \in L(\mathcal{T}_{\mathcal{D}}) \setminus L(\mathcal{S})$, which is a contradiction to the correctness of $\mathcal{D}$, and we are done. □

4. Libraries with costs

Given a library and a specification, there are possibly many, in fact infinitely many, designs that are solutions to the design problem. As a trivial example, assume that $L(\mathcal{S}) = a^*$ and that the library contains a component $\mathcal{B}$ that traverses the letter a (that is, $\mathcal{B}$ consists of an accepting initial state that has an a-transition to an exit state). An optimal design for $\mathcal{S}$ uses $\mathcal{B}$ once: it has a single state with a self loop in which $\mathcal{B}$ is called. Other designs can use $\mathcal{B}$ arbitrarily many times. When we wrote "optimal" above, we assumed that the smaller the design is, the better it is. In this section we would like to formalize the notion of optimality and add to the composition picture different costs that components in the libraries may have.

In order to capture a wide set of scenarios in practice, we associate with each component in $\mathcal{L}$ two costs: a *construction cost* and a *quality cost*. The costs are given by the functions $c\text{-}cost, q\text{-}cost : \mathcal{L} \to \mathbb{R}^+ \cup \{0\}$, respectively. The construction cost of a component is the cost of adding it to the library. Thus, a design that uses a component pays its construction cost once, and (as would be the case in Section 6), when several designs use a component, they share its construction cost. The quality cost measures the performance of the component, and involves, for example, its number of states. Accordingly, a design pays the quality cost of a component every time it uses it, and the fact the component is used by other designs is not important.

Formally, consider a library $\mathcal{L} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ and a design $\mathcal{D} = \langle E, [n], D, d^0, \delta, \nu\rangle$. The number of times $\mathcal{D}$ uses a component B_i is $nused(\mathcal{D}, \mathcal{B}_i) = |\{d \in D : \nu(d) = i\}|$. The set of components that are used in $\mathcal{D}$, is $used(\mathcal{D}) = \{\mathcal{B}_i : nused(\mathcal{D}, \mathcal{B}_i) \geq 1\}$. The cost of a design is then $cost(\mathcal{D}) = \sum_{\mathcal{B} \in used(\mathcal{D})} c\text{-}cost(\mathcal{B}) + nused(\mathcal{D}, \mathcal{B}) \cdot q\text{-}cost(\mathcal{B})$.

We state the problem of finding the cheapest design as a decision problem. For a specification DFA $\mathcal{S}$, a library $\mathcal{L}$, and a threshold μ, we say that an input $\langle \mathcal{S}, \mathcal{L}, \mu\rangle$ is in BCD (standing for "bounded cost design") iff there exists a correct design $\mathcal{D}$ such that $cost(\mathcal{D}) \leq \mu$. In this section we study the BCD problem in a setting with a single user. Thus, decisions are independent of other users of the library, which, recall, may influence the construction cost.

In section 3, we reduced the design problem to the problem of the solution of a safety game. In particular, we showed how a winning strategy in the game induces a correct design. Note that while we know that safety games admit memoryless strategies, memoryless strategies are not guaranteed to lead to optimal designs. We first study this point and show that, surprisingly, while memoryless strategies are sufficient for obtaining an optimal design in the closed setting, this is not the case in the open setting. The source of the difference is the fact that the language of a design in the open setting may be strictly contained in the language of the specification. The approximation may enable the user to generate a design that is more complex and is still cheaper in terms of cost. This is related to the fact that over approximating the language of a DFA may result in exponentially bigger DFAs [6]. We are still able to bound the size of the cheapest design by the size of the game.

4.1. On the optimality and non-optimality of memoryless strategies

Consider a closed library $\mathcal{L}$ and a DFA $\mathcal{S}$. Recall that a correct design for $\mathcal{S}$ from components in $\mathcal{L}$ is induced by a winning strategy of Player 1 in the game $\mathcal{G}_{\mathcal{S}}$ (see Theorem 3.1). If the winning strategy is not memoryless, we can trim it to a memoryless one and obtain a design whose state space is a subset of the design induced by the original strategy. Since the design has no flexibility with respect to the language of $\mathcal{S}$, we cannot do better. Hence the following lemma.

Lemma 4.1. *Consider a closed library $\mathcal{L}$ and a DFA $\mathcal{S}$. For every $\mu \geq 0$, if there is a correct design $\mathcal{D}$ with $cost(\mathcal{D}) \leq \mu$, then there is a correct design $\mathcal{D}'$ induced by a memoryless strategy for Player 1 in $\mathcal{G}_{\mathcal{S}}$ such that $cost(\mathcal{D}') \leq \mu$.*

Proof. Consider a correct design $\mathcal{D}$ with $cost(\mathcal{D}) \leq \mu$. Let $f_{\mathcal{D}}$ be a winning Player 1 strategy in the safety game $\mathcal{G}_{\mathcal{S}}$ as constructed in the proof of Theorem 3.1. Note that while we know that safety games admit memoryless strategies, memoryless strategies are not guaranteed to lead to optimal designs. Thus, $f_{\mathcal{D}}$ need not be a memoryless strategy. We construct a Player 1 memoryless strategy $f'_{\mathcal{D}}$ by *trimming* $f_{\mathcal{D}}$ in every *memoryfull* vertex. Then, we construct a design $\mathcal{D}'$ from $f'_{\mathcal{D}}$ and show that it costs no more than $\mathcal{D}$.

In order to prove the claim formally, we need a few definitions. Consider a Player 1 vertex $v = \langle e, s\rangle \in V_1$. We define $adj(f_{\mathcal{D}}, v) \in V$ to be the vertices $f_{\mathcal{D}}$ continues to from v. That is, $u \in adj(f_{\mathcal{D}}, v)$ if there is a play π_u that ends in v and is an outcome of the game when Player 1 plays $f_{\mathcal{D}}$ and Player 2 plays some strategy, and $f_{\mathcal{D}}(\pi_u) = u$. We refer to π_u as a *witness* play for $u \in adj(f_{\mathcal{D}}, v)$. Note that there can be many witness plays for a vertex in $adj(f_{\mathcal{D}}, v)$. Further note that if $adj(f_{\mathcal{D}}, v)$ is a singleton, then v is a *memoryless* vertex with respect to $f_{\mathcal{D}}$. Consider a vertex $u = \langle c_i^0, s\rangle \in adj(f_{\mathcal{D}}, v)$. Intuitively, since we define $f_{\mathcal{D}}$ to assign control as $\mathcal{D}$, and since $f_{\mathcal{D}}$ continues to u after π_u, there must be a state $d_u \in D$ with $\nu(d_u) = i$. Thus, d_u is a *witness* state for the fact that $f_{\mathcal{D}}(\pi_u) = u$. Again, there can be several witness states for a vertex in $adj(f_{\mathcal{D}}, v)$. Formally, let π_u be a witness play for u and let $w_u \in \Sigma^*$ be the word that is induced by the Player 2 choices in π_u. That is, $\delta^*_{\mathcal{S}}(w_u) = s$, where recall that $v = \langle e, s\rangle$. Moreover, let $e_1, \ldots, e_m$ be the exit states that are traversed in π_u, and let $d_u = \delta^*_{\mathcal{D}}(e_1 \ldots e_m)$. Since $f_{\mathcal{D}}$ assigns control to the same components as $\mathcal{D}$, the run of $\mathcal{A}_{\mathcal{D}}$ on w_u ends in the state $\langle c_i^0, d_u\rangle$. We refer to w_u and d_u as a *witness* word and state for u, respectively.

We define $f'_{\mathcal{D}}$ as follows. For every $v \in V_1$, we choose a vertex $u \in adj(f_{\mathcal{D}}, v)$ arbitrarily and define $f'_{\mathcal{D}}(v) = u$. It is not hard to prove that $f'_{\mathcal{D}}$ is a winning strategy. Let $\mathcal{D}'$ be the design that corresponds to $f'_{\mathcal{D}}$ and is constructed as in the proof

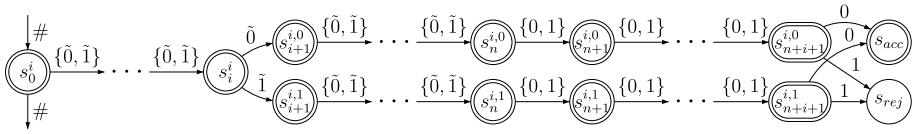


Fig. 1. A description of the i-th chain of the specification $\mathcal{S}_n$.

of Theorem 3.1. Since $f'_{\mathcal{D}}$ is winning, $\mathcal{D}'$ is a correct design. We claim that $cost(\mathcal{D}') \leq cost(\mathcal{D})$. Recall that the states of $\mathcal{D}'$ are the Player 2 vertices in $\mathcal{G}_{\mathcal{S}}$. Further recall that in the construction of $\mathcal{D}'$ we skip Player 1 vertices and proceed according to $f'_{\mathcal{D}}$. Consider a reachable Player 2 vertex $\langle c_i^0, s\rangle$. We denote by ν' the labeling function of $\mathcal{D}'$. Recall that c_i^0 is the initial state of the component $\mathcal{B}_i \in \mathcal{L}$ and $\nu'(\langle c_i^0, s\rangle) = i$. First, we claim that if a component is used in $\mathcal{D}'$ then it is used in $\mathcal{D}$, thus $used(\mathcal{D}') \subseteq used(\mathcal{D})$. Indeed, if $\mathcal{B}_i \in used(\mathcal{D}')$, there is a state in $\mathcal{D}'$ that corresponds to a reachable Player 2 vertex $v = \langle c_i^0, s\rangle$, for some $s \in S$. Let s_v be a witness state of v. Since the labeling of s_v is $\nu(s_v) = i$, we have $\mathcal{B}_i \in used(\mathcal{D})$. We conclude that the sum of construction costs that is incurred by $\mathcal{D}'$ is at most that of $\mathcal{D}$.

We prove that the sum of quality costs incurred by $\mathcal{D}'$ is at most that of $\mathcal{D}$. We prove that for every $\mathcal{B}_i \in used(\mathcal{D}')$ we have $nused(\mathcal{D}', \mathcal{B}_i) \leq nused(\mathcal{D}, \mathcal{B}_i)$. For a reachable vertex $v \in V_2$ let $d_v \in D$ be an arbitrary choice of witness state of v. Let w_v be the corresponding witness word. We show that the mapping from a reachable vertex $v \in V_2$ to $d_v \in D$ is a one-to-one mapping. Consider $v, u \in V_2$ with $d_v = d_u$. Let $d = d_v = d_u$. Let $i = \nu(d)$. Since the component-state of v and u is the initial state of $\mathcal{B}_i$, we have $v = \langle c_i^0, s\rangle$ and $u = \langle c_i^0, s'\rangle$. To conclude the proof, we show that $s = s'$, thus $v = u$. We claim that $L(\mathcal{S}^s) = L(\mathcal{S}^{s'})$ and since $\mathcal{S}$ is a minimal DFA, it would follow that $s = s'$. Recall that w_v and w_u are the witness words for v and u, respectively, thus $\delta^*_{\mathcal{S}}(w_v) = s$ and $\delta^*_{\mathcal{S}}(w_{v'}) = s'$. Moreover, since w_v and w_u are the witness words that correspond to the witness states $d_v = d_u = d$, we have $\delta^*_{\mathcal{D}}(w_v) = \delta^*_{\mathcal{D}}(w_u) = \langle c_i^0, d\rangle$. Consider a word $x \in \Sigma^*$. If $x \in L(\mathcal{S}^s)$, then since $\delta^*_{\mathcal{S}}(w_v) = s$, we have $w_v \cdot x \in L(\mathcal{S})$. Since $\mathcal{D}$ is a correct design, $w_v \cdot x \in L(\mathcal{A}_{\mathcal{D}})$. Thus, $\delta^*_{\mathcal{D}}(w_v \cdot x)$ is an accepting state. Since $\delta^*_{\mathcal{D}}(w_v) = \delta^*_{\mathcal{D}}(w_u)$, we have $w_u \cdot x \in L(\mathcal{A}_{\mathcal{D}})$ and in turn $w_u \cdot x \in L(\mathcal{S})$. Since $\delta^*_{\mathcal{S}}(w_u) = s'$, we have $u \in L(\mathcal{S}^{s'})$. The other direction is symmetric, and we are done. □

While Lemma 4.1 seems intuitive, it does not hold in the setting of open systems. There, a design has the freedom to generate a language that is a subset of $L(\mathcal{S})$, as long as it stays receptive. This flexibility allows the design to generate a language that need not be related to the structure of the game $\mathcal{G}_{\mathcal{S}}$, which may significantly reduce its cost. Formally, we have the following.

Lemma 4.2. *There is an open library $\mathcal{L}$ and a family of DFAs $\mathcal{S}_n$ such that $\mathcal{S}_n$ has a correct design $\mathcal{D}_n$ with cost 1 but every correct design for $\mathcal{S}_n$ that is induced by a memoryless strategy for Player 1 in $\mathcal{G}_{\mathcal{S}_n}$ has cost n.*

Proof. We define $\mathcal{S}_n = \langle \Sigma_I \times \Sigma_O, S_n, \delta_{\mathcal{S}_n}, s_0^0, F_{\mathcal{S}_n}\rangle$, where $\Sigma_I = \{\tilde{0}, \tilde{1}, \#\}$, $\Sigma_O = \{0, 1, \_\}$, and S_n, $\delta_{\mathcal{S}_n}$ and $F_{\mathcal{S}_n}$ are as follows. When describing $\delta_{\mathcal{S}_n}$, for ease of presentation, we sometimes omit the letter in Σ_I or Σ_O and we mean that every letter in the respective alphabet is allowed. The words we focus on consist of an input sequence in $\#^*$ followed by an input sequence in $\{\tilde{0}, \tilde{1}\}^n$, and end in an output sequence in $\{0, 1\}^n$. Consider an input word $\#^i u$, for $1 \leq i \leq n$ and $u_1, \ldots, u_n \in (\tilde{0} + \tilde{1})^n$. The prefix of #'s constraints the i-th output letter following this input sequence. If $u_i = \tilde{0}$, then the i-th output letter should be 0, and if $u_i = \tilde{1}$, then it should be 1. The automaton $\mathcal{S}_n$ consists of n chains, sharing an accepting sink s_{acc} and a rejecting sink s_{rej}. For $0 \leq i \leq n-1$, we depict the i-th chain in Fig. 1.

Intuitively, we construct $\mathcal{S}_n$ to count the number of #'s and then check the corresponding indices. Another method is to disregard the count of #'s. Then, since we do not know which input letter needs to match the corresponding output letter, we match all of them. That is, we remember all the input letters in $(\tilde{0} + \tilde{1})^n$ and output a matching word in $(0 + 1)^n$. In our component library, counting #'s has a cost whereas remembering input letters in $\{\tilde{0}, \tilde{1}\}$ is free, thus the size of the optimal design is going to be exponential in the input. A design that corresponds to a memoryless strategy counts #'s and is thus suboptimal.

We define $\mathcal{S}_n$ and the library $\mathcal{L}$ so that the structure of $\mathcal{G}_{\mathcal{S}_n}$ supports the first method, but counting each # has a cost of 1. Consequently, a memoryless strategy of Player 1 in $\mathcal{G}_{\mathcal{S}_n}$ induces a design that counts, and is therefore of cost n, whereas an optimal design follows the second method, and since it does not count the number of #'s, its cost is only 1.

We can now describe the DFA $\mathcal{S}_n$ in more detail. For $0 \leq i < n-1$, we define $\delta_{\mathcal{S}_n}(s_0^i, \#) = s_0^{i+1}$ and $\delta_{\mathcal{S}_n}(s_0^n, \#) = s_0^n$. Note that words of the form $\#^i a_1 \tilde{1} a_2 b 0$ or $\#^i a_1 \tilde{0} a_2 b 1$ are not in $L(\mathcal{S}_n)$, where if $0 \leq i \leq n-1$, then $a_1 \in (\tilde{0} + \tilde{1})^i$, $a_2 \in (\tilde{0} + \tilde{1})^{n-i-1}$, and $b \in (0 + 1)^i$, and if $i > n-1$ then the lengths of a_1, a_2, and b are $n-1$, 0, and $n-1$, respectively. We require that after reading a word in $\#^*(\tilde{0} + \tilde{1})^n$ there is an output of n letters in $\{0, 1\}$. Thus, for $n \leq j \leq n + i + 1$, we define $\delta_{\mathcal{S}_n}(s_j^{i,0}, \_) = \delta_{\mathcal{S}_n}(s_j^{i,1}, \_) = s_{rej}$. Also, $\mathcal{S}_n$ accepts every word that has a # after the initial prefix of # letters. Thus, for $1 \leq j \leq i$, we define $\delta_{\mathcal{S}_n}(s_j^i, \#) = s_{acc}$, and for $i + 1 \leq j \leq n + i + 1$ and $t \in \{0, 1\}$ we define $\delta_{\mathcal{S}_n}(s_j^{i,t}, \#) = s_{acc}$.

The library $\mathcal{L}$ is depicted in Fig. 2. The quality and construction costs of all the components is 0, except for $\mathcal{B}_1$ which has $q\text{-}cost(\mathcal{B}_1) = 1$ and $c\text{-}cost(\mathcal{B}_1) = 0$.

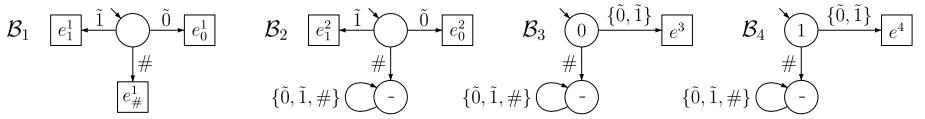


Fig. 2. The library $\mathcal{L}$. Exit states are square nodes and the output of a state is written in the node.

We first claim that every correct design must cost at least 1. Indeed, such a design must use $\mathcal{B}_1$ at least once. Otherwise, the output sequence for the input word $\#\tilde{0}^{n+1}$ is a sequence in $\_^*$. Recall that we require that after reading a word in $\#^*(\tilde{0}+\tilde{1})^n$ there is an output of n letters in $\{0,1\}$, thus such a design is incorrect.

We describe a correct design $\mathcal{D}_n$ with $cost(\mathcal{D}_n) = 1$. Intuitively, as explained above, $\mathcal{D}_n$ does not track the number of #'s that are read and can thus use $\mathcal{B}_1$ only once. Instead, $\mathcal{D}_n$ keeps track of all of the n input letters in $\{\tilde{0}, \tilde{1}\}$ that follow the sequence in $\#^*$. Thus, after $\tilde{0}$ or $\tilde{1}$ is read, $\mathcal{B}_2$ gains control for n consecutive times. Then, assuming the word $u \in \{\tilde{0}, \tilde{1}\}^n$ is read in the first phase, control is alternated between $\mathcal{B}_3$ and $\mathcal{B}_4$ so that the word $v \in \{0,1\}^n$ is output, where u and v represent the same vector. Note that $\mathcal{D}_n$ uses the components $\mathcal{B}_2$, $\mathcal{B}_3$, and $\mathcal{B}_4$ exponentially many times in n.

Formally, we define $\mathcal{D}_n = \langle \Sigma_I, \Sigma_O, D_n, \delta_{\mathcal{D}_n}, d^0, \nu \rangle$. We define $\nu(d^0) = 1$ and $\delta_{\mathcal{D}_n}(d^0, e_\#^1) = d^0$. When $\mathcal{B}_1$ relinquishes control through e_1^1 or e_0^1 control is passed to $\mathcal{B}_2$ for n consecutive times. During this phase, $\mathcal{D}_n$ *remembers* the vector in $\{\tilde{0}, \tilde{1}\}^n$ that is read, thus $\mathcal{D}_n$ has $2^{\Omega(n)}$ states. Next, the components $\mathcal{B}_3$ and $\mathcal{B}_4$ alternate control for n times as we describe in the following. Consider an input word $w = \#^i \cdot a_1 \dots a_n b$, where $a_1, \dots, a_n \in \{\tilde{0}, \tilde{1}\}^n$ and $b \in \{\tilde{0}, \tilde{1}\}^m$ for $0 \leq m < n-1$. After reading w either $\mathcal{B}_3$ or $\mathcal{B}_4$ relinquish control. If $a_{m+1} = 0$, then the next component to gain control is $\mathcal{B}_3$ and otherwise it is $\mathcal{B}_4$. After n alternations of control $\mathcal{B}_4$ is assigned control indefinitely.

Since the initial state of $\mathcal{D}_n$ is the only state that is labeled with $\mathcal{B}_1$, we have $used(\mathcal{B}_1) = 1$, thus $cost(\mathcal{D}_n) = 1$. We claim that $\mathcal{D}_n$ is a correct design. Indeed, note that $L(\mathcal{T}_{\mathcal{D}_n})$ consists of two types of words. The first type are prefixes of words in $\#^*ab0^*$, for $a \in \{\tilde{0}, \tilde{1}\}^n$ and $b \in \{0,1\}^n$ that represent the same vector, thus for $1 \leq i \leq n$, if $a_i = \tilde{0}$, then $b_i = 0$ and if $a_i = \tilde{1}$, then $b_i = 1$. The second type are words that have a prefix in $\#^*(\tilde{0}+\tilde{1})^+\#$. Clearly, $L(\mathcal{T}_{\mathcal{D}_n}) \subseteq L(\mathcal{S}_n)$, and we are done. □

4.2. Solving the BCD problem

Theorem 4.3. *The BCD problem is NP-complete for closed designs.*

Proof. Consider an input $\langle \mathcal{S}, \mathcal{L}, \mu \rangle$ to the BCD problem. By Lemma 4.1, we can restrict the search for a correct design $\mathcal{D}$ with $cost(\mathcal{D}) \leq \mu$ to these induced by a memoryless strategy for Player 1 in $\mathcal{G}_{\mathcal{S}}$. By the definition of the game $\mathcal{G}_{\mathcal{S}}$, such a design has at most $|\mathcal{C}_0 \times S|$ states. Since checking if a design is correct and calculating its cost can be done in polynomial time, membership in NP follows.

For the lower bound, we describe a reduction from SET-COVER. Consider an input $\langle U, T, \mu \rangle$ to SET-COVER, where $U = \{1, \dots, l\}$ is a universe, $T = \{T_1, \dots, T_m\}$ is a set of subsets over U, i.e., $T_i \subseteq U$, for $1 \leq i \leq m$, and $\mu \in \mathbb{N}$. We construct a closed-library $\mathcal{L}$ and a DFA $\mathcal{S}$ such that there is a design $\mathcal{D}$ that costs at most μ iff there is a set-cover of size at most μ.

We construct the DFA $\mathcal{S}$ over the alphabet U as a chain of $l+1$ states that accepts only the word $12 \dots l$. There are $m+2$ components in $\mathcal{L}$. For $1 \leq i \leq m$, the component $\mathcal{B}_i$ corresponds to the set T_i. Its initial state is c_i^0 and it has an exit state e_j^i for every element $j \in T_i$. For $j \in T_i$ there is a transition $\delta_i(c_i^0, j) = e_j^i$. The last two components are $\mathcal{B}_{acc}$ and $\mathcal{B}_{rej}$. For consistency we also refer to these components as $\mathcal{B}_{m+1}$ and $\mathcal{B}_{m+2}$, respectively. They consist of a single initial non-exit state that is accepting in $\mathcal{B}_{acc}$ and rejecting in $\mathcal{B}_{rej}$. For $\mathcal{B}_i \in \mathcal{L}$, we define $c\text{-}cost(\mathcal{B}_i) = 1$ and we define $c\text{-}cost(\mathcal{B}_{acc}) = c\text{-}cost(\mathcal{B}_{rej}) = 0$. Finally, the quality cost of the components is 0, thus we define $q\text{-}cost \equiv 0$.

We claim that there is a set cover of size at most μ iff there is a design of cost at most μ. For the first direction, consider a set cover $T' \subseteq T$ with $|T'| \leq \mu$. For $j \in U$, let $cover(j)$ be the index such that $j \in T_{cover(j)}$ and $T_{cover(j)} \in T'$. We construct a design $\mathcal{D}$ with $cost(\mathcal{D}) \leq \mu$. We define $\mathcal{D}$ so that when $\mathcal{A}_{\mathcal{D}}$ reads the word $1 \dots l$, the component that gets j-th control, for $1 \leq j \leq l$, is $\mathcal{B}_{cover(j)}$. If the j-th input letter is $j' \neq j$, then the next component to gain control is $\mathcal{B}_{rej}$. Finally, the component that gains $l+1$-th control is $\mathcal{B}_{acc}$.

Formally, $\mathcal{D} = \langle \mathcal{E}, [m+2], D, \delta_{\mathcal{D}}, d_1, \nu \rangle$, where $\{d_1, \dots, d_l, d_{acc}, d_{rej}\}$ and we define ν and $\delta_{\mathcal{D}}$ as follows. For $1 \leq j \leq l$, we define $\nu(d_j) = cover(j)$. For $1 \leq i \leq m$, recall that the component $\mathcal{B}_i$ has an exit state e_j^i for every $j \in T_i$. For $j < l$ and $1 \leq i \leq m$, we define $\delta_{\mathcal{D}}(d_j, e_j^i) = d_{j+1}$ and we define $\delta_{\mathcal{D}}(d_l, e_l^i) = d_{acc}$. For $j \neq i$, we define $\delta_{\mathcal{D}}(d_j, e_j^i) = d_{rej}$. It is not hard to see that $L(\mathcal{A}_I) = \{12 \dots l\}$, and thus $\mathcal{D}$ is a correct design. The components that $\mathcal{D}$ uses are the ones that correspond to the sets in T', thus $\mathcal{D}$ uses at most μ components with construction cost 1, and its cost is at most μ.

For the other direction, assume there is a correct design $\mathcal{D}$ with $cost(\mathcal{D}) \leq \mu$. Consider the collection $T' \subseteq T$ of sets that correspond to the components $\{\nu(d) : d \in D \text{ and } \nu(d) \notin \{\mathcal{B}_{acc}, \mathcal{B}_{rej}\}\}$. We claim that T' is a set cover with at most μ sets. Since $cost(\mathcal{D}) \leq \mu$, it uses at most μ components that correspond to sets in T, thus $|T'| \leq \mu$. We are left to show that T' is a set cover. Consider the run r of $\mathcal{A}_{\mathcal{D}}$ on the word $1 \dots l$. Since $\mathcal{D}$ is correct, r is accepting. Specifically it does not get stuck. Since the components we described above relinquish control after reading a single letter, the control is passed $l+1$

times during r. Let $\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_l}, \mathcal{B}_{i_{l+1}}$ be the sequence of components that gain control. Consider $j \in U$. We claim that j is covered in T'. Since r does not get stuck, there is a transition labeled j in the component $\mathcal{B}_{i_j}$, the j-th component to gain control in r. Thus, $j \in T_{i_j}$, the set in T that corresponds to $\mathcal{B}_{i_j}$. Since $T_{i_j} \in T'$, j is covered, and we are done. □

Remark 4.4. Note that a different attempt to reduce SET-COVER to the BCD problem would be to define the components as in the proof of Theorem 4.3 and define $\mathcal{S}$ so that it accepts one-letter words for each element in U. However, this attempt fails since, intuitively, a design does not know which letter is going to be read. Even if there is a set cover $T' \subseteq T$ and control is assigned to the component $\mathcal{B}_i$ where $T_i \in T'$, a letter $j \in U \setminus T_i$ can be read. Thus, the fact that we use a one-word specification allows a design to expect the next letter that should be read.

We turn to study the open setting, which is significantly harder than the closed one. For the upper bound, we first show that while we cannot restrict attention to designs induced by memoryless strategies, we can still bound the size of optimal designs:

Theorem 4.5. *For an open library $\mathcal{L}$ with ℓ components and a specification $\mathcal{S}$ with n states, a cheapest correct design $\mathcal{D}$ has at most $\binom{n}{\lceil n/2 \rceil} \cdot \ell$ states.*

Proof. Given $\mathcal{S}$ and $\mathcal{L}$, assume towards contradiction that any cheapest smallest design $\mathcal{D}$ for $\mathcal{S}$ using the components in $\mathcal{L}$ has more than $\binom{n}{\lceil n/2 \rceil} \cdot \ell$ states.

Consider a word $w \in L(\mathcal{T}_\mathcal{D})$. Let $\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_m} \in \mathcal{L}$ be the components that are traversed in the run r of $\mathcal{T}_\mathcal{D}$ that induces w. Let $w = w_1 \cdot \ldots \cdot w_m$, where, for $1 \leq j \leq m$, the word w_j is induced in the component $\mathcal{B}_{i_j}$. We say that w is suffix-less if $w_m = \epsilon$, thus r ends in the initial state of the last component to gain control. We denote by $\pi_w(\mathcal{D}) = e_{i_1}, \ldots, e_{i_{m-1}} \in \mathcal{E}^*$ the sequence of exit states that r visits.

For a state $d \in D$, we define the set $S_d \subseteq S$ so that $s \in S_d$ iff there exists a suffix-less word $w \in (\Sigma_I \times \Sigma_O)^*$ such that $\delta^*_\mathcal{S}(w) = s$ and $\delta^*_\mathcal{D}(\pi_w(\mathcal{D})) = d$. Since $\mathcal{D}$ has more than $\binom{n}{\lceil n/2 \rceil} \cdot \ell$ states, there is a component $\mathcal{B}_i \in \mathcal{L}$ such that the set $D' \subseteq D$ of states that are labeled with $\mathcal{B}_i$ is larger than $\binom{n}{\lceil n/2 \rceil}$. The set D' can be seen as a set of subsets of states of $\mathcal{S}$. A pigeonhole-principle argument[5] shows that there must be two states $d, d' \in D'$ that have $S_{d'} \subseteq S_d$. Note that $\nu(d) = \nu(d') = i$.

We construct a new design $\mathcal{D}'$ by merging d' into d. Formally, we define $D' = D \setminus \{d'\}$. If d' is the initial state of $\mathcal{D}$, then we define $d'^0 = d$ and otherwise we define $d'^0 = d^0$. For $t \in D'$ and $e \in \mathcal{E}$, if $\delta_\mathcal{D}(t, e) = d'$, then we define $\delta_{\mathcal{D}'}(t, e) = d$, and otherwise we define $\delta_{\mathcal{D}'}(t, e) = \delta_\mathcal{D}(t, e)$. Finally, for every $t \in D'$ we define $\nu'(t) = \nu(t)$.

Clearly, for every component $\mathcal{B} \in \mathcal{L}$ we have $\mathit{nused}(\mathcal{D}, \mathcal{B}) \geq \mathit{nused}(\mathcal{D}', \mathcal{B})$, thus $\mathit{cost}(\mathcal{D}) \geq \mathit{cost}(\mathcal{D}')$. Moreover, $\mathcal{D}'$ has less states than $\mathcal{D}$. Thus, in order complete the contradiction and conclude the proof of the theorem, we prove the following.

Claim 4.6. *The design $\mathcal{D}'$ is correct. That is, $L(\mathcal{T}_{\mathcal{D}'}) \subseteq L(\mathcal{S})$.*

In order to prove the claim we prove the following.

Claim 4.7. *For every suffix-less (w.r.t $\mathcal{D}'$) word $x \in L(\mathcal{T}_{\mathcal{D}'})$ there is a suffix-less (w.r.t $\mathcal{D}$) word $y \in L(\mathcal{T}_\mathcal{D})$ such that $\delta^*_\mathcal{S}(x) = \delta^*_\mathcal{S}(y)$ and $\delta^*_\mathcal{D}(\pi_y(\mathcal{D})) = \delta^*_{\mathcal{D}'}(\pi_x(\mathcal{D}'))$.*

Before proving the correctness of Claim 4.7 we show that it implies the correctness of Claim 4.6. Assume towards contradiction that there is a word $w \in L(\mathcal{T}_{\mathcal{D}'}) \setminus L(\mathcal{S})$. Let r be the run of $\mathcal{T}_{\mathcal{D}'}$ that induces w and let $\mathcal{B}_i \in \mathcal{L}$ be the last component to gain control in r. Let $w_0, \ldots, w_m$ be the partition of w with respect to $\mathcal{D}'$ and let $x = w_0 \cdot \ldots \cdot w_{m-1}$. That is, x is the longest suffix-less prefix of w (possibly $x = \epsilon$). Note that $\pi_w(\mathcal{D}') = \pi_x(\mathcal{D}')$. By Claim 4.7 there is a suffix-less (w.r.t $\mathcal{D}$) word $y \in L(\mathcal{T}_\mathcal{D})$ such that $\delta^*_\mathcal{S}(x) = \delta^*_\mathcal{S}(y)$ and $\delta^*_\mathcal{D}(\pi_y(\mathcal{D})) = \delta^*_{\mathcal{D}'}(\pi_x(\mathcal{D}'))$. Since $\delta^*_\mathcal{D}(\pi_y(\mathcal{D})) = \delta^*_{\mathcal{D}'}(\pi_x(\mathcal{D}'))$, the last component to gain control in the two runs is $\mathcal{B}_i$. Since both x and y are suffix-less, the runs that induce them end in the initial state of $\mathcal{B}_i$, thus $\delta^*_\mathcal{D}(y) = \delta^*_{\mathcal{D}'}(x)$. Recall that $w = x \cdot w_n \in L(\mathcal{T}_{\mathcal{D}'})$. Thus, $y \cdot w_n \in L(\mathcal{T}_\mathcal{D})$. Since $\delta^*_\mathcal{S}(y) = \delta^*_\mathcal{S}(x)$ and $w \notin L(\mathcal{S})$, we have $y \cdot w_n \notin L(\mathcal{S})$. Thus, $y \cdot w_n \in L(\mathcal{T}_\mathcal{D}) \setminus L(\mathcal{S})$, and we reach a contradiction to the correctness of $\mathcal{D}$, and we are done.

We continue to prove Claim 4.7. Consider a suffix-less (w.r.t $\mathcal{D}'$) word $x \in L(\mathcal{T}_{\mathcal{D}'})$ and let r be the run of $\mathcal{D}'$ on $\pi_x(\mathcal{D}')$, which are the exit states that $\mathcal{T}_{\mathcal{D}'}$ visits when inducing x. The proof is by induction on the number of visits of r to $d \in D'$. For the base case, r does not visit d. Thus, r is a run of $\mathcal{D}$ on $\pi_x(\mathcal{D}')$, and we define $y = x$.

Assume the claim is correct for runs with m visits to d and we prove for runs with $m + 1$ visits. Let $\pi_x(\mathcal{D}') = \pi_{x_1}(\mathcal{D}') \cdot e \cdot \pi_{x_3}(\mathcal{D}')$ such that $\delta^*_{\mathcal{D}'}(\pi_{x_1}(\mathcal{D}') \cdot e) = d$ is the last visit of r to d. Let $x = x_1 \cdot x_2 \cdot x_3$, where x_1 and $x_1 \cdot x_2$ are suffix-less.

[5] Consider a set A having m elements and the power set lattice $\langle 2^A, \subseteq \rangle$. Two elements $P, Q \subseteq A$ are comparable if $P \subseteq Q$ or $Q \subseteq P$. The argument follows from the fact that the maximal size of an incomparable set in such a lattice is $\binom{m}{\lceil m/2 \rceil}$.

That is, assuming $t = \delta^*_{\mathcal{D}'}(\pi_{x_1}(\mathcal{D}'))$ and $\nu'(t) = i$, then x_2 is the sub word of x that is induced by the component $\mathcal{B}_i$, thus $\delta^*_i(x_2) = e \in \mathcal{E}$. We distinguish between two cases. In the first case $x_2 = \epsilon$, thus $x_1 = \epsilon$ and $d'^0 = d$, and we define $y = x$ as in the base case. In the second case, $x_2 \neq \epsilon$. By the induction hypothesis, there is a suffix-less (w.r.t $\mathcal{D}$) word $y_1 \in (\Sigma_I \times \Sigma_O)^*$ such that $\delta^*_{\mathcal{D}}(\pi_{y_1}(\mathcal{D})) = \delta^*_{\mathcal{D}'}(\pi_{x_1}(\mathcal{D}')) = t \in D'$ and $\delta^*_{\mathcal{S}}(y_1) = \delta^*_{\mathcal{S}}(x_1)$. If $\delta_{\mathcal{D}}(t, e) = d$, then $\delta_{\mathcal{D}'}(t, e) = d$ and $y = y_1 \cdot x_2 \cdot x_3$ clearly meets the requirements of the claim. Assume $\delta_{\mathcal{D}}(t, e) = d'$. Note that $y_1 \cdot x_2$ is a suffix-less (w.r.t $\mathcal{D}$) word with $\delta^*_{\mathcal{D}}(\pi_{y_1 \cdot x_2}(\mathcal{D})) = d'$. Let $\delta^*_{\mathcal{S}}(y_1 \cdot x_2) = s$. Since $S_{d'} \subseteq S_d$, there is a suffix-less (w.r.t $\mathcal{D}$) word $z \in (\Sigma_I \times \Sigma_O)^*$ such that $\delta_{\mathcal{D}}(\pi_z(\mathcal{D})) = d$ and $\delta^*_{\mathcal{S}}(z) = s$. We define $y = z \cdot x_3$, which clearly meets the requirements of the claim, and we are done. □

Before we turn to the lower bound, we argue that the exponential blow-up proven in Theorem 4.5 cannot be avoided:

Theorem 4.8. *For every $n \geq 1$, there is an open library $\mathcal{L}$ and specification $\mathcal{S}_n$ such that the size of $\mathcal{L}$ is constant, the size of $\mathcal{S}_n$ is $O(n^2)$, and every cheapest correct design for $\mathcal{S}_n$ that uses components from $\mathcal{L}$ has at least 2^n states.*

Proof. Consider the specification $\mathcal{S}_n$ and library $\mathcal{L}$ that are described in Lemma 4.2. We claim that every correct design that costs 1 cannot count #'s and should thus remember vectors in $\{\tilde{0}, \tilde{1}\}^n$. Assume towards contradiction that there is a correct design $\mathcal{D}$ with $cost(\mathcal{D}) = 1$ and $\mathcal{D}$ has less than 2^n states. Thus, $used(\mathcal{D}, \mathcal{B}_1) = 1$. Since $\mathcal{D}$ must assign initial control to $\mathcal{B}_1$, its initial state d^0 is labeled with 1. We claim that if $\mathcal{B}_1$ relinquishes control after reading #, it is assigned control again. Indeed, if one of the other components gains control, for the input word $\#\#\tilde{0}^{n+1}$ the output sequence is a sequence in $\_^*$. Recall that we require that after reading a word in $\#^*(\tilde{0} + \tilde{1})^n$ there is an output of n letters in $\{0, 1\}$, thus we reach a contradiction to the correctness of $\mathcal{D}$. We conclude that the initial state has a $e^1_\#$-labeled self loop, thus $\delta_{\mathcal{D}}(d^0, e^1_\#) = d^0$.

Since $\mathcal{D}$ has less that 2^n states and the components $\mathcal{B}_2$, $\mathcal{B}_3$, and $\mathcal{B}_4$ relinquish control after reading a letter in $\{\tilde{0}, \tilde{1}\}$, there are two words $a, b \in \{\tilde{0}, \tilde{1}\}^n$ such that $a \neq b$ and $\delta^*_{\mathcal{D}}(a) = \delta^*_{\mathcal{D}}(b)$. Let $0 \leq i \leq n-1$ such that, w.l.o.g., $1 = a_i \neq b_i = 0$. By the above, for the initial state $q^0_{\mathcal{D}}$ of $\mathcal{T}_{\mathcal{D}}$ we have $\delta_{\mathcal{D}}(q^0_{\mathcal{D}}, \#) = q^0_{\mathcal{D}}$. Thus, $\delta^*_{\mathcal{D}}(\#^i a \tilde{0}^i) = \delta^*_{\mathcal{D}}(\#^i b \tilde{0}^i)$. Recall that $\mathcal{S}_n$ requires that either 0 or 1 is output after these words are read, thus either $\mathcal{B}_3$ or $\mathcal{B}_4$ gain control. In the first case, the output letter is 0 and the input word $\#^i a \tilde{0}^{i+1}$ produces an output that violates the specification, and in the second case 1 is output and the input word $\#^i b \tilde{0}^{i+1}$ produces an output that violates the specification, and we are done. □

Theorem 4.9. *The BCD problem for open libraries is NEXPTIME-complete.*

Proof. Membership in NEXPTIME follows from Theorem 4.5 and the fact we can check the correctness of a design and calculate its cost in polynomial time.

For the lower bound, we describe a reduction from *EXP-TILING*. Consider an input to EXP-TILING $\langle T, V, H, n\rangle$, where $T = \{t_1, \ldots, t_m\}$ is a set of tiles, $V, H \subseteq T \times T$ are *vertical* and *horizontal* relations, respectively, and $n \in \mathbb{N}$ is an index. We say that $\langle T, V, H, n\rangle \in$ EXP-TILING if it is possible to fill a $2^n \times 2^n$ square with the tiles in T that respects the two relations. Formally, $\langle T, V, H, n\rangle \in$ EXP-TILING if there is a function $f : 2^n \times 2^n \to T$ such that for $a, b \in 2^n$, if $a < 2^n$ then $V(f(a, b), f(a+1, b))$, and if $b < 2^n$, then $H(f(a, b), f(a, b+1))$.

Given an input $\langle T, V, H, n\rangle$, we construct an input $\langle \mathcal{S}, k\rangle$ to the open-BCD problem such that there is an exponential tiling iff there is a correct design $\mathcal{D}$ with $cost(\mathcal{D}) \leq 2^{2n+1} + 1$. The idea behind the reduction is similar to that of Lemma 4.2. We define $\Sigma_I = \{\tilde{0}, \tilde{1}, \#, c, v, h, -\}$ and $\Sigma_O = \{0, 1, \_\} \cup T$. For $x \in \{0, 1\}^n$, we use $\tilde{x}$ to refer to the $\{\tilde{0}, \tilde{1}\}$ copy of x. The library $\mathcal{L}$ has the same components as in Lemma 4.2 with an additional *tile component* $\mathcal{B}_t$ for every $t \in T$. The component $\mathcal{B}_t$ outputs t in its initial state, and when reading c, v, or h, it relinquishes control. When reading every other letter, it enters an accepting sink. The construction costs of the components in $\mathcal{L}$ is 0. We define $q\text{-}cost(\mathcal{B}_1) = 2^{2n} + 1$, and $q\text{-}cost(\mathcal{B}_t) = 1$ for all $t \in T$. The other components' quality cost is 0. Note that the costs are given in binary.

Consider a correct design $\mathcal{D}$ with $cost(\mathcal{D}) \leq 2^{2n+1} + 1$. We define $\mathcal{S}$ so that a correct design must use $\mathcal{B}_1$ at least once, thus $\mathcal{D}$ uses it exactly once. Intuitively, $a \cdot b$, for $a, b \in \{0, 1\}^n$, can be thought of as two coordinates in a $2^n \times 2^n$ square. We define $\mathcal{S}$ so that after reading the word $\tilde{a} \cdot \tilde{b} \in \{\tilde{0}, \tilde{1}\}^{2n}$, a component is output, which can be thought of as the tile in the (a, b) coordinate in the square. The next letter that can be read is either c, v, or h. Then, $\mathcal{S}$ enforces that the output is $a \cdot b$, $(a+1) \cdot b$, and $a \cdot (b+1)$, respectively. Thus, we show that $\mathcal{D}$ uses exactly 2^{2n} tile components and the tiling that it induces is legal.

Recall that $\mathcal{D}$ uses $\mathcal{B}_1$ exactly once and uses at most 2^{2n} tile components. We describe the specification $\mathcal{S}$ as an intersection of the languages of three DFAs. The first DFA $\mathcal{S}_c$ is similar to the specification in Theorem 4.8. The differences are that there are $2n$ chains and after the sequence of $2n$ input letters, a letter $\langle t, c\rangle$, for $t \in T$, must be read. Thus, it guarantees that when $\mathcal{D}$ reads $\tilde{a}\tilde{b}c\_^{2n}$, for $\tilde{a}, \tilde{b} \in \{\tilde{0}, \tilde{1}\}^n$, it outputs a word $\_^{2n}tab$, where $t \in T$ and recall that $a, b \in \{0, 1\}^n$ represent the same vectors as $\tilde{a}$ and $\tilde{b}$, respectively.

Consider $a, b \in \{0, 1\}^n$, and let $d_{a,b}$ be the state of $\mathcal{D}$ that is reached after reading the word $\tilde{a}\tilde{b}$. Then, control must be assigned to a tile component, thus $t = \nu(\delta^*_{\mathcal{D}}(\tilde{a}\tilde{b}))$ is a tile component. Consider the input word $w = c\_^{2n}$. If w is read when $\mathcal{D}$ is at state $d_{a,b}$, the word $a \cdot b$ must be output. Recall that $\mathcal{D}$ uses at most 2^{2n} tile components. A key observation is the following. Consider a state d of $\mathcal{D}$ that is labeled by a tile component. If $\mathcal{D}$ is in state d and ab is output when reading w, then $d = d_{a,b}$.

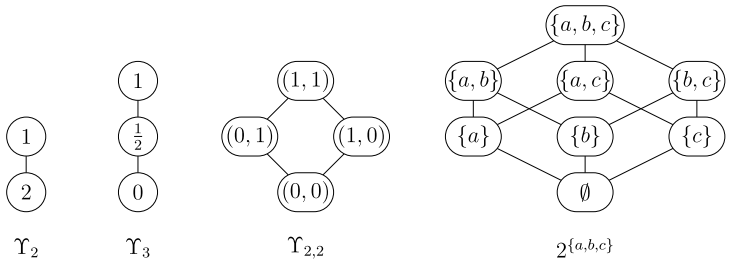


Fig. 3. Some lattices.

The next DFA from which $\mathcal{S}$ is devised is $\mathcal{S}_v$. It guarantees that when $\mathcal{D}$ reads the word $\tilde{a}\tilde{b}vc^{-2n}$ it outputs $\_^{2n}t_it_j(a+1)b$ for $t_i, t_j \in T$ with $V(t_i, t_j)$. Finally, in order to deal with the tiles in the top row, S_v accepts every word that starts with a prefix in $\#^*\tilde{1}^n(\tilde{0}+\tilde{1})^n v$. The DFA $\mathcal{S}_h$ is defined similarly to $\mathcal{S}_v$.

We define a tiling f by $f(a,b) = \nu(d_{a,b})$ for $a, b \in \{0,1\}^n$. We show that the above observation implies that f respects V, and the proof for H similar. Consider $a, b \in \{0,1\}^n$. If $a = 1^n$, then $f(a,b)$ is in the top row, and there is nothing to prove. Otherwise, we view a and b as numbers and claim that $V(f(a,b), f(a+1,b))$. Recall that after reading ab, $\mathcal{D}$ reaches the state $d_{a,b}$ and the tile $f(a,b)$ is output. By the above, $\mathcal{S}_v$ guarantees that if v is read, the next tile that is output respects V. We claim that $f(a+1,b)$ is output. Indeed, when reading c^{-2n}, $\mathcal{S}_v$ guarantees that $(a+1)b$ must be output. By the observation above, $d_{(a+1),b}$ is the only state of $\mathcal{D}$ from which an input of c^{-2n} produces an output of $(a+1)b$.

The other direction, namely, if there is a tiling of $2^n \times 2^n$, then there is a design that costs $2^{2n}+1$, is dual to the above. □

5. Computation-based cost

As detailed in Section 1, while the cost model we use in Section 4 is suited for measuring structural properties of the components, like their sizes, one often wants to associate the computations of the components with a value. For example, in a system that issues grants upon requests, a goal of the designer can be to design a system that minimizes the waiting time for a grant once a request is received. A standard model for reasoning about such costs of computations is *lattice automata* [26], which assign to each word a value that is an element of a finite lattice.

We study the synthesis problem from component libraries where the specification is given by a deterministic lattice automaton (LDFA, for short) and the components are box LDFAs. Thus, our goal is to compose the components in the library to construct an LDFA that is equivalent to the specification LDFA, where equivalence means agreement on the values assigned to all words.

The complexity of problems on lattice automata typically coincide with the complexity of the corresponding problem in the Boolean setting, possibly with a blow-up that depends on the size of the lattice (more precisely, on the number of its join-irreducible elements, as would be defined below). An exception is the problem of minimization of LDFAs, which is NP-complete [22]. It is shown in [22] that there is no canonical minimal LDFA for a latticed language. Recall that minimal DFAs play a key role in our upper bound for the design problem in the closed setting (Theorem 3.1) as we assumed the specification is given as such a DFA.

Even with no canonical minimal LDFA for the language of the specification, we show that the design problem can be solved in polynomial time. We assume the specification is given as a *separable* LDFA, which is a type of LDFAs we introduce here. In such an LDFA, every two states have a separating word (similar to minimal DFAs). That is, if there are words w_1 and w_2 whose runs reach two different states, then there is a word w such that $w_1 \cdot w$ is assigned a value that is different from the one the automaton assigns to $w_2 \cdot w$. We show that every latticed language has a separable LDFA of polynomial size. This result might be of independent interest.

5.1. Lattice automata

Let $\langle A, \leq \rangle$ be a partially ordered set, and let P be a subset of A. An element $a \in A$ is an *upper bound* on P if $a \geq b$ for all $b \in P$. Dually, a is a *lower bound* on P if $a \leq b$ for all $b \in P$. An element $a \in A$ is the *least element of* P if $a \in P$ and a is a lower bound on P. Dually, $a \in A$ is the *greatest element of* P if $a \in P$ and a is an upper bound on P. A partially ordered set $\langle A, \leq \rangle$ is a *lattice* if for every two elements $a, b \in A$ both the least upper bound and the greatest lower bound of $\{a, b\}$ exist, in which case they are denoted $a \vee b$ (*a join b*) and $a \wedge b$ (*a meet b*), respectively. A lattice is *complete* if for every subset $P \subseteq A$ both the least upper bound and the greatest lower bound of P exist, in which case they are denoted $\bigvee P$ and $\bigwedge P$, respectively. In particular, $\bigvee A$ and $\bigwedge A$ are denoted $\top$ (*top*) and $\bot$ (*bottom*), respectively. A lattice $\langle A, \leq \rangle$ is finite if A is finite. Note that every finite lattice is complete. A lattice $\langle A, \leq \rangle$ is *distributive* if for every $a, b, c \in A$, we have $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ and $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.

In Fig. 3 we describe some finite lattices. The elements of the lattice Υ_2 are the usual truth values 1 (*true*) and 0 (*false*) with the order $0 \leq 1$. The lattice Υ_n contains the values $0, 1, \ldots, n-1$, with the order $0 \leq 1 \leq \ldots \leq n-1$. The lattice $\Upsilon_{2,2}$ is

the Cartesian product of two Υ_2 lattices, thus $(a,b) \leq (a',b')$ if both $a \leq a'$ and $b \leq b'$. Finally, the lattice $2^{\{a,b,c\}}$ is the power set of $\{a,b,c\}$ with the set-inclusion order. In this lattice, *join* and *meet* coincide with union and intersection, respectively, and we have, for example, $\{a\} \vee \{b\} = \{a,b\}$, $\{a\} \wedge \{b\} = \bot$, $\{a,c\} \vee \{b\} = \top$, and $\{a,c\} \wedge \{b\} = \bot$.

Consider a lattice Υ. We abuse notation and refer to Υ also as the set of elements (rather than a pair of a set with an order on it). A *join-irreducible* element $d \in \Upsilon$ is a value, other than $\bot$, for which if $d_1 \vee d_2 \geq d$, then either $d_1 \geq d$ or $d_2 \geq d$. For example, in the power set lattice, the join-irreducible elements are the singletons. Indeed, if $d_1 \cup d_2 \geq \{a\}$, then either $a \in d_1$ or $a \in d_2$, thus $d_1 \geq \{a\}$ in the first case and $d_2 \geq \{a\}$ in the second. On the other hand, $\{a,b\}$ is not join-irreducible since $\{a\} \cup \{b\} \geq \{a,b\}$ and $\{a\} \not\geq \{a,b\}$ and $\{b\} \not\geq \{a,b\}$. By Birkhoff's representation theorem for finite distributive lattices in order to prove that $d_1 = d_2$, it is sufficient to prove that for every join-irreducible element d it holds that $d_1 \geq d$ iff $d_2 \geq d$. We denote the set of join-irreducible elements of Υ by $JI(\Upsilon)$.

For a set X of elements, an Υ-*set over* X is a function $S : X \to \Upsilon$ assigning to each element of X a value in Υ. Thus, $S \in \Upsilon^X$. It is convenient to think about $S(x)$ as the truth value of the statement "x is in S". We say that an Υ-set S is *Boolean* if $S(x) \in \{\top, \bot\}$ for all $x \in X$. Consider an alphabet Σ. An Υ-*language* is an Υ-set over Σ^*. Thus, an Υ-language $L : \Sigma^* \to \Upsilon$ assigns a value in Υ to each word over Σ.

A *deterministic lattice automaton* on finite words (LDFA, for short) is a tuple $\mathcal{A} = \langle \Upsilon, \Sigma, Q, Q_0, \delta, F \rangle$, where Υ is a finite lattice, Σ is a finite alphabet, Q is a finite set of states, $Q_0 \in \Upsilon^Q$ is an Υ-set of initial states, $\delta \in \Upsilon^{Q \times \Sigma \times Q}$ is an Υ-transition-relation, and $F \in \Upsilon^Q$ is an Υ-set of accepting states.

The fact that $\mathcal{A}$ is deterministic is reflected in two conditions on Q_0 and δ. First, there is at most one state $q \in Q$, called the *initial state* of $\mathcal{A}$, such that $Q_0(q) \neq \bot$. In addition, for every state $q \in Q$ and letter $\sigma \in \Sigma$, there is at most one state $q' \in Q$, called the σ-*destination* of q, such that $\delta(q, \sigma, q') \neq \bot$. The *run* of an LDFA on a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_n$ is a sequence $r = q_0, \ldots, q_n$ of $n+1$ states, where q_0 is the initial state of $\mathcal{A}$, and for all $1 \leq i \leq n$, it holds that q_i is the σ_i-*destination* of q_{i-1}. We extend the notion of destination to words. For a word $w \in \Sigma^*$ and a state $q \in Q$, we use $\delta^*(q, w)$ to refer to the w-destination of q. When $q \in Q_0$, we omit it and use $\delta^*(w)$. The *value* of w is $val(w) = Q_0(q_0) \wedge \bigwedge_{i=1}^{n} \delta(q_{i-1}, \sigma_i, q_i) \wedge F(q_n)$. Intuitively, $Q_0(q_0)$ is the value of q_0 being initial, $\delta(q_{i-1}, \sigma_i, q_i)$ is the value of q_i being a successor of q_{i-1} when σ_i is the input letter, $F(q_n)$ is the value of q_n being accepting, and the value of w is the meet of all these values. The *traversal value* of w is $trav\text{-}val(w) = Q_0(q_0) \wedge \bigwedge_{i=1}^{n} \delta(q_{i-1}, \sigma_i, q_i)$, and its *acceptance value* is $F(q_n)$. The Υ-language of $\mathcal{A}$, denoted $L(\mathcal{A})$, maps each word w to the value of its run in $\mathcal{A}$. In case such a run does not exist, the value of the word is $\bot$. An example of an LDFA can be found in Fig. 4. We say that two LDFA $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* iff they assign the same values to all words, thus for $w \in \Sigma^*$, we have $val(\mathcal{A}, w) = val(\mathcal{B}, w)$.

Note that traditional deterministic automata over finite words (DFA, for short) correspond to LDFA over the lattice Υ_2. Indeed, over Υ_2, a word is mapped to the value $\top$ iff the run on it uses only transitions with value $\top$ and its final state has value $\top$.

An LDFA is *simple* if Q_0 and δ are Boolean. Note that the traversal value of a run r of a simple LDFA is either $\bot$ or $\top$, thus the value of r is induced by F. For simplicity, in such LDFAs we assume $\delta : Q \times \Sigma \to Q$.

The following lemma was proven in [26] and we give it here for completeness.

Lemma 5.1. *[26] Every LDFA over a lattice Υ with n states has an equivalent simple LDFA with $n \cdot |\Upsilon|$ states.*

Proof. Consider an LDFA $\mathcal{D} = \langle \Upsilon, \Sigma, D, d_0, \delta, F \rangle$. Intuitively, in the simple LDFA $\mathcal{D}'$ we "remember" the lattice value of a run of $\mathcal{D}$. Formally, let $\mathcal{D}' = \langle \Upsilon, \Sigma, D \times \Upsilon, \langle d_0, \top \rangle, \delta', F' \rangle$, where $F'(\langle d, \ell \rangle) = \ell \wedge F(d)$ and we define δ' below. Recall that for every $d \in D$ and $\sigma \in \Sigma$, there is at most one state $d' \in D$ such that $\delta(d, \sigma, d') \neq \bot$. We define $\delta'(\langle d, \ell \rangle, \sigma) = \langle d', \ell \wedge \delta(d, \sigma, d') \rangle$). Clearly, the size of $\mathcal{D}'$ is $|D| \cdot |\Upsilon|$, and it is simple. Moreover, it is not hard to see that $\mathcal{D}$ and $\mathcal{D}'$ are equivalent. □

5.2. Separable LDFAs

Recall that in the Boolean setting, our solution to the closed design problem relied on the fact that the specification was given as a *minimal* DFA. Specifically, we used the fact that states in such a DFA have *separating words*. It is known that there is no canonical minimal LDFA for a lattice language [22]. However, we show below that it is possible to assume that the specification is given by means of an LDFA whose states have separating words.

Formally, consider an LDFA $\mathcal{D} = \langle \Upsilon, \Sigma, D, d_0, \delta, F \rangle$. We say that two states $d_1, d_2 \in D$ have a separating word, if there is a word $w \in \Sigma^*$ such that for every two words $w_1, w_2 \in \Sigma^*$ with $\delta^*_{\mathcal{D}}(w_1) = d_1$ and $\delta^*_{\mathcal{D}}(w_2) = d_2$, we have $val(\mathcal{D}, w_1 \cdot w) \neq val(\mathcal{D}, w_2 \cdot w)$. We say that $\mathcal{D}$ is *separable* if every two states have a separating word.

For example, consider the LDFA $\mathcal{D}$ that is depicted in Fig. 4. The alphabet of $\mathcal{D}$ is $\Sigma = \{1,2,3,4\}$ and it is defined over the lattice $\langle 2^{a,b,c}, \subseteq \rangle$. The states' names appear above them and their acceptance value inside. We show that $\mathcal{D}$ is not separable. For this, we show that the states d_1 and d_2 are not separable, as for the words $w_1 = 1$ and $w_2 = 2$, we have $\delta^*_{\mathcal{D}}(w_1) = d_1$ and $\delta^*_{\mathcal{D}}(w_2) = d_2$, but there is no word w such that $val(\mathcal{D}, w_1 \cdot w) \neq val(\mathcal{D}, w_2 \cdot w)$. Indeed, words with a prefix w_1 or w_2 have either a value of $\{a\}$ or a value of $\bot$. If 1^* is read after reading w_1 or w_2, then the value is $\{a\}$, and otherwise it is $\bot$. Note that it is not possible to simply merge d_1 and d_2 as that would result in a change of the value of either the word 32 or the word 42.

We show that every simple LDFA $\mathcal{D}$ has an equivalent separable LDFA $\mathcal{D}'$. By Lemma 5.1, this would imply that every LDFA has an equivalent separable LDFA. Let $\mathcal{D} = \langle \Upsilon, \Sigma, D, d_0, \delta, F \rangle$. We define an equivalence relation on D and use it to

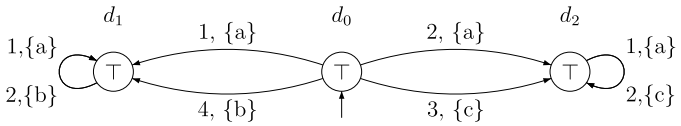


Fig. 4. An example of an LDFA in which the states d_1 and d_2 are not separable.

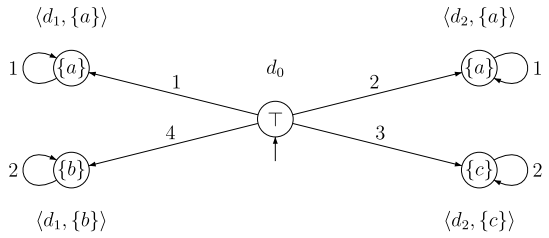


Fig. 5. A simple LDFA that is equivalent to the one in Fig. 4.

construct $\mathcal{D}'$. Consider a join-irreducible element $a \in JI(\Upsilon)$. We construct a DFA $\mathcal{D}_a$ with $L(\mathcal{D}_a) = \{w \in \Sigma^* : val(\mathcal{D}, w) \geq a\}$ as follows. We define $\mathcal{D}_a = \langle \Sigma, D, d_0, \delta_a, F_a\rangle$, where for $q \in Q$ and $\sigma \in \Sigma$, we have $\delta_a(q, \sigma) = q'$ iff $\delta(q, \sigma, q') = \top$, and $d \in F_a$ iff $a \leq F(d)$. Since $\mathcal{D}_a$ is a DFA, its language has a canonical minimal DFA $\mathcal{D}'_a$. Moreover, the states of $\mathcal{D}_a$ refine these of $\mathcal{D}'_a$ such that there is a mapping f_a from the states of $\mathcal{D}_a$ to these of $\mathcal{D}'_a$ such that for two states $d_1, d_2 \in D$, we have $f_a(d_1) = f_a(d_2)$ iff for every word $w \in \Sigma^*$, we have $w \in L(\mathcal{D}_a^{d_1})$ iff $w \in L(\mathcal{D}_a^{d_2})$.

We are ready to define the equivalence relation on D. For two states d_1 and d_2 in D, we say that d_1 and d_2 are equivalent, denoted $d_1 \sim d_2$, iff $F(d_1) = F(d_2)$ and for every $a \in JI(\Upsilon)$, we have $f_a(d_1) = f_a(d_2)$. Let $D_\sim$ be the partition of D according to $\sim$.

Consider two states $d_1, d_2 \in D$ such that $d_1 \sim d_2$ and a letter $\sigma \in \Sigma$. We claim that $d'_1 = \delta(d_1, \sigma) \sim \delta(d_2, \sigma) = d'_2$. Assume otherwise, thus either $F(d'_1) \neq F(d'_2)$ or there is $a \in JI(\Upsilon)$ such that $f_a(d'_1) \neq f_a(d'_2)$. Assume that the second case holds. Then, there is a word $w \in \Sigma^*$, such that $a \leq val(\mathcal{D}^{d'_1}, w)$ and $a \not\leq val(\mathcal{D}^{d'_2}, w)$, or the other way around. But then, the word $\sigma \cdot w$ is separating for d_1 and d_2, contradicting the fact that $f_a(d_1) = f_a(d_2)$. The proof for the first case is similar.

Let $\mathcal{D}' = \langle \Upsilon, \Sigma, D_\sim, d'_0, \delta', F'\rangle$, where $d'_0 \in D_\sim$ is such that $d_0 \in d'_0$, and δ' and F' are defined as follows. For $A \in D_\sim$, $d \in A$, and $\sigma \in \Sigma$, we define $\delta'(A, \sigma) = A'$ iff $\delta(d, \sigma) \in A'$. Then, $F'(A) = F(d)$. By the above, δ' is well defined. Note that $\mathcal{D}'$ is a simple LDFA.

For example, consider the simple LDFA $\mathcal{D}$ that is equivalent to the one in Fig. 5. We do not state the values of the transitions as they are all $\top$. Note that the states $\langle d_1, \{a\}\rangle$ and $\langle d_2, \{a\}\rangle$ are not separable and in $\mathcal{D}'$ they are merged.

We claim that $\mathcal{D}'$ is equivalent to $\mathcal{D}$. Consider a word $w = w_1, \ldots, w_n \in \Sigma^*$ and let $r = r_0, r_1, \ldots, r_n$ and $r' = r'_0, r'_1, \ldots, r'_n$ be the runs of $\mathcal{D}$ and $\mathcal{D}'$ on w, respectively. It is not hard to prove by induction on i that for every $0 \leq i \leq n$, we have $r_i \in r'_i$. Since $\mathcal{D}$ and $\mathcal{D}'$ are simple, we have $val(\mathcal{D}, w) = F(r_n)$ and $val(\mathcal{D}', w) = F'(r'_n)$. By the claim, $r_n \in r'_n$, and by the definition of F', we have $F'(r'_n) = F(r_n)$, and we are done.

Finally, we claim that $\mathcal{D}'$ is separable. Consider two words $w_1, w_2 \in \Sigma^*$ such that $A_1 = \delta^*_{\mathcal{D}'}(w_1) \neq \delta^*_{\mathcal{D}'}(w_2) = A_2$. If $F(A_1) \neq F(A_2)$, then ϵ is a separating word. Assume $F(A_1) = F(A_2)$, and we show that they have a separating word. Let $d_1 = \delta^*_{\mathcal{D}}(w_1)$ and $d_2 = \delta^*_{\mathcal{D}}(w_2)$. By the above, we have $d_1 \in A_1$ and $d_2 \in A_2$. Since $A_1 \neq A_2$, there is $a \in JI(\Upsilon)$ such that $f_a(d_1) \neq f_a(d_2)$. Thus, there is a word $w \in \Sigma^*$ such that $a \leq val(\mathcal{D}^{d_1}, w)$ and $a \not\leq val(\mathcal{D}^{d_2}, w)$, or the other way around. Note that since $\mathcal{D}$ is simple, we have $val(\mathcal{D}, w_1 \cdot w) = val(\mathcal{D}^{d_1}, w)$ and $val(\mathcal{D}, w_2 \cdot w) = val(\mathcal{D}^{d_2}, w)$. Since $\mathcal{D}$ and $\mathcal{D}'$ are equivalent, they assign the same values to the words $w_1 \cdot w$ and $w_2 \cdot w$, thus we have $a \leq val(\mathcal{D}', w_1 \cdot w)$ and $a \not\leq val(\mathcal{D}', w_1 \cdot w)$. Recall that two lattice values $x, y \in \Upsilon$ are equal iff for every $a \in JI(\Upsilon)$, we have $a \leq x$ iff $a \leq y$. Thus, we have that $val(\mathcal{D}', w_1 \cdot w) \neq val(\mathcal{D}', w_1 \cdot w)$, and we are done.

Note that the size of $\mathcal{D}'$ is at most the size of $\mathcal{D}$. We conclude with the following.

Theorem 5.2. *Every simple LDFA with n states has an equivalent separable simple LDFA with at most n states.*

Recall that, by Lemma 5.1, the construction of a simple LDFA blows-up the state space in a $|\Upsilon|$ multiplicative factor. Thus, we have the following.

Corollary 5.3. *Every LDFA with n states over a lattice Υ has an equivalent simple separable LDFA of size at most $n \cdot |\Upsilon|$.*

5.3. The closed-lattice synthesis problem

Consider a library $\mathcal{L}$ of box-LDFAs and a design $\mathcal{D}$. Recall that a design in the Boolean setting is a recipe to construct an LDFA by glueing the components in $\mathcal{L}$. Given a design $\mathcal{D}$, we refer to $\mathcal{A}_\mathcal{D}$ as the compositional LDFA that is constructed

using $\mathcal{D}$ and the components from $\mathcal{L}$. We can now define the *latticed closed-design* problem. Given a library of box-LDFAs $\mathcal{L}$ and a specification LDFA $\mathcal{S}$, decide whether there is a design $\mathcal{D}$ such that $\mathcal{A}_\mathcal{D}$ is equivalent to $\mathcal{S}$. Recall that in the lattice setting this means that $\mathcal{S}$ and $\mathcal{A}_\mathcal{D}$ assign the same values to all words. We assume that the components in $\mathcal{L}$ as well as $\mathcal{S}$ are all defined with respect to the same lattice.

Theorem 5.4. *The latticed closed design problem can be solved in polynomial time.*

Proof. The solution is similar to that of the closed design problem studied in Theorem 3.1. Given a library $\mathcal{L}$ of box-LDFAs and a specification LDFA $\mathcal{S}$, we construct a full information safety game $\mathcal{G}_\mathcal{S}$ such that Player 1 wins iff there is a design for $\mathcal{S}$ using the components of $\mathcal{L}$. Recall that a safety game is a turn-based two player game in which Player 1 wins iff the token that the players move stays within the "safe" vertices. Again, similar to the Boolean setting, Player 1's strategies correspond to designs. He selects the first component to gain control, and once a component relinquishes control, he selects which component gains control next.

Player 2 challenges Player 1's choice of design. In the Boolean setting, he selects the word that is read while a component is in control, which amounts to selecting an exit state from which the component relinquishes control. In the latticed setting, different runs that exit through the same exit state might have different traversal values. Player 1 should not know what the traversal value is. This has the sense of partial information, which caused the exponential blowup in the open setting. However, we are able to bypass this problem. Assume Player 1 assigns control to a components $\mathcal{B}_i$, and Player 2 selects the exit state e. Then, we maintain both the join and the meet of all possible values of runs that exit through e.

Formally, assume $\mathcal{L}$ consists of components of the form $\mathcal{B}_i = \langle \Upsilon, \Sigma, C_i, c_i^0, F_i, E_i \rangle$, for $i \in [n]$. As in the Boolean setting, we denote by $\mathcal{C}$, $\mathcal{C}_0$, and $\mathcal{E}$, the union of the sets of states, initial states, and exit states, respectively, of the components. Let $\mathcal{S} = \langle \Upsilon, \Sigma, S, s_0, F_\mathcal{S} \rangle$. By Corollary 5.3, we can assume that $\mathcal{S}$ is a simple separable LDFA.

We construct a game $\langle V, E, V_0, \alpha \rangle$, where $V_1 = \mathcal{E} \times \Upsilon \times \Upsilon \times S$ and $V_2 = \mathcal{C}_0 \times \Upsilon \times \Upsilon \times S$, $V_0 = \mathcal{C}_0 \times \{\top\} \times \{\top\} \times \{s_0\}$, and E and α are defined as follows. All the vertices in V_1 are safe, i.e., they are in α. We describe when a vertex $\langle c_0^i, \ell_\downarrow, \ell_\uparrow, s \rangle$ in V_2 is not in α. We alter the definitions we had in the Boolean setting of "infix witness" and "suffix witness" to incorporate the lattice values. First, we have that $\langle c_0^i, \ell_\downarrow, \ell_\uparrow, s \rangle$ is an infix witness if there exists words $w_1, w_2 \in \Sigma^*$ such that $\delta_i^*(w_1) = \delta_i^*(w_2) \in \mathcal{E}$ and $\delta_\mathcal{S}^*(s, w_1) \neq \delta_\mathcal{S}^*(s, w_2)$. Second, we have that $\langle c_0^i, \ell_\downarrow, \ell_\uparrow, s \rangle$ is a suffix witness if there exists a word $w \in \Sigma^*$ such that either $val(\mathcal{B}_i, w) \wedge \ell_\downarrow \neq val(\mathcal{S}^s, w)$ or $val(\mathcal{B}_i, w) \wedge \ell_\uparrow \neq val(\mathcal{S}^s, w)$.

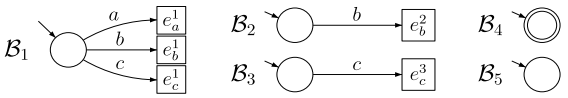
We describe the edges of the game. First, edges leaving Player 1 vertices are as in the Boolean setting and correspond to choosing the next component to gain control; we have $\langle \langle e, \ell_\downarrow, \ell_\uparrow, s \rangle, \langle c_0^i, \ell_\downarrow, \ell_\uparrow, s \rangle \rangle \in E$, for every $i \in [n]$. Vertices in $V_2 \setminus \alpha$ have no outgoing transitions. Consider a vertex $v = \langle c_0^i, \ell_\downarrow, \ell_\uparrow, s \rangle \in V_2 \cap \alpha$. Edges leaving v correspond to a choice of exit state e of $\mathcal{B}_i$. Since $v \in \alpha$, there is a state $s' \in S$ such that every word $w \in \Sigma^*$ that has $\delta_i^*(w) = e$ also has $\delta_\mathcal{S}^*(s, w) = s'$. Finally, the traversal values of these words might be different. We update the meet and join of all these traversal values. Formally, there is an edge $\langle \langle c_0^i, \ell_\downarrow, \ell_\uparrow, s \rangle, \langle e, \ell'_\downarrow, \ell'_\uparrow, s' \rangle \rangle$ iff there exists a word $w \in \Sigma^*$ such that $\delta_i^*(w) = e$ and $\delta_\mathcal{S}^*(s, w) = s'$, and $\ell'_\downarrow = \ell_\downarrow \wedge \bigwedge_{w \in \Sigma^* : \delta_i^*(w) = e} trav\text{-}val(\mathcal{B}_i, w)$ and $\ell'_\uparrow = \ell_\uparrow \wedge \bigvee_{w \in \Sigma^* : \delta_i^*(w) = e} trav\text{-}val(\mathcal{B}_i, w)$.

We claim that Player 1 wins $\mathcal{G}_\mathcal{S}$ iff there is a correct design for $\mathcal{S}$ using the component of $\mathcal{L}$. For the first direction, assume Player 1 has a winning strategy $f_\mathcal{D}$ and let $\mathcal{D}$ be the corresponding design as in Theorem 3.1. We claim that $\mathcal{D}$ is a correct design. Assume towards contradiction that there is a word $w \in \Sigma^*$ such that $val(\mathcal{A}_\mathcal{D}, w) \neq val(\mathcal{S}, w)$. Let $w = w_1 \cdot w_2 \cdot \ldots w_m$ be the partition of w according to the components that process it in $\mathcal{A}_\mathcal{D}$, and let $\ell_1, \ell_2, \ldots, \ell_m$ be the traversal values of each of the sub-words in the corresponding component. Let $\mathcal{B}_j$ be the component that processes w_m. Note that $val(\mathcal{A}_\mathcal{D}, w) = \bigwedge_{1 \leq i \leq m-1} \ell_i \wedge val(\mathcal{B}_j, w_m)$. Consider the Player 2 strategy f_w that, intuitively, selects the word $w_1 \cdot \ldots \cdot w_{m-1}$. The vertex in the game $\mathcal{G}_\mathcal{S}$ that the game reaches is of the form $v = \langle c_j^0, \ell_\downarrow, \ell_\uparrow, s \rangle \in V_2$. It is not hard to show that $\ell_\downarrow \leq \bigwedge_{1 \leq i \leq m-1} \ell_i \leq \ell_\uparrow$.

We claim that $v \notin \alpha$, which contradicts the fact that $f_\mathcal{D}$ is winning. Recall that two lattice elements are equal iff they agree on their order with respect to all join-irreducible elements. We distinguish between two cases. First, there is an element $a \in JI(\Upsilon)$ such that $a \leq val(\mathcal{A}_\mathcal{D}, w)$ and $a \not\leq val(\mathcal{S}, w)$. We claim that $a \leq (\ell_\downarrow \wedge val(\mathcal{B}_j, w_m))$. Indeed, since $a \leq val(\mathcal{A}_\mathcal{D}, w)$, we have $a \leq val(\mathcal{B}_j, w_m)$ as well as $a \leq \ell_i$, for $1 \leq i \leq m-1$. By the above, the latter implies that $a \leq \ell_\uparrow$. On the other hand, we claim that $a \not\leq val(\mathcal{S}^s, w_m)$. It is not hard to see that $\delta_\mathcal{S}^*(w_1 \cdot \ldots \cdot w_{m-1}) = s$. Since $\mathcal{S}$ is simple, $val(\mathcal{S}, w) = val(\mathcal{S}^s, w_m)$, thus the claim follows. We conclude that $\ell_\uparrow \wedge val(\mathcal{B}_j, w_m) \neq val(\mathcal{S}^s, w_m)$, implying that $v \notin \alpha$, and we are done. Showing that $v \notin \alpha$ when $a \not\leq val(\mathcal{A}_\mathcal{D}, w)$ and $a \leq val(\mathcal{S}, w)$, is done similarly using $\ell_\downarrow$.

For the other direction, assume there is a correct design $\mathcal{D}$, and we show that the corresponding Player 1 strategy $f_\mathcal{D}$ is a winning strategy in $\mathcal{G}_\mathcal{S}$. Assume otherwise, and let f_2 be a Player 2 strategy that is winning against $f_\mathcal{D}$. Let $v = \langle c_i^0, \ell_\downarrow, \ell_\uparrow, s \rangle \notin \alpha$ that the game reaches. Assume $v \notin \alpha$ because of an infix witness. The contradiction is attained similarly to Theorem 3.1. Recall that in this case there are words $w_1, w_2 \in \Sigma^*$ such that $\delta_i^*(w_1) = \delta_i^*(w_2) \in \mathcal{E}$ but $s_1 = \delta_\mathcal{S}^*(s, w_1) \neq \delta_\mathcal{S}^*(s, w_2) = s_2$. Since $\mathcal{S}$ is separable, there is a separating word w for s_1 and s_2, thus $val(\mathcal{S}^{s_1}, w) \neq val(\mathcal{S}^{s_2}, w)$. But, since $\mathcal{B}_i$ exists from the same exit state when reading w_1 and w_2, the LDFA $\mathcal{A}_\mathcal{D}$ assigns the same values to the words with suffix $w_1 \cdot w$ and $w_2 \cdot w$, while they get different values in $\mathcal{S}$.

Assume $v \notin \alpha$ because of a suffix witness. Thus, there is a word $w \in \Sigma^*$ such that either $\ell_\downarrow \wedge val(\mathcal{B}_j, w) \neq val(\mathcal{S}^s, w)$ or $\ell_\uparrow \wedge val(\mathcal{B}_j, w) \neq val(\mathcal{S}^s, w)$. We prove for the first case and the second is similar. Assume the play $out(f_\mathcal{D}, f_2)$ traverses



	$\{\mathcal{B}_1, \mathcal{B}_1\}$	$\{\mathcal{B}_1, \mathcal{B}_2\}$
$\{\mathcal{B}_3\}$	$5, 12$	$5, 13$
$\{\mathcal{B}_1\}$	$4, 8$	$6, 7$

Fig. 6. The library of the CLG with no NE, and the costs of the players in its profiles.

the components $\mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_m}$ and exit states $e_1, \ldots, e_m$. We choose words $w_1, \ldots, w_m$ that the components traverse. Thus, we need, for every $1 \leq j \leq m$ that $\delta^*_{i_j}(w_j) = e_j$. Again, we distinguish between two cases. First, let $a \in JI(\Upsilon)$ such that $a \leq (\ell_\downarrow \wedge val(\mathcal{B}_j, w))$ and $a \not\leq val(\mathcal{S}^s, w)$. Since $a \leq \ell_\downarrow$, for every $1 \leq j \leq m$, every word $u \in \Sigma^*$ such that $\delta^*_{i_j}(u) = e_j$, has $a \leq trav\text{-}val(\mathcal{B}_{i_j}, u)$. So we can choose any such word u as w_j. Note that since the intermediate vertices in the play $out(f_\mathcal{D}, f_2)$ are in α, it is not hard to show that $\delta^*_\mathcal{S}(w_1 \cdot \ldots \cdot w_m) = s$. So, we have $a \leq val(\mathcal{A}_\mathcal{D}, w_1 \cdot \ldots w_m \cdot w)$ while $a \not\leq val(\mathcal{S}, w_1 \cdot \ldots w_m \cdot w)$, and we are done. For the second case, $a \not\leq (\ell_\downarrow \wedge val(\mathcal{B}_j, w))$ and $a \leq val(\mathcal{S}^s, w)$. If $a \not\leq val(\mathcal{B}_j, w)$, then we choose for each component some word as in the above. If $a \not\leq \ell_\downarrow$, then there is $1 \leq j \leq m$ and a word $w_j \in \Sigma^*$ such that $\delta^*_{i_j}(w_j) = e_j$ and $a \not\leq trav\text{-}val(\mathcal{B}_{i_j}, w_j)$. We choose the other words as in the above, so that we have $a \not\leq val(\mathcal{A}_\mathcal{D}, w_1 \cdot \ldots \cdot w_m)$, and the proof is similar to the above. □

6. Libraries with costs and multiple users

In this section we study the setting in which several designers, each with his own specification, use the library. The construction cost of a component is now shared by the designers that use it, with the share being proportional to the number of times the component is used. For example, if $c\text{-}cost(\mathcal{B}) = 8$ and there are two designers, one that uses $\mathcal{B}$ once and a second that uses $\mathcal{B}$ three times, then the construction costs of $\mathcal{B}$ of the two designers are 2 and 6, respectively. The quality cost of a component is not shared. Thus, the cost a designer pays for a design depends on the choices of the other users and he has an incentive to share the construction costs of components with other designers. We model this setting as a multi-player game, which we dub *component library games* (CLGs, for short). The game can be thought of as a one-round game in which each player (user) selects a design that is correct according to his specification. In this section we focus on closed designs.

Formally, a CLG is a tuple $\langle \mathcal{S}_1, \ldots, \mathcal{S}_k \rangle$, where $\mathcal{L}$ is a closed component library and, for $1 \leq i \leq k$, the DFA $\mathcal{S}_i$ is a specification for Player i. A strategy of Player i is a design that is correct with respect to $\mathcal{S}_i$. We refer to a choice of designs for all the players as a *strategy profile*. Consider a profile $P = \langle \mathcal{D}_1, \ldots, \mathcal{D}_k \rangle$ and a component $\mathcal{B} \in \mathcal{L}$. The construction cost of $\mathcal{B}$ is split proportionally between the players that use it. Formally, for $1 \leq i \leq k$, recall that we use $nused(\mathcal{B}, \mathcal{D}_i)$ to denote the number of times $\mathcal{D}_i$ uses $\mathcal{B}$. For a profile P, let $nused(\mathcal{B}, P)$ denote the number of times $\mathcal{B}$ is used by all the designs in P. Thus, $nused(\mathcal{B}, P) = \sum_{1 \leq i \leq k} nused(\mathcal{B}, \mathcal{D}_i)$. Then, the construction cost that Player i pays in P for $\mathcal{B}$ is $c\text{-}cost_i(P, \mathcal{B}) = c\text{-}cost(\mathcal{B}) \cdot \frac{nused(\mathcal{B}, \mathcal{D}_i)}{nused(\mathcal{B}, P)}$. Since the quality costs of the components is not shared, it is calculated as in Section 4. Thus, the cost Player i pays in profile P, denoted $cost_i(P)$ is $\sum_{\mathcal{B} \in \mathcal{L}} c\text{-}cost_i(P, \mathcal{B}) + nused(\mathcal{B}, \mathcal{D}_i) \cdot q\text{-}cost(\mathcal{D}_i)$. We define the cost of a profile P, denoted $cost(P)$, as $\sum_{i \in [k]} cost_i(P)$.

For a profile P and a correct design $\mathcal{D}$ for Player i, let $P[i \leftarrow \mathcal{D}]$ denote the profile obtained from P by replacing the choice of design of Player i by $\mathcal{D}$. A profile P is a *Nash equilibrium* (NE) if no Player i can benefit by unilaterally deviating from his choice in P to a different design; i.e., for every Player i and every correct design $\mathcal{D}$ with respect to $\mathcal{S}_i$, it holds that $cost_i(P[i \leftarrow \mathcal{D}]) \geq cost_i(P)$.

Theorem 6.1. *There is a CLG with no NE.*

Proof. We adapt the example for multiset cost-sharing games from [7] to CLGs. Consider the two-player CLG over the alphabet $\Sigma = \{a, b, c\}$ in which Player 1 and 2's specifications are (the single word) languages $\{ab\}$ and $\{c\}$, respectively. The library is depicted in Fig. 6, where the quality costs of all components is 0, $c\text{-}cost(\mathcal{B}_1) = 12$, $c\text{-}cost(\mathcal{B}_2) = 5$, $c\text{-}cost(\mathcal{B}_3) = 1$, and $c\text{-}cost(\mathcal{B}_4) = c\text{-}cost(\mathcal{B}_5) = 0$. Both players have two correct designs. For Player 1, the first design uses $\mathcal{B}_1$ twice and the second design uses $\mathcal{B}_1$ once and $\mathcal{B}_2$ once. There are also uses of $\mathcal{B}_4$ and $\mathcal{B}_5$, but since they can be used for free, we do not include them in the calculations. For Player 2, the first design uses $\mathcal{B}_3$ once, and the second design uses $\mathcal{B}_1$ once. The table in Fig. 6 shows the players' costs in the four possible CLG's profiles, and indeed none of the profiles is a NE. □

We study computational problems for CLGs. The most basic problem is the *best-response problem* (BR problem, for short). Given a profile P and $i \in [k]$, find the cheapest correct design for Player i with respect to the other players' choices in P. Apart from its practical importance, it is an important ingredient in the solutions to the other problems we study. The next problem we study is finding the *social optimum* (SO, for short), namely the profile that minimizes the total cost of all players; thus the one obtained when the players obey some centralized authority. For both the BR and SO problems, we study the decision (rather than search) variants, where the input includes a threshold μ. Finally, since CLGs are not guaranteed to have a NE, we study the problem of deciding whether a given CLG has a NE. We term this problem ∃NE.

Definition 6.1. We define the decision problems formally as follows. Let $\mathcal{G}$ be a CLG.

BR An input $\langle \mathcal{G}, P, i, \mu \rangle$ is in BR, where P is a profile, $i \in [k]$, and $\mu \in \mathbb{R}$, iff there is a design $\mathcal{D}_i$ that is correct with respect to $\mathcal{S}_i$ such that $cost_i(P[i \leftarrow \mathcal{D}_i]) \leq \mu$.
SO An input $\langle \mathcal{G}, \mu \rangle$ is in SO, where $\mu \in \mathbb{R}$, iff there is a profile P such that $cost(P) \leq \mu$.
∃NE An input $\langle \mathcal{G} \rangle$ is in ∃NE iff there is a NE profile in $\mathcal{G}$.

Note that the BCD problem studied in Section 4 is a special case of BR problem when there is only one player. Also, in a setting with a single player, the SO and BR problems coincide, thus the lower bound of Theorem 4.3 applies to them. In Lemma 4.1 we showed that if there is a correct design $\mathcal{D}$ with $cost(\mathcal{D}) \leq \mu$, then there is also a correct design $\mathcal{D}'$, based on a memoryless strategy and hence having polynomially many states, such that for every component $\mathcal{B}$, we have $nused(\mathcal{D}', \mathcal{B}) \leq nused(\mathcal{D}, \mathcal{B})$. The arguments there apply in the more general case of CLGs. Thus, we have the following.

Theorem 6.2. *The BR and SO problems are NP-complete.*

We continue to study the ∃NE problem. We show that ∃NE is complete for Σ_2^P – the second level of the polynomial hierarchy. Namely, decision problems solvable in polynomial time by a nondeterministic Turing machine augmented by an *oracle* for an NP-complete problem. An oracle for a computational problem is a black box that is able to produce a solution for any instance of the problem in a single operation. Thus, for every problem $P \in \Sigma_2^P$ there is a machine such that for every $x \in P$ there is a polynomial-time accepting computation (with polynomially many queries to the oracle). As co-NP is the dual complexity class of NP, the dual complexity class of Σ_2^P is Π_2^P. Thus, a problem P is Σ_2^P-complete iff its complement $\overline{P}$ is Π_2^P-complete.

Theorem 6.3. *The ∃NE problem is Σ_2^P-complete.*

Proof. For the upper bound, we describe a nondeterministic Turing machine with an oracle for the SBR problem – the strict version of the BR problem, where we seek a design whose cost is strictly smaller than μ. Given a CLG $\mathcal{G} = \langle \mathcal{S}_1, \ldots, \mathcal{S}_k \rangle$, the machine guesses a profile $P = \langle \mathcal{D}_1, \ldots, \mathcal{D}_k \rangle$, where for $1 \leq i \leq k$, the design $\mathcal{D}_i$ has at most $|\mathcal{C}_0 \times S_i|$ states, where S_i are the states of $\mathcal{S}_i$. First, we check whether P is a profile of correct designs. That is, for $i \in [k]$, we check whether $\mathcal{D}_i$ is a correct design with respect to $\mathcal{S}_i$, which can be done in polynomial time. If there is an incorrect design we terminate and reject. Next, we check whether P is a NE profile by checking if there is a player that has a beneficial move from P. That is, for $i \in [k]$, we feed the oracle the input $\langle \mathcal{G}, P, i, cost_i(P) \rangle$. If the oracle answers YES, then Player i can benefit from deviating and P is not a NE in which case we reject. On the other hand, if for every $i \in [k]$ the oracle answers NO, then P is a NE in which case we accept. Clearly the machine recognizes ∃NE. Note that if $\mathcal{G} \in$ ∃NE, one of the profiles P we guess is a NE, and the computation in which we guess P is a polynomial accepting computation.

For the lower bound, we show a reduction from the complement of the Π_2^P-complete problem *min-max vertex cover* [25] (MMVC, for short). The input to the MMVC problem is $\langle G, I, J, N, \mu \rangle$, where $G = \langle V, E \rangle$ is an undirected graph, I and J are sets of indices, $N : V \to \{V_{i,j} \subseteq V : i \in I \text{ and } j \in J\}$ partitions the vertices, and $\mu \in \mathbb{N}$ is a value. Note that since G is undirected, its edges are sets with two vertices. We refer to the sets in the partition $\{V_{i,j}\}_{i \in I, j \in J}$ as *neighborhoods* and for $v \in V$ we refer to $N(v)$ as the neighborhood of v. Note that there is a coarser partition of V, namely $\{V_i\}_{i \in I}$, where $V_i = \bigcup_{j \in J} V_{i,j}$. We refer to the elements in this partition as *districts* and, for $v \in V$, use $D(v)$ to denote the district v belongs to. For a function $t : I \to J$ we define $V_t = \bigcup_{i \in I} V_{i,t(i)}$. Intuitively, t is a choice of neighborhood in each district. Let $G_t = \langle V_t, E_t \rangle$ be the induced subgraph of G on the vertex set V_t. Formally, for $e \in E$, we have $e \in E_t$ iff $e \subseteq V_t$. For a graph G, we say that $V' \subseteq V$ is a vertex cover of G if for every $e = \{u, v\} \in E$ we have $V' \cap \{u, v\} \neq \emptyset$. An input $\langle G, I, J, N, \mu \rangle$ is in MMVC iff for any choice of neighborhoods in the districts given by a function t, the smallest vertex cover of the resulting graph is at most μ. Formally, $\max_{t \in J^I} \min\{|V'| : V' \subseteq V_t \text{ is a vertex cover of } G_t\} \leq \mu$. We assume without loss of generality that $\mu \leq |V|$.

Consider an input $\langle G, I, J, N, \mu \rangle$ to MMVC. We construct a CLG $\mathcal{G}$ with library $\mathcal{L}$ and specifications $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{|I|}$ such that $\mathcal{G}$ has a NE iff $\langle G, I, J, N, \mu \rangle \notin$ MMVC. The alphabet Σ consists of letters $i, \tilde{i} \in I$, $v, \tilde{v} \in V$, and $e \in E$. Let $E = \{e_1, \ldots, e_m\}$. The specification of Player 0 consists of words of length $3|E|$ of the form $v_1 \tilde{v}_1 e_1 \ldots v_m \tilde{v}_m e_m$, for some $v_1, \ldots, v_m \in V$ (allowing duplicates). For $i \in I$, the specifications of Player i consist of the single word $i \cdot (\tilde{i})^\ell$, where ℓ is a polynomial in $|V|$, which we define in the following.

We describe the components of $\mathcal{L}$ (see Fig. 7). When describing the components' costs we only refer to the construction cost as their quality costs are 0. The simplest components are $\mathcal{B}_{acc}$ and $\mathcal{B}_{rej}$. They consist of a single initial state that is accepting in $\mathcal{B}_{acc}$ and rejecting in $\mathcal{B}_{rej}$. The cost of both these components is 0. The next component is $\mathcal{B}_0$, which is exactly $\mathcal{S}_0$, and its cost is $\mu + 1$. The rest of the components have no accepting states. We describe these components by the words that they can process. For each word there is a unique disjoint path from the initial to a separate exit state. For every neighborhood $V_{i,j}$ there is a component $\mathcal{B}_{i,j} \in \mathcal{L}$ that costs $(3|E| + 2)(\mu + 1)$. We refer to these components as *neighborhood components*. The single-lettered words it can process are i, $\tilde{i}$, and v for $v \in V_{i,j}$. Also, it can process the words $v\tilde{v}e$ for $v \in V \setminus V_{i,j}$ and $e \in E$, and $\tilde{v}e$ for $v \in V_{i,j}$ and $e \in E$ such that $e \cap (V_i \setminus V_{i,j}) \neq \emptyset$. For every $v \in V$ there

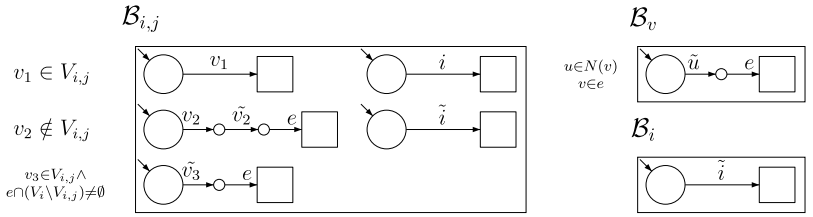


Fig. 7. Some of the components in the library produced by the reduction from MMVC.

is a component $\mathcal{B}_v$ that costs 1. We refer to these components as *vertex components*. The words it can process are $\tilde{u}e$ for $u \in N(v)$ and $e \in E$ such that $v \in e$. For $i \in I$ there is a component $\mathcal{B}_i$ that can process the word $\tilde{i}$ and costs a very small value $\xi > 0$. The construction is clearly polynomial in the input.

We claim that $\langle G, I, J, N, \mu\rangle \notin$ MMVC iff $\mathcal{G} = \langle \mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_{|I|}\rangle \in \exists$NE. For the first direction, assume $\langle G, I, J, N, \mu\rangle \notin$ MMVC. Thus, there is a function $t : I \to J$ such that every vertex cover of V_t has more than μ vertices. We claim that the CLG $\mathcal{G}$ has a NE. Consider the profile P in which Player 0 uses only the component $\mathcal{B}_0$ and, for $i \in I$, Player i uses the design $\mathcal{D}_{i,t(i)}$, which we describe below, and uses only the neighborhood component $\mathcal{B}_{i,t(i)}$ and the components $\mathcal{B}_{acc}$ and $\mathcal{B}_{rej}$. For $i \in I$ and $j \in J$, we describe the design $\mathcal{D}_{i,j}$. In $\mathcal{D}_{i,j}$, the component $\mathcal{B}_{i,j}$ gains initial control. If the component relinquishes control after reading the single-lettered word i, it regains control. If it relinquishes control after reading any other word, the design assigns control to $\mathcal{B}_{rej}$. Similarly, for ℓ times, control is given to $\mathcal{B}_{i,j}$ assuming it reads the word $\tilde{i}$, and otherwise control is given to $\mathcal{B}_{rej}$. After $\mathcal{B}_{i,j}$ gains control $\ell + 1$ times, control is assigned to $\mathcal{B}_{acc}$. Clearly, $L(\mathcal{D}_{i,j}) = L(\mathcal{S}_i)$, thus it is a correct design for Player i.

Assume towards contradiction that P is not a NE. Thus, there is a player that benefits from deviating. We start by showing that, for $i \in I$, Player i cannot benefit from deviating. Note that for $j \in J$, no other player uses a component $\mathcal{B}_{i,j}$ in P. Thus, deviating to a design $\mathcal{D}_{i,j}$, for $j \neq t(i)$, would result in the same cost for Player i, and deviating to any other correct design would increase his cost, and is clearly not beneficial.

We continue to show that Player 0 cannot benefit from a deviation. Assume towards contradiction that there is a correct design $\mathcal{D}$ such that $cost_0(P) > cost_0(P[0 \leftarrow \mathcal{D}])$. Consider the set $V' \subseteq V$ of vertices that correspond to vertex components that are used in $\mathcal{D}$, thus $v \in V'$ iff $\mathcal{B}_v \in used(\mathcal{D})$. Recall that $c\text{-}cost(\mathcal{B}_0) = \mu + 1$, thus $cost_0(P) = \mu + 1$. Since the construction costs of the vertex components is 1 and Player 1 does not share them, we have $|V'| \leq \mu$.

We claim that there is a vertex cover $V'' \subseteq V' \cap V_t$ for G_t. Since $|V'| \leq \mu$, this would contradict our assumption that $\langle G, I, J, N, \mu\rangle \notin$ MMVC. Recall that the cost of using a neighborhood component without sharing is more than $\mu + 1$. Since we assume Player 0 benefits from deviating to $\mathcal{D}$, he must share all his uses of these components. Since the neighborhood components that are used by the players $1, \ldots, |I|$ are exactly these that are dictated by t, every neighborhood component $\mathcal{B}_{i,j} \in used(\mathcal{D})$ has $j = t(i)$.

We claim that for every $e \in E_t$ there is a vertex $v \in V' \cap V_t$ such that $v \in e$. Let $E_t = \{e_{i_1}, \ldots, e_{i_l}\}$ and $j \in [l]$. Consider the word $w = v\tilde{v}e_1 v\tilde{v}e_2 \ldots v\tilde{v}e_{i_j - 1}$ for some $v \in V$. Since w is a prefix of a word in $L(\mathcal{S}_0)$, the run of $\mathcal{A}_{\mathcal{D}}$ on w does not get stuck. Since the last letter in w is in E the component that gains control after reading w is a neighborhood component, thus it is $\mathcal{B}_{i,t(i)}$ for some $i \in I$.

Consider the word $w' = w \cdot u\tilde{u}e_{i_j}$ for $u \in V_{i,t(i)}$. Again, since w' is a prefix of a word in $L(\mathcal{S}_0)$ the run of $\mathcal{A}_{\mathcal{D}}$ on w' does not get stuck. Since $u \in V_i$, $\mathcal{B}_{i,t(i)}$ relinquishes control after reading u. We claim that a vertex component $\mathcal{B}_v$ must gain control next. Indeed, since $u \in V_{i,t(i)}$, the only neighborhood component that is a candidate to process the word $\tilde{u}e$ is $\mathcal{B}_{i,t(i)}$. However, since $u \in G_t$ if it has an endpoint that belongs to the district V_i, then the endpoint is in $V_{i,t(i)}$. Thus, $\mathcal{B}_{i,t(i)}$ cannot process $\tilde{u}e$ and it is processed by a vertex component $\mathcal{B}_v$. Note that since it can process the word $\tilde{u}e$, we have $u \in N(v)$, thus $N(v) = V_{i,t(i)}$. Moreover, $v \in e$. Clearly $v \in V'$ as $\mathcal{D}$ uses $\mathcal{B}_v$, and we are done.

For the second direction, assume $\langle G, I, J, N, \mu\rangle \in$ MMVC. Assume towards contradiction that there is a NE profile P in $\mathcal{G}$. We distinguish between two cases. In the first case, Player 0 does not use $\mathcal{B}_0$. Recall that Player 0 has a correct design that uses only $\mathcal{B}_0$ and costs $\mu + 1$. Moreover, every correct design that does use $\mathcal{B}_0$ must use a neighborhood component $\mathcal{B}_{i,j}$, which costs $(3|E|+2)(\mu+1)$. Since P is a NE, Player i, the only player that can use $\mathcal{B}_{i,j}$, uses $\mathcal{B}_{i,j}$ at least $3|E|+1$ times. We claim that Player i has a beneficial deviation from P, contradicting the fact that P is a NE. First, we bound $cost_i(P)$. Clearly, Player 0 uses $\mathcal{B}_{i,j}$ at most $3|E|$ times, thus $cost_i(P) \geq \frac{3|E|+1}{3|E|+3|E|+1} \cdot cost(\mathcal{B}_{i,j})$. We describe a beneficial deviation for Player i. Consider the design $\mathcal{D}$ that assigns initial control to $\mathcal{B}_{i,j}$. If it relinquishes control after reading anything different from i, $\mathcal{D}$ assigns control to $\mathcal{B}_{rej}$. Otherwise, the component $\mathcal{B}_i$, which can process only the word $\tilde{i}$, gains control ℓ consecutive times after which $\mathcal{B}_{acc}$ gains control. It is not hard to see that $L(\mathcal{D}) = L(\mathcal{S}_i)$. Since $nused(\mathcal{D}, \mathcal{B}_{i,j}) = 1$ and $nused(\mathcal{D}, \mathcal{B}_i) = \ell$, we have $cost_i(P[i \leftarrow \mathcal{D}]) \leq \frac{1}{2} cost(\mathcal{B}_{i,j}) + \xi$. For sufficiently small ξ, we have $cost_i(P) > cost_i(P[i \leftarrow \mathcal{D}])$, and we are done.

In the second case, the design Player 0 chooses in P is the design that uses only the component $\mathcal{B}_0$. Thus, $cost_0(P) = \mu + 1$. Recall that for every $i \in I$ and $j \in J$, the design $\mathcal{D}_{i,j}$ is a correct design for Player i that uses only the neighborhood component $\mathcal{B}_{i,j}$. It is not hard to see that since P is a NE, every Player i chooses some design $\mathcal{D}_{i,j}$. Let $t : I \to J$ be the function that corresponds to these players choices. Since $\langle G, I, J, N, \mu\rangle \in$ MMVC, there is a vertex cover $V' \subseteq V_t$ of G_t such that $|V'| \leq \mu$.

We construct a design $\mathcal{D}$ for Player 0 that is a beneficial deviation from P. Recall that Player 0's specification is the set of words of length $3|E|$ of the form $v_1\tilde{v_1}e_1 \dots v_m\tilde{v_m}e_m$, where $E = \{e_1, \dots, e_m\}$ and $v_1, \dots, v_m \in V$ (allowing duplicates). The definition of $\mathcal{D}$ is inductive. Let $1 \leq l \leq |E|$. Consider a word $w \in \Sigma^*$ of length $3(l-1)$ that can be extended to a word in $L(\mathcal{S}_0)$. That is, there is a word $x \in \Sigma^*$ such that $w \cdot x \in L(\mathcal{S}_0)$. For $v \in V$, let $w_v = w \cdot v\tilde{v}e_{l+1}$. Note that w_v can also be extended to a word in $L(\mathcal{S}_0)$ (possibly with ϵ). Assuming the run of $\mathcal{A}_{\mathcal{D}}$ on w does not get stuck and control is relinquished from some component after reading w, we describe how $\mathcal{D}$ assigns control next such that the run of $A_{\mathcal{D}}$ on every w_v does not get stuck and control is relinquished after reading w_v.

We distinguish between two cases. In the first case, $e_l \notin E_t$. Thus, there is a vertex $v \in e_l \setminus V_t$. Let V_i be v's district, thus $V_i = D(v)$. Let $V_{i,j} \subseteq V_i$ be the neighborhood in V_i that is selected by t, thus $j = t(i)$. Note that this since $v \notin V_t$, it does not belong to $V_{i,j}$. The component to which $\mathcal{D}$ assigns control after reading w is the neighborhood component $\mathcal{B}_{i,j}$. Consider a vertex $u \in V$. If $u \notin V_{i,j}$, then when reading $u\tilde{u}e$ the run in $\mathcal{B}_{i,j}$ does not get stuck and control is relinquished at its end. If $u \in V_{i,j}$, then $\mathcal{B}_{i,j}$ relinquishes control after reading u. In such a case we define $\mathcal{D}$ to reassign control to $\mathcal{B}_{i,j}$. Note that the run of $\mathcal{B}_{i,j}$ on $\tilde{u}e_l$ does not get stuck. Indeed, the vertex $v \in e_l$ is a witness to the fact that $(V_i \setminus V_{i,j}) \cap e_l \neq \emptyset$. Clearly, control is relinquished after $\mathcal{B}_{i,j}$ reads $\tilde{u}e_l$. If in one of the times $\mathcal{B}_{i,j}$ gains control it relinquishes it after reading any other word $x \in \Sigma^*$, then there is no $y \in \Sigma^*$ such that $w \cdot x \cdot y \in L(\mathcal{S}_0)$, and we assign control to $\mathcal{B}_{rej}$.

In the second case, $e_l \in E_t$. Thus, there is a vertex v in the vertex cover V' such that $v \in e_l$. Let $N(v) = V_{i,j}$. Note that since $v \in V_t$, we have $t(i) = j$. The component to which $\mathcal{D}$ assigns control after reading w is the neighborhood component $\mathcal{B}_{i,j}$. Consider a vertex $u \in V$. The case where $u \notin V_{i,j}$ is similar to the previous case. For $u \in V_{i,j}$, when $\mathcal{B}_{i,j}$ reads u, it relinquishes control. In such a case, $\mathcal{D}$ assigns control to $\mathcal{B}_v$. Since $v \in V_{i,j}$ and $v \in e_l$, the component $\mathcal{B}_v$ can process the word $\tilde{u}e_l$, and it relinquishes control after its end. Similarly to the above, if $\mathcal{B}_{i,j}$ or $\mathcal{B}_v$ relinquish control after reading any other word $x \in \Sigma^*$, then we assign control to $\mathcal{B}_{rej}$. Finally, if $l = |E|$, we assign control $\mathcal{B}_{acc}$.

Correctness of the design $\mathcal{D}$ is immediate from the construction. We claim that $cost_0(P[0 \leftarrow \mathcal{D}]) < \mu + 1$. Note that the $\mathcal{D}$ uses two types of components. The first type is neighborhood components. Consider $\mathcal{B}_{i,j} \in used(\mathcal{D})$. Note that we constructed $\mathcal{D}$ so that $j = t(i)$. Thus, Player 0 shares the cost of $\mathcal{B}_{i,j}$ with Player i. Recall that Player i chooses the design $\mathcal{D}_{i,j}$ that uses $\mathcal{B}_{i,j}$ ℓ times. We define ℓ to be sufficiently large so that the proportion of the cost that Player 0 pays is less than $\frac{1}{2|E|}$. Thus, the total cost Player 0 endures for neighborhood components is less than 1. The second type of components $\mathcal{D}$ uses is vertex components. Since $\mathcal{D}$ uses vertex components that correspond to vertices in the vertex cover V', the number of such component that $\mathcal{D}$ uses is at most μ, which is the total cost for these components. Thus, $cost_0(P[0 \leftarrow \mathcal{D}]) < \mu + 1$, and we are done. □

7. Discussion

Traditional synthesis algorithms assumed that the system is constructed from scratch. Previous work adjusted synthesis algorithms to a reality in which systems are constructed from component libraries. We adjust the algorithms further, formalize the notions of quality and cost and seek systems of high quality and low cost. We argue that one should distinguish between quality considerations, which are independent of uses of the library by other designs, and pricing considerations, which depend on uses of the library by other designs.

Once we add multiple library users to the story, synthesis is modeled by a resource-allocation game and involves ideas and techniques form algorithmic game theory. In particular, different models for sharing the price of components can be taken. Recall that in our model, users share the price of a component, with the share being proportional to the number of uses. In some settings, a *uniform sharing rule* may fit better, which also makes the game more stable. In other settings, a more appropriate sharing rule would be the one used in *congestion games* – the more a component is used, the higher is its price, reflecting, for example, a higher load. Somewhat surprising, games with congestion effects turn out to be more stable than cost-sharing games [9]. Still, the complexity of the decision problems we study here for CLGs match the ones for CLGs with congestion effects. Moreover, synthesis of different specifications in different times gives rise to *dynamic allocation* of components, and synthesis of collections of specifications by different users gives rise to *coalitions* in the games. These notions are well studied in algorithmic game theory and enable an even better modeling of the rich settings in which traditional synthesis is applied.

References

[1] M. Abadi, L. Lamport, P. Wolper, Realizable and unrealizable concurrent program specifications, in: Proc. 25th ICALP, in: LNCS, vol. 372, Springer, 1989, pp. 1–17.

[2] S. Almagor, U. Boker, O. Kupferman, Formalizing and reasoning about quality, J. ACM 63 (3) (2016) 24:1–24:56.

[3] G. Alonso, F. Casati, H.A. Kuno, V. Machiraju, Web Services – Concepts, Architectures and Applications, Data-Centric Systems and Applications, Springer, 2004.

[4] R. Alur, K. Etessami, P. Madhusudan, A temporal logic of nested calls and returns, in: Proc. 10th TACAS, in: LNCS, vol. 2725, Springer, 2004, pp. 67–79.

[5] B. Aminof, F. Mogavero, A. Murano, Synthesis of hierarchical systems, Sci. Comput. Program. 83 (2014) 56–79.

[6] G. Avni, O. Kupferman, When does abstraction help?, Inform. Process. Lett. 113 (2013) 901–905.

[7] T. Tamir, private communication.

[8] G. Avni, O. Kupferman, T. Tamir, Network-formation games with regular objectives, in: Proc. 17th FoSSaCS, in: LNCS, vol. 8412, Springer, 2014, pp. 119–133.

[9] G. Avni, O. Kupferman, T. Tamir, Congestion games with multisets of resources and applications in synthesis, in: Proc. 35th FSTTCS, in: LIPIcs, vol. 45, Schloss Dagstuhl, 2015, pp. 365–379.
[10] D. Berwanger, L. Doyen, On the power of imperfect information, in: Proc. 28th TST& TCS, 2008, pp. 73–82.
[11] A. Bohy, V. Bruyère, E. Filiot, J-F. Raskin, Synthesis from LTL specifications with mean-payoff objectives, in: Proc. 19th TACAS, in: LNCS, vol. 7795, Springer, 2013, pp. 169–184.
[12] U. Boker, K. Chatterjee, T.A. Henzinger, O. Kupferman, Temporal specifications with accumulative values, in: Proc. 26th LICS, 2011, pp. 43–52.
[13] L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, M. Stoelinga, Model checking discounted temporal properties, Theoret. Comput. Sci. 345 (1) (2005) 139–170.
[14] L. de Alfaro, T.A. Henzinger, Interface theories for component-based design, in: Proc. 1st EMSOFT, in: LNCS, vol. 2211, Springer, 2001, pp. 148–165.
[15] L. Doyen, T.A. Henzinger, B. Jobstmann, T. Petrov, Interface theories with component reuse, in: Proc. 8th EMSOFT, 2008, pp. 79–88.
[16] M. Droste, W. Kuich, H. Vogler (Eds.), Handbook of Weighted Automata, Springer, 2009.
[17] J. Elgaard, N. Klarlund, A. Möller, Mona 1.x: new techniques for WS1S and WS2S, in: Proc. 10th CAV, in: LNCS, vol. 1427, Springer, 1998, pp. 516–520.
[18] A. Fabrikant, C. Papadimitriou, K. Talwar, The complexity of pure Nash equilibria, in: Proc. 36th STOC, 2004, pp. 604–612.
[19] M. Faella, A. Legay, M. Stoelinga, Model checking quantitative linear time logic, Electron. Notes Theor. Comput. Sci. 220 (3) (2008) 61–77.
[20] E. Filiot, N. Jin, J.-F. Raskin, Antichains and compositional algorithms for LTL synthesis, Form. Methods Syst. Des. 39 (3) (2011) 261–296.
[21] G. Gößler, J. Sifakis, Composition for component-based modeling, Sci. Comput. Program. 55 (1–3) (2005) 161–183.
[22] S. Halamish, O. Kupferman, Minimizing deterministic lattice automata, in: Proc. 14th FoSSaCS, in: LNCS, vol. 6604, Springer, 2011, pp. 199–213.
[23] N.D. Jones, W.T. Laaser, Complete problems for deterministic polynomial time, Theoret. Comput. Sci. 3 (1) (1976) 105–117.
[24] M. Jurdzinski, Small progress measures for solving parity games, in: Proc. 17th STACS, in: LNCS, vol. 1770, Springer, 2000, pp. 290–301.
[25] K-I. Ko, C-L. Lin, On the complexity of min–max optimization problems and their approximation, in: Minimax and Applications, in: Nonconvex Optimization and Its Applications, vol. 4, Springer, 1995, pp. 219–239.
[26] O. Kupferman, Y. Lustig, Lattice automata, in: Proc. 8th VMCAI, in: LNCS, vol. 4349, Springer, 2007, pp. 199–213.
[27] O. Kupferman, N. Piterman, M.Y. Vardi, Safraless compositional synthesis, in: Proc. 18th CAV, in: LNCS, vol. 4144, Springer, 2006, pp. 31–44.
[28] D. Krob, The equality problem for rational series with multiplicities in the tropical semiring is undecidable, Internat. J. Algebra Comput. 4 (3) (1994) 405–425.
[29] O. Kupferman, M.Y. Vardi, Safraless decision procedures, in: Proc. 46th FOCS, 2005, pp. 531–540.
[30] Y. Lustig, M.Y. Vardi, Synthesis from component libraries, Int. J. Softw. Tools Technol. Transf. 15 (2013) 603–618.
[31] M. Mohri, Finite-state transducers in language and speech processing, Comput. Linguist. 23 (2) (1997) 269–311.
[32] A. Pnueli, R. Rosner, On the synthesis of a reactive module, in: Proc. 16th POPL, 1989, pp. 179–190.
[33] J.-F. Raskin, K. Chatterjee, L. Doyen, T. Henzinger, Algorithms for ω-regular games with imperfect information, Log. Methods Comput. Sci. 3 (3) (2007).
[34] R.W. Rosenthal, A class of games possessing pure-strategy Nash equilibria, Internat. J. Game Theory 2 (1973) 65–67.
[35] T. Roughgarden, E. Tardos, How bad is selfish routing?, J. ACM 49 (2) (2002) 236–259.
[36] S. Safra, On the complexity of ω-automata, in: Proc. 29th FOCS, 1988, pp. 319–327.
[37] A.P. Sistla, E.M. Clarke, The complexity of propositional linear temporal logic, J. ACM 32 (1985) 733–749.