Control of input/output discrete-event systems

Citation for published version (APA):

Document status and date:
Published: 01/01/2008

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 15. Nov. 2023
Abstract

A class of control problems for discrete-event systems is proposed, inspired by applications in the domain of high-tech systems. The control problem asks for controllers which generate control inputs based on the outputs. We formalize the above control problem, whereby the plant behavior is modeled as an input-output relation recognizable by a finite-state quasi-sequential deterministic transducer, and the controller is modeled as a sequential map realizable by a Moore-automaton. The control objective is formalized as a language over the alphabet of internal (unobservable) events generated by the plant. We propose a solution to the control problem above by reducing it to a Ramadge-Wonham control problem with partial observations.  

1This work was supported by the ITEA project Twins 05004; the DARWIN project at Philips Healthcare under the responsibility of the Embedded Systems Institute, partially supported by the Netherlands Ministry of Economic Affairs under the BSIK program.
1 Introduction

Motivated by applications in the area of high-tech systems, in particular, printers [35] and MRI scanners [37], we consider the following control problem.

Control problem The plant we are interested in changes its internal state under the influence of inputs and it generates outputs and internal events. We distinguish three kinds of inputs; control inputs, invisible disturbances and external inputs. The external inputs and disturbances are imposed by the environment (which includes the user). The control inputs are the ones which can be used by the controller to influence the plant behavior. The external inputs, control inputs, disturbances, outputs and internal events are allowed to happen simultaneously. In other words, the systems of interest react to any pair of control inputs and external inputs by generating outputs and internal events. The control inputs, external inputs, disturbances, outputs and internal events are all sequences of symbols from a finite alphabet. The plant behaves more like a classical discrete-time control system than an event-driven one. There is a number of applications, including those from [35, 37], where this paradigm naturally fits the control problem at hand. In particular, time-driven systems with discrete-valued inputs and outputs fit the paradigm described above.

A controller reads the outputs generated by the plant and the external inputs and generates a control input for each possible history. It is activated on sampling times or whenever an output event occurs. The objective is to make sure that the closed-loop system generates finite or infinite sequences of internal events which belong to the language of the control requirements.

Contribution of the paper The contribution of the paper can be summarized as follows.

1. Formalization of the control problem We formalize the control problem sketched above. We model the distinction between inputs and outputs explicitly. That is, the plant is a dynamical system which generates outputs under the influence of inputs. Similarly, we model controllers as dynamical systems which read sequences of outputs and generate control symbols which are fed back to the plant. Mathematically, the external behavior of the plant is an input-output relation mapping finite strings to finite strings. The underlying state-space representation of the plant is a so called quasi-sequential deterministic transducer. The mathematical model of the controller is a sequential input-output function, and the underlying state-space representation is a finite-state deterministic Moore-automaton [13, 16]. Notice that we also allow non-deterministic plant models.

2. Rigorous explanation of the role of non-determinism and partial observations We show that without loss of generality, the plant can be viewed as a deterministic plant subject to invisible disturbances. The non-determinism which is possibly present in the original plant can then be attributed to the effect of the invisible disturbances. More precisely, we will show that for each instance of the control problem described above, we can formulate another control problem. The control requirements of this new control problem are the same as those of the original one. However, in contrast to the original control problem, the plant of the new control problem is deterministic and has no external inputs. Any solution of the new control problem is a solution of the original one and vice versa. In addition, the deterministic plant of the new control problem can then be represented as the input-output map of a deterministic sequential automaton. The transformation above is computationally effective.
Although the underlying idea of the transformation is intuitively quite simple, it was never formalized, to the best of our knowledge. We believe that this result clarifies the relationship between non-determinism and partial observations. In turn, the presence of this relationship was noticed by several authors, see for example [25, 1, 26, 39].

3. **Rigorous solution of the control problem** We show that despite apparent differences, the formulated control problem can be transformed to a classical Ramadge-Wonham control problem (abbreviated as RW), and solved using RW theory. The latter contribution illustrates vividly the versatility of the classical RW framework. The proposed transformation is theoretically sound and computable, but it works only under additional assumptions. The general case can be treated using Rabin-parity-games [23, 19]. We defer the treatment of the general case to another paper.

Informally, the proposed solution of the control problem consist of the following steps:

1. Provide a model of the plant as a quasi-sequential deterministic transducer.
2. Provide a model of the control requirements as finite-state automaton.
3. Construct a finite-state automaton from the quasi-sequential deterministic transducer which models the plant. This finite-state automaton will play the role of the new plant model of the corresponding RW control problem.
4. Likewise, transform the finite-state automaton of the requirements to a finite-state automaton (defined over a different alphabet) which will be interpreted as the automaton recognizing the control requirements of the corresponding RW control problem.
5. Solve the corresponding RW control problem (with partial observations) for the plant automaton and the automaton of the control requirements obtained in the previous steps and obtain a (not necessarily maximally permissive) supervisor.
6. Extract the Moore-automaton of the controller from the finite-state automaton implementing the supervisor.

Note that the transformations of Step 3,4 and 6 are not only formally defined, but they have been implemented in software. Step 5 can be carried out using the standard tool TCT for RW supervisory synthesis. The results of the paper allow the application of RW theory to control problems which do not directly fit the RW framework. We do not deal with issues such as modular and distributed control in this paper.

**Motivation for the new formalism** Formally, the control problem considered in this paper is different from RW formalism. The motivation for using a separate formalism instead of the RW framework is the following.

1. **More accurate formalization of physical constraints** Our framework formalizes those physical aspects of the control problem, which are relevant for the correct solution, but are not made explicit and formal within the RW framework. Our framework explicitly formalizes the distinction between inputs and outputs, and which events are generated by the plant, which by the environment and which by the controller. The RW framework leaves these issues unspecified. Moreover, in contrast to a supervisor in RW framework which just enables or disables events,
a controller in our setting actively generates events. While this generality makes the RW framework very flexible, it may also lead to difficulties in applications, [3, 12, 2, 9, 14, 11, 31]. More precisely, it becomes challenging to translate the practical control problem to the RW framework, and to come up with a correct implementation of the controller. Notice that RW theory guarantees only that the interconnection of the plant with a supervisor satisfies the control requirements. It does not say anything about whether the implementation of the controller together with the physical plant satisfies the control requirements. Hence, if the gap between the physical system and the formal model is too big, then the resulting controller might still fail to be correct. While there is always a gap between the reality and the model, decreasing this gap is important for practice. We believe that the formalization proposed in this paper decreases this gap.

In the light of this, the contribution of the paper can also be viewed as a formal framework for applying RW theory to control problem arising in high-tech systems. More precisely, Step 1 and Step 3 can be viewed as a formalization of a strategy to build RW plant models. The corresponding transformation of the control requirements, i.e. Step 2 and 4 can be viewed as formalization of a modeling strategy to model control requirements. Finally, Step 6 can be viewed as a formalization of a method for implementing the obtained supervisor as a controller.

Using engineering intuition, many of the challenges arising in the application of RW framework can be overcome. However, a systematic and formal approach to application and implementation of RW framework offers several advantages. One of the advantages is that it may make it easier to apply RW theory for automated generation of control software. In many applications, the challenge is not so much to come up with a control algorithm, but to come up with a method for fast adaptation of the control algorithms to the ever changing control requirements and plant specifications. In particular, this is the case in complex systems, new generation of which have to be designed every few years. One particular approach is to use control theory for automatic synthesis of control algorithms and software. In principle, the RW framework could be a suitable tool for this. However, despite successes [37], the challenges regarding implementation and modeling in RW framework mean that significant human input is necessary. This makes the design process more time consuming and error-prone. While it is impossible to fully automate this process, we believe that the presented framework will make control software generation less dependent on human input.

2. **Abstraction of hybrid systems** Another motivation is that many systems of practical importance exhibit hybrid, i.e. both discrete and continuous behavior. One widely accepted approach in control of hybrid systems is to approximate them by a finite-state system and use techniques (for example RW theory) for control of the finite-state system. However, it is not always clear how to translate the resulting event-driven controller to a time-driven controller necessary to control the hybrid plant. The framework presented in this paper appears to be easily adaptable for control of hybrid systems. The presentation of the corresponding details are deferred to another paper, preliminary results can be found in [36].

3. **Inherent differences with respect to RW theory** Notice that not every instance of the proposed control problem can be solved using RW theory. In fact, game theory [19] can also be used to solve the control problem, even for cases when RW theory cannot be applied. Hence, it makes sense to formalize the proposed control problem separately.

**Related work** To the best of our knowledge, the presented results are new. Many of the
challenges regarding application of RW theory, including the need for explicit modeling of inputs and outputs, were already mentioned in [3, 12, 2, 9, 14, 11, 31]. With respect to [3], we allow unobservable internal events and disturbances. In addition, our way of modeling inputs and outputs explicitly is closer to classical control [22] and it is perhaps more intuitive for the problem at hand. However, [3] also addresses communication delays, which topic is absent from this paper. With respect to [30, 18, 8, 24], the main difference is that we allow unobservable internal events, disturbances and uncontrollable inputs. Furthermore, our control problem is not based on enabling/disabling events. The control problem of this paper is completely different from [12]. In [21] input/output discrete-event systems were investigated, but in contrast to this paper, the inputs were viewed as external uncontrollable inputs, and the outputs and internal events were the events which could be controlled by the supervisor. In [33, 34] input/output discrete-event systems were introduced in order to facilitate hierarchical control design. More precisely, there the task of the controller is to ensure that the input/output behavior of the plant with respect to the operator and environment satisfies the specification. Then, the closed-loop system can be viewed as another plant, defined only on the alphabet of the environment and the operator. In contrast, in this paper we are not interested in hierarchical control. This leads to several subtle differences between our framework and that of [33, 34]. In particular, we do not have an operator component with operator input and output, we do not need to ensure the the closed-loop system does not starve the operator from getting a response, and we do not allow the generation of internal event to prevent the activation of the controller. In our model, the activation of the controller and the generation of internal events take place in parallel, in an asynchronous way. In addition, we use a completely different mathematical language to formalize the control problem. Extensions of RW theory where the supervisor forces controllable events was investigated in [14, 2, 11, 7, 17, 20]. However, in these papers partial observations and explicit input-output modeling were not considered, and the framework seems to be further from the physical reality of the systems we are interested in than the one of this paper. Contrary to [11, 20], non-blockingness is not relevant in our case, due to the specific problem formulation. The problem of extracting deterministic supervisors was addressed in [32], however there the explicit modeling of inputs and outputs and partial observations were not addressed. In addition, all the above cited papers use languages of finite strings for specification, while we allow infinite strings in the problem formulation. In this respect our framework resembles [38, 26], but the presence of inputs and outputs and the event generating property of the controller makes it different. Automata with inputs and outputs have appeared in the context of model matching problem [10, 4]. Model matching is related to, but different from the control problem of this paper. The main difference can briefly be summarized as follows; in model matching we seek to find a controller such that the behavior of the closed-loop system matches the behavior of a designated system. In contrast, the control problem of this paper is to find a controller such that the behavior of the closed-loop system exhibits certain properties. The properties which are considered need not be expressible as behavior of a system of the same type as the plant. Hence we believe that the two problems are conceptually different. Probably there are cases when model matching and the control problem studied in this paper can be transformed one to another. However, exploring all those cases goes beyond the intended scope of the paper.

Other aspects of input-output modeling and automata were discussed in [28]. Automata with inputs and outputs is a classical topic, see [13, 16], and [29]. Game-theoretic methods for control of discrete-event system were discussed in [25, 1], but the results there do not cover the control problem of this paper. In [6] related problems are addressed. Note that [6] concentrates more on engineering aspects. In contrast, we focus on the formal problem formulation and correctness of the solution. In addition, we deal with partial observations and multiple event occurring at the same time. The paper [25] addresses a different control problem where they follow a game-theoretic approach. In contrast, in
this paper we connect the solution of the control problem to RW frameworks.

Outline of the paper

In §3 we formally state the discrete-event control problem we are interested in. In §4 we show that without loss of generality, the formulated control problem can be restricted to the case when the plant has no external inputs and it is deterministic. In §5 we describe how to solve our control problem using RW theory. Finally, in §6 we illustrate the theory on a practical example. We collected the proofs of a number of technical results in §8.

2 Automata-theoretic preliminaries

The goal of this section is to present an overview of the necessary background on automata theory. In Subsection 2.1 we review notion and terminology from formal language theory. In Subsection 2.2 we recall the definition of Moore-automata and related concepts. In Subsection 2.3 we review the classical concept of monoid, automata on monoids and rational subsets of monoids. In Subsection 2.4 we will use these notions to define the concept of sequential input-output relations, quasi-sequential deterministic transducer and quasi-recognizability. Finally, in Subsection 2.5 we present the relationship between the new concept of quasi-recognizability and the classical notions of rationality and recognizability of relations. The material of Subsection 2.4 and 2.5 are new, to the best of our knowledge. In contrast, the material of Subsection 2.1, 2.2, 2.3 can be found in the literature.

2.1 General notation

Most of the time, we will use the standard notation and terminology from automata theory [13, 16]. Let \( \Sigma \) be a finite set, referred to as the alphabet. \( \Sigma^* \) denotes the set of finite strings (words) of elements of \( \Sigma \), i.e. an element of \( \Sigma^* \) is a sequence \( w = a_1 a_2 \cdots a_k \), where \( a_1, a_2, \ldots, a_k \in \Sigma \), and \( k \geq 0 \); \( k \) is the length of \( w \) and it is denoted by \( |w| \). If \( k = 0 \), then \( w \) is the empty word, denoted by \( \epsilon \). The concatenation of two words \( v \) and \( w \) is denoted by \( vw \). An infinite (\( \omega \)-) word over \( \Sigma \) is an infinite sequence \( w = a_1 a_2 \cdots a_k \cdots \) with \( a_i \in \Sigma \), \( i \in \mathbb{N} \). The set of infinite words is denoted by \( \Sigma^\omega \).

A language over \( \Sigma \) is a set of finite strings (words) over \( \Sigma \). For any (in)finite word \( w \), and for any \( i \in \mathbb{N} \) (in case \( w \) is finite word, for any \( i \in \mathbb{N} \) such that \( i \leq |w| \)), \( w_{1:i} \) denotes the finite word formed by the first \( i \) letters of \( w \), i.e. \( w_{1:i} = a_1 a_2 \cdots a_i \). If \( i = 0 \), then \( w_{1:i} \) is the empty word \( \epsilon \).

For any word \( w \in \Sigma^* \cup \Sigma^\omega \), a finite word \( p \in \Sigma^* \) is a prefix of \( w \), if there exists an index \( i \in \mathbb{N} \), such that \( w_{1:i} = p \). If \( K \subseteq \Sigma^* \), then \( \text{lim}(K) \subseteq \Sigma^\omega \) is the set of all infinite words, infinitely many prefixes of which belong to \( K \), i.e.

\[
\text{lim}(K) = \{ w \in \Sigma^\omega \mid \exists \{ k_i \in \mathbb{N} \}_{i \in \mathbb{N}} : \text{such that } \forall i \in \mathbb{N} : \\
(k_{i+1} > k_i) \text{ and } \forall i \in \mathbb{N} : w_{1:k_i} \in K \}
\]

If \( L \subseteq \Sigma^* \cup \Sigma^\omega \), then the prefix closure of \( L \) is denoted by \( \overline{L} \) and is defined by \( \overline{L} = \{ p \in \Sigma^* \mid \exists v \in L : p \text{ is a prefix of } v \} \); \( L \) is called prefix closed, if \( \overline{L} = L \).

A map \( \theta : X^* \rightarrow Y^* \), where \( X \) and \( Y \) are finite alphabets, is called a morphism, if \( \theta \) preserves the empty sequence and concatenation, i.e. \( \theta(\epsilon) = \epsilon \) and \( \theta(wv) = \theta(w)\theta(v) \).
For a finite set $S$, we denote by $2^S$ the set of subsets of $S$. The cardinality of a finite set $S$ is denoted by $|S|$.

### 2.2 Moore-automata

Below we will review the notion of Moore-automata. Note that Moore-automata will play the role of controllers in our setting. Recall from [13, 16] that a Moore-automaton is a tuple $A = (Q, I, Y, \delta, \lambda, q_0)$ where $Q$ is the finite state-space of $A$, $I$ is the input alphabet of $A$, $Y$ is the output alphabet of $A$, $\delta : Q \times I \rightarrow Q$ is the state-transition map of $A$, $\lambda : Q \rightarrow Y$ is the readout map of $A$, and $q_0 \in Q$ is the initial state of $A$. The Moore-automaton $A$ is a realization of a map $\phi : I^* \rightarrow Y$, if for all $w = u_1u_2 \cdots u_k \in I^*$, $k \geq 0$ and $u_1, u_2, \ldots, u_k \in I$, $\phi(w) = \lambda(q_k)$ where $q_i = \delta(q_{i-1}, u_i)$ for all $i = 1, 2, \ldots, k$. The map $\phi$ is realizable by a Moore-automaton, if there exists a Moore-automaton which is a realization of $\phi$.

### 2.3 Monoid, automata, rational sets

The goal of this section is to recall the notion of monoid, rational and recognizable subsets of a monoid, and automata on monoids. These concepts will then be used to define the concept of sequential input-output relations and their automaton representations. The latter concepts are used for modeling the behavior of the plant.

Recall from [5, 13] that a monoid $M$ is a (not necessarily finite) semi-group with a unit element which is denoted by $1_M$, or simply 1, if $M$ is clear from the context. That is, there exists a multiplication operation, denoted by $\cdot$. For $m_1, m_2 \in M$, the product of $m_1$ with $m_2$ (in this order!) will be denoted by $m_1m_2$. The multiplication is associative, i.e. $m_1(m_2m_3) = (m_1m_2)m_3$ for all $m_1, m_2, m_3 \in M$. In addition, for the unit element 1 it holds that $1m = m1 = m$ for all $m \in M$. The set of all finite strings $\Sigma^*$ over the finite alphabet $\Sigma$ forms a monoid, if we take the concatenation as multiplication and the empty word $\epsilon$ as the unit element. The monoid $\Sigma^*$ is also referred to as the free monoid. If $M_1$ and $M_2$ are two monoids, then the cartesian product $M_1 \times M_2$ has a monoid structure where multiplication is taken as the component-wise multiplication, i.e. for all $(m_1, m_2), (\hat{m}_1, \hat{m}_2) \in M_1 \times M_2$, $(m_1, m_2)(\hat{m}_1, \hat{m}_2) = (m_1\hat{m}_1, m_2\hat{m}_2)$, and the unit element is $(1_{M_1}, 1_{M_2})$. Important examples of cartesian products of monoids are products of the form $X^* \times Y^*$, where $X$ and $Y$ are finite alphabets. According to the definition above, $X^* \times Y^*$ is a monoid, with identity element $(\epsilon, \epsilon)$, and with multiplication operation defined by $(s_1, s_2)(v_1, v_2) = (s_1v_1, s_2v_2)$.

Below we will recall from [5, 13] the notion of a finite-state automaton on monoids.

**Definition 1 (Automaton on monoid [5, 13]):** A finite-state automaton on a monoid $M$, abbreviated as DFA, is a tuple $T = (Q, M, E, F, q_0)$ where

- $Q$ is the finite set of states
- $M$ is the monoid of inputs
- $E \subseteq Q \times M \times Q$ is the state-transition relation. It is assumed that the set $E$ is finite.
- $F \subseteq Q$ is the finite set of accepting states
- $q_0 \in Q$ is the initial state

7 Automata-theoretic preliminaries
The finite-state automaton $T$ is called deterministic, if the relation $E$ is a partial map of the form $E : Q \times M \to q$.

Definition 2 (Accepting run, [5, 13]). An element $m \in M$ is accepted by $T$ if there exist elements $m_i \in M_i$ and states $q_i \in Q$, $i = 1, 2, \ldots, k$ for some $k \geq 0$ such that $(q_i, m_i, q_{i+1}) \in E$ for $i = 0, 1, \ldots, k - 1$, $q_k \in F$ and $m = m_1m_2 \cdots m_k$.

The definition of a subset of $M$ accepted by the DFA $T$ is completely analogous to the definition of languages accepted by an automaton.

Definition 3 (Sets recognized by DFA , [5, 13]). The set $L \subseteq M$ is recognized by $T$, and it is denoted by $L(T)$, if $L$ consists of precisely those elements $m \in M$ which are accepted by $T$.

Definition 4 (Rationality). A subset $L \subseteq M$ is called rational, if there exists a finite-state automaton $T$ on $M$ such that $L$ is recognized by $T$.

In other words, rational subsets of $M$ are precisely those subsets which can be described by (possibly non-deterministic) finite state automata. Rational subsets of monoids have been studied since the 1960’s [5, 13, 27] and the references therein. One of the motivation for studying these subsets is to be able to define the concept of rational relations defined on words over finite alphabets. Rational relations can be described by finite-state machines of some sort (finite-state transducer). Input-output maps of classical sequential machines represent the simplest such class [5, 13]. The topic of rational relations and their subclasses has been a field of intensive research with a lots of open problems, [5, 27, 15].

2.4 Sequential input-output relations

The goal of this section is to define the notion of sequential input-output relations. Sequential input-output relations will be used to model the input-output behavior of non-deterministic discrete-event plants.

Definition 5 (Sequential input-output relations). A multi-valued map $R : \Sigma^* \to 2^{X^* \times Y^*}$ is called a sequential input-output relation, if the following conditions are satisfied

1. $R(\epsilon) = (\epsilon, \epsilon)$, and for all $s \in \Sigma^*$, $R(s)$ is a finite and non-empty set.
2. For all $s \in \Sigma^*$, if $(\alpha, \hat{\alpha}) \in R(s)$, with $\alpha \in X^*$ and $\hat{\alpha} \in Y^*$, then the length of $s$ and $\alpha$ are the same, i.e. $|s| = |\alpha|$.
3. $R$ is prefix preserving, i.e. for each word $s \in \Sigma^*$, for each letter $a \in \Sigma$, and for each pair of words $(x, y) \in R(sa)$, there exist a letter $x \in X$ and words $y \in Y^*$, $\hat{x} \in X^*$, $\hat{y} \in Y^*$ such that $x = \hat{x}x$, $y = \hat{y}y$ and $(\hat{x}, \hat{y}) \in R(s)$.
4. $R$ is non-blocking, i.e. for each word $s \in \Sigma^*$, for each letter $a \in \Sigma$, and for each word $x \in X^*$, $y \in Y^*$ such that $(x, y) \in R(s)$, there exists a letter $x \in X$ and a word $y \in Y^*$, such that $(xx, yy) \in R(sa)$.

In this paper we are mainly be interested in sequential input-output relations which are quasi-recognizable, i.e. sequential input-output relations whose graph is a rational subset of the monoid $M = \Sigma^* \times X^* \times Y^*$ and which can be recognized which can be recognized by a QSTD

Definition 6 (QSTD ). A DFA $T = (Q, (\Sigma^* \times X^* \times Y^*), E, F, q_0)$ defined over the monoid $M = \Sigma^* \times X^* \times Y^*$ is called a quasi-sequential deterministic transducer (QSTD for short), if
1. \( F = Q \), i.e. all states are accepting.

2. the state-transition relation is a partial map \( E : Q \times \Sigma \times X \times Y^* \rightarrow Q \). That is, the state-transitions are deterministic and are labeled by of letters from \( \Sigma \) and \( X \) and by sequences from \( Y^* \). Notice that \( (\Sigma \times X \times Y^*) \) can naturally be identified with a subset of \( (\Sigma^* \times X^* \times Y^*) \).

3. For each \( q \in Q \) and \( a \in \Sigma \) there exist a letter \( x \in X \) and a word \( y \in Y^* \) such that \( E(q,u,x,y) \) is defined.

Definition 7 (Quasi-recognizable sequential input-output maps). The sequential input-output relation \( R : \Sigma^* \rightarrow 2^{X^* \times Y^*} \) is called quasi-recognizable, if the corresponding graph \( \text{graph} \ R \) of \( R \), defined as

\[
\text{graph} \ R = \{(u,x,y) \in \Sigma^* \times X^* \times Y^* \mid (x,y) \in R(u)\}
\]

has the following property. If \( \text{graph} \ R \) is viewed as subset of the monoid \( M = \Sigma^* \times X^* \times Y^* \), then \( \text{graph} \ R \) is recognized by a quasi-sequential deterministic transducer.

2.5 Relationship between quasi-recognizability, recognizability and rationality

As it was mentioned above, several classes of relations on words which can be described by finite-state automata were already investigated in the literature. The class of quasi-recognizable relations represents yet another such class of relations. Hence, one may wonder about the relationship between quasi-recognizability and the classical notion of recognizability and rationality of relations. In this section we will elaborate on this relationship. The material of this section is not necessary for understanding the rest of the paper. First, recall from [5, 13] the notion of a recognizable subset of a monoid \( M \).

Definition 8 (Recognizability of a subset, [5, 13]). A subset \( L \subseteq M \) of a monoid \( M \) is said to be recognizable, if there exists a finite semi-group \( S \), a subset \( H \subseteq S \) and a monoid morphism \( \phi : M \rightarrow S \), such that \( \phi^{-1}(H) = L \).

It is well-known [5, 13] that recognizable sets are also rational. In fact, recognizable sets are rational sets which can be recognized by a deterministic DFA. The converse is not true in general. Notice that if \( R \) is a sequential input-output relation as in Definition 5, then the graph \( \text{graph} \ R \) of \( R \), defined in (1), can be viewed as a subset of the monoid \( M = \Sigma^* \times X^* \times Y^* \) and hence we can speak of recognizable sequential input-output relations, i.e. \( R \) is said to be recognizable, if the graph of \( R \) is recognizable as a subset of \( M \). Similarly, we will say that \( R \) is rational, if the graph of \( R \) is rational if viewed as a subset of \( M \). Then, we can establish the following relationship.

Theorem 1 (Recognizability, rationality and quasi-recognizability). If the sequential input-output relation \( R : \Sigma^* \rightarrow 2^{X^* \times Y^*} \) is recognizable, then it is quasi-recognizable. If \( R \) is quasi-recognizable, then it is rational.

The proof of the theorem can be found in §8.

3 Problem formulation

The goal of this section is to formulate the control problem studied in this paper. The systems (plants) of interest have five types of signals;
1. *control inputs* \((U)\),
2. *external inputs* \((V)\),
3. *unobservable disturbances* \((D)\),
4. *observable outputs* \((O)\), and
5. *internal events* from \((E_i)\).

The sets \(U\), \(V\), \(D\), \(O\) and \(E_i\) are all finite. Only the elements of \(U\), \(V\) and \(D\) are capable of changing the dynamics of the system. The appearance of symbols from \(O\) and \(E_i\) *may indicate* an occurrence of a state-transition, but is *does not trigger a state-transition itself*. Typical elements of \(U\) could be commands to switch an engine on/off, typical elements of \(V\) are events such as a button pressed, error message has arrived, etc. Typical elements of \(D\) are error-conditions. Typical elements of \(O\) are sensor data, typical elements of \(E_i\) are invisible events which are needed for the specification of the control objectives.

Informally, the uncontrolled plant is a discrete-time control system. At each step, the plant reads a control input symbol from \(U\), an external input symbol from \(V\) and a disturbance symbol from \(D\). The plant then changes its internal state to a new one, which depends on the control input symbol, external input symbol and disturbance symbol which were read. In addition, while executing the state transition, it also generates an output symbol from \(O\) and several (possibly none) internal event symbols from \(E_i\). After completing the state transition, the plant starts the cycle again. Note that the description above fits the time-driven paradigm better than the event-driven one. In the event-driven paradigm the plant should operate as an event generator, generating one event at a time. Here, the plant behaves more like a discrete-time system, simultaneously reading certain symbols and generating others. Moreover, the reading of the control and external inputs and disturbances takes place simultaneously. We believe that the above paradigm arises naturally in a number of application, for example, when looking at time-driven control of systems with discrete-valued inputs and outputs.

We will use the notion of *sequential input-output relations* to formalize the input-output behavior of the plant.

\[
\text{Controller} \, C \quad \text{Plant} \, P
\]

\[
U^* \quad V^* \quad D^* \quad O^* \quad E_i^* \quad \text{Plant} \, P \quad \text{Controller} \, C
\]

**Figure 1: Control architecture**

**Definition 9 (Plant behavior).** The *plant* is a sequential input-output relation of the form

\[
P : (U \times V \times D)^* \rightarrow 2^{O^* \times E_i^*}.
\]

In other words, the plant of Definition 9 is a map which maps sequences of triples from \((U \times V \times D)\) to pairs of sequences of output symbols and internal symbols respectively. Furthermore, from Definition 5 of sequential input-output relations it follows that if the length of the control input, disturbance and external input sequence increases, so does the length of the sequence of the observable outputs produced by the plant modeled by \(P\). In fact, the length of the observable output sequence is the same as the length of the input sequence. That is, the reading in of the control, external inputs, disturbance
and the writing of the outputs are synchronized. In contrast, the production of internal events are totally asynchronous with respect to the reading of the inputs and production of outputs. This combination of synchronous/asynchronous behavior is motivated by the control architecture we are interested in. The precise motivation should become clear after the control architecture is described.

The task of a would-be controller is to generate control inputs based on past outputs and external inputs, such that the control objectives are met.

**Definition 10 (Sequential controllers).** A sequential controller is a map of the form \( C : (V \times O)^* \to U \) such that \( C \) is realizable by a Moore-automaton with input alphabet \( V \times O \) and output alphabet \( U \).

That is, a sequential controller is simply a dynamical system, which reads the external inputs and the output of the plant, updates its internal state and generates a control input.

The structure of the controller explains the peculiar combination of requirements regarding synchrony/asynchrony between inputs, outputs, disturbances and internal events. The elements of \( O \) an \( V \) represent the information available to the controller at activation times, and hence their number is tied to the number of times the controller was activated. In contrast, the symbols from \( E_i \) are never used for control, they appear only in the specification and the plant model. Hence, it is not natural to require any synchronization between the reading in of control inputs/external inputs/disturbances and generation of outputs on the one hand, and generation of internal events on the other hand. In contrast, for control synthesis, one has to assume synchronization of reading in the inputs and generating outputs. If the plant does not satisfy this condition, one has to replace the sets of inputs and outputs with subsets of the corresponding spaces, in order to adequately describe the reaction of the plant to the control actions. Next, we define the behavior of the closed-loop system.

**Definition 11 (Feedback).** Let \( P : (U \times V \times D)^* \to 2^{O^* \times E_i^*} \) be the plant and \( C : (V \times O)^* \to U \) be a sequential controller. The behavior of the feedback interconnection of \( P \) with \( C \) is the map \( B(P/C) : (V \times D)^* \to 2^{E_i^*} \), defined as follows. Fix a sequence of external inputs and disturbances \( s = (v_1, d_1)(v_2, d_2) \cdots (v_k, d_k) \in (V \times D)^* \), \( (v_1, d_1), (v_2, d_2), \ldots, (v_k, d_k) \in (V \times D) \), \( k \geq 0 \). If \( k = 0 \), i.e. \( s = \epsilon \) is the empty sequence, then let \( B(P/C)(\epsilon) = \epsilon \). If \( k > 0 \), then let \( B(P/C)(s) \) consist of precisely those words \( \hat{o} \in E_i^* \) which are of the form

\[
\hat{o} = \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k \in E_i^* \tag{2}
\]

where \( \hat{o}_i \in E_i^* \), \( i = 1, \ldots, k \), and for which there exist control inputs \( u_i \in U \), and outputs \( o_i \in O \), \( i = 1, 2, \ldots, k \) such that

\[
(o_1 o_2 \cdots o_i, \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k) \in P((u_1, v_1, d_1)(u_2, v_2, d_2) \cdots (u_i, v_i, d_i))
\]

\[
u_i = C((v_1, o_1)(v_2, o_2) \cdots (v_{i-1}, o_{i-1})) \tag{3}
\]

Here, for \( i = 1 \) \((d_1, o_1)(d_2, o_2) \cdots (d_{i-1}, o_{i-1})\) is identified with the empty sequence \( \epsilon \) and hence \( u_1 = \phi(\epsilon) \).

Intuitively, the external inputs from \( V \) and disturbances from \( D \) are taken as inputs of the closed-loop system, and the internal events as outputs. Next, we define the notion of the language of the closed-loop system. The latter is used to define the control requirements which the closed-loop system should meet. Recall from §2.1 that \( w_{1:i} \) denotes the prefix of a (possibly infinite) word \( w \), formed by the first \( i \) letters of \( w \).

**Definition 12 (Language of the closed-loop system).** Define the closed-loop language \( L(P/C) \subseteq E_i^* \cup E_i^o \) of the interconnection of \( P \) and \( C \) as follows.
1. \( \hat{o} \in E^* \) belongs to \( L(\mathbf{P}/\mathbf{C}) \) if there exists an infinite word \( s \in (V \times D)^\omega \) such that starting from some index \( N \in \mathbb{N} \), for each index \( i \geq N \), \( \hat{o} \in B(\mathbf{P}/\mathbf{C})(s_{1:i}) \).

2. \( \hat{o} \in E^\omega \) belongs to \( L(\mathbf{P}/\mathbf{C}) \) if there exists an infinite word \( s \in (V \times D)^\omega \) and an infinite sequence of indices \( k_0 \leq k_1 \leq \cdots \leq k_i \leq \cdots \), \( k_i \in \mathbb{N} \), \( i \in \mathbb{N} \), such that \( \sup_{i \in \mathbb{N}} k_i = \infty \) and for each \( i \in \mathbb{N} \), \( \hat{o}_{1:k_i} \in B(\mathbf{P}/\mathbf{C})(s_{1:i}) \).

The language \( L(\mathbf{P}/\mathbf{C}) \) is the set of all (in)finite sequences of internal events generated by the closed-loop system. Finite sequences of \( L(\mathbf{P}/\mathbf{C}) \) describe the situation when the closed-loop system ceases to produce internal events after a finite number of steps. Infinite strings represent the situation when the closed-loop system never stops producing internal events. The definition above simply says that the closed-loop system consists of the plant receiving inputs from the controller and the controller reacting to the outputs generated by the plant. Hence, only external inputs and disturbances can alter the behavior of the closed-loop system, and therefore they were taken as its inputs. The internal events are taken as the outputs of the closed-loop system. Next we formulate the control problem studied in this paper.

\textbf{Problem 1 (Discrete-event control problem).} For a specified plant \( \mathbf{P} \), and for a specification language \( K \subseteq E^*_i \cup E^\omega_i \) modeling the control requirements, find a sequential controller \( \mathbf{C} \) such that \( L(\mathbf{P}/\mathbf{C}) \subseteq K \).

That is, the internal events generated by the closed-loop system must belong to the specification language \( K \) containing both finite and infinite words.

\textbf{Remark 1 (Restriction on \( U, V, D \) and \( O \)).} Notice that the elements of \( U, V, D \) and \( O \) are not explicitly included in the specification language \( K \). This was done in order to simplify notation and to emphasize the distinction between what the controller can observe/do and what it should achieve. However, conditions on control inputs, external inputs, disturbances and observable outputs can be incorporated into our framework as follows. Modify the plant model by adding the current control input, and/or external inputs, and/or disturbances and/or observable output as new components of the currently generated internal event, and then adapt the specification language \( K \) accordingly.

\textbf{Remark 2 (Role of internal events in the specification of control requirements).} As it was noted above, the role of the internal events is to provide a vehicle for expressing control requirements. In particular, internal events allow us to express control objectives such as avoiding bad states, or reaching good states, without explicitly using the state-space representation of the plant model.

This approach has the advantage that it explicitly distinguishes between the behavior of the system (sequences of internal events) and its state-space representation. While the former is intrinsic in some sense, the latter is a matter of choice.

This distinction may also yield practical advantage when applying the results of the paper to automatic synthesis of control software. Notice that the control requirements typically involve those state components which have some physical interpretation and which are intrinsically part of the system. However, the actual state-space model of the system often contains state-space components presence of which is a matter of abstraction level and modeling accuracy. These state components tend to be modified quite often during the development phase. For example, if the state-space model of the plant is obtained by discretizing a continuous model, then the state-space changes every time when the accuracy of discretization is changed. In contrast, the control requirements often changes less frequently. For example, replacing one type engine in a printer by another type may influence the plant model, but it will hardly change the description of the correct functionality of the printer, which is exactly our control requirement. By using internal
events to specify properties of the desired behavior, we avoid the need to change the control requirements every time the state-space is changed. Finally, we would like to remark that there also practical examples where it is the control requirements, rather than the plant which change frequently.

The rest of the paper will be devoted to the solution of Problem 1. In order to obtain a solution using Ramadge-Wonham theory, in the rest of the paper we will assume the following.

**Assumption 1 (Finite-state assumption).** In the sequel, we will assume the following.

- **Plant** The sequential input-output relation $P$ is quasi-recognizable.
- **Requirements** $K = K_{safe} \cup \lim(K_{safe})$ where $K_{safe} \subseteq E_i^*$ is regular and prefix-closed.

The assumption above on $K$ essentially says that the control requirement should be a safety specification, i.e. the system should always produce words belonging to $K_{safe}$. That is, no liveness is really involved. Exactly this restriction is the one which allows applying classical RW theory to achieve control requirements specified by infinite strings. We can formulate the main contribution of this paper as follows.

**Theorem 2 (Main result).** If Assumption 1 holds, then a controller solving Problem 1 can be computed using classical Ramadge-Wonham supervisory control synthesis with partial observations.

The rest of the paper is devoted to proving the main result.

## 4 Elimination of external inputs and non-determinism

The control problem formulated above might reflect the setting of real-life control problems quite precisely, but it is (at least notationally) rather complicated. In order to make the analysis of the control problem above easier, we formulate a simpler control problem and we will show how to transform the original control problem to this simpler one. The core idea is that we can eliminate the external inputs by viewing occurrence of each external input as an occurrence of a suitable disturbance and generation of a suitable output. In addition, we can eliminate non-determinism by viewing each non-deterministic choice as a result of occurrence of an invisible disturbance symbol. That is, we extend the space of disturbances with external input symbols and symbols describing the non-deterministic choices. Similarly, we extend the space of outputs with copies of external inputs. If the plant is in a certain state, then the arrival of an external input symbol is simulated as occurrence of the symbol’s copy on the disturbance channel and generation of a copy of this symbol on the output channel. Non-deterministic generation of an output or internal event is simulated by an occurrence of an invisible disturbance.

In order to formalize the procedure described above, we need the notation of sequential machine and maps realized by sequential machines.

**Definition 13 (Sequential machines).** A sequential machine is a tuple

$$M = (Q, \Sigma, X, Y, \delta, \lambda, q_0)$$

where
• $X, Y, \Sigma$ are finite alphabets,
• $Q$ is a finite set of states,
• $\delta : Q \times \Sigma \rightarrow Q$ is the state-transition map,
• $\lambda : Q \times \Sigma \rightarrow X \times Y^*$ is the readout map,
• $q_0 \in Q$ is the initial state.

We can extend the map $\delta$ to act on sequences $\Sigma^*$; $\delta(q, \epsilon) = q$ and $\delta(q, sa) = \delta(\delta(q, s), a)$ for word $s \in \Sigma^*$ and letter $a \in \Sigma$. Note that any sequential machine $M$ can be viewed as a quasi-sequential deterministic transducer $T_M = (Q, M, E, F, q_0)$ where $E(q, (u, x, y))$ is defined if $\lambda(q, u) = (x, y)$ and $E(q, (u, x, y)) = \delta(q, u)$. The map realized by a sequential machine $M$ is defined as follows.

**Definition 14 (Sequential maps).** A deterministic (single-valued) sequential input-output relation will be called a sequential map. A sequential map $R : \Sigma^* \rightarrow \mathcal{O}^* \times \mathcal{E}^*_i$ is said to be realized by a sequential machine $M = (Q, \Sigma, O, E_i, \delta, \lambda, q_0)$, if $R(\epsilon) = (\epsilon, \epsilon)$, and for all words $s \in \Sigma^*$ and letters $a \in \Sigma$,

$$R(sa) = R(s) \lambda(\delta(q_0, s), a)$$

Here we used the component-wise concatenation operation on $\mathcal{O}^* \times \mathcal{E}^*_i$.

Notice that if $R$ is realized by a sequential machine, then $R$ is necessarily quasi-recognizable. In fact, if $R$ is realized by the sequential machine $M$ if and only if $R$ is realized by the quasi-sequential deterministic transducer $T_M$ corresponding to $M$.

Below we will show that without loss of generality we can always restrict attention to the so called deterministic simple plants.

**Definition 15 (Deterministic simple plant).** A deterministic simple plant is a plant without external inputs, modeled as a sequential input-output map of the form $P : (U \times D)^* \rightarrow \mathcal{O}^* \times \mathcal{E}^*_i$ such that $P$ can be realized by a sequential machine. Here $U$ is the set of control inputs, $D$ is the set of invisible disturbances, $O$ is the set of outputs and $E_i$ is the set of internal events.

In other words, a deterministic simple plant has no external inputs, and for any sequence of control inputs and disturbances it generates a unique sequence of outputs and internal events. Moreover, it admits a finite-state deterministic state-space realization, such that the output and internal events generated at each step depend only on the current state.

Notice that a deterministic simple plant can always be viewed as a plant whose set $V$ of external inputs is a singleton set. Since external inputs are absent, the controllers for a deterministic simple plant $P : (U \times D)^* \rightarrow \mathcal{O}^* \times \mathcal{E}^*_i$ are maps of the form $C : \mathcal{O}^* \rightarrow U$ such that $C$ is realizable by a Moore-automaton. The definition of the interconnection $B(P/C)$ and of the closed-loop language $L(B(P/C))$ follow from the general definitions.

For the sake of completeness, we repeat them here. That is, $B(P/C) : D^* \rightarrow \mathcal{E}^*_i$ such that for any $s = d_1 d_2 \cdots d_k \in D^*$, $d_1, d_2, \ldots, d_k \in D$, $B(P/C)(s) = \hat{o}$, if there exist letters $u_i \in U$, $o_i \in O$ and words $\hat{o}_i \in \mathcal{E}^*_i$ for $i = 1, 2, \ldots, k$ such that

$$\hat{o} = \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k$$

and

$$\begin{align*}
(o_1 o_2 \cdots o_k, \hat{o}) &= R((u_1, d_1)(u_2, d_2) \cdots (u_k, d_k)) \\
u_{i+1} &= C(o_1 o_2 \cdots o_i), i = 0, 2, \ldots, k
\end{align*}$$
The language $L(B(\mathbf{P}/\mathbf{C}))$ is the set of all finite or infinite words $\hat{o} \in E^*_i \cup E^*_o$ for which there exists an infinite word $s \in D^*$ such that either $\hat{o} \in E^*_i$ and there exists an index $N \in \mathbb{N}$ such that $\hat{o} = B(\mathbf{P}/\mathbf{C})(s_{1,i})$ for all $i \geq N$, or $\hat{o} \in E^*_o$ and there exists an increasing sequence $k_i \in \mathbb{N}$, $i \in \mathbb{N}$, $k_i \leq k_{i+1}$, such that $\sup_{i \in \mathbb{N}} k_i = +\infty$ and $\hat{o}_{1,k_i} = B(\mathbf{P}/\mathbf{C})(s_{1,i})$ for all $i \in \mathbb{N}$. The control problem of Problem 1 can be specialized to deterministic simple plants as follows.

Problem 2 (Simplified control problem). For a specified plant modeled as $\mathbf{P} : (U \times D)^* \rightarrow O^* \times E^*_i$, and for the control requirements modeled as a language $K \subseteq E^*_i \cup E^*_o$, find a controller $\mathbf{C} : O^* \rightarrow U$ such that $L(B(\mathbf{P}/\mathbf{C})) \subseteq K$.

The control architecture of the control problem above is depicted on Fig. 1. As we mentioned above, Problem 1 and Problem 2 are equivalent. More precisely, one can associate with each plant from Definition 9 a deterministic simple plant, such that a controllers solves Problem 1 for the original plant, if and only if it solves Problem 2 for the associated deterministic simple plant. For the formal statement, we need the following notation.

Notation 1 (Projection). Let $A$ and $B$ be two finite sets and define the projection operator $\Pi_X : (A \times B)^* \rightarrow X$, $X \in \{A, B\}$ as follows: $\Pi_A(\epsilon) = \epsilon$, $\Pi_B(\epsilon) = \epsilon$ and for $s = (a_1, b_1)(a_2, b_2) \cdots (a_k, b_k) \in (A \times B)^*$, $a_i \in A, b_i \in B$, $i = 1, 2, \ldots, k, k > 0$.

\[ \Pi_A(s) = a_1a_2 \cdots a_k \quad \text{and} \quad \Pi_B(s) = b_1b_2 \cdots b_k \]

That is, $\Pi_A$ and (resp. $\Pi_B$) project any word over $s \in (A \times B)^*$ to a word over $A$ (resp. $B$) consisting of the $A$-valued (resp. $B$-valued) components of the letters $s$; the relative order of the letters preserved.

Theorem 3 (Equivalence of control concepts). Consider a plant modeled by a quasi-recognizable sequential input-output relation $\mathbf{P} : (U \times D \times V) \rightarrow 2^{O^* \times E^*_i}$. Then the following holds.

- There exists a set $D_{ax}$ and a deterministic simple plant $\mathbf{S}(\mathbf{P}) : (U \times \hat{D}) \rightarrow \hat{O}^* \times E^*_i$ such that the following holds. The sets $U$, $\hat{O} = V \times O$ and $E_i$ are the sets of control inputs, outputs and internal events of $\mathbf{S}(\mathbf{P})$, and $\hat{D} = V \times D \times D_{ax}$ is the set of disturbances of $\mathbf{S}(\mathbf{P})$. Moreover, for all $s \in (U \times \hat{D})^*$, if $\mathbf{S}(\mathbf{P})(s) = (\tilde{o}, \tilde{d})$, then $(\Pi_O(\tilde{o}), \tilde{d}) \in \mathbf{P}(\Pi_U \times V \times D(s))$ and $\Pi_V(\tilde{o}) = \Pi_V(\tilde{d})$. Conversely, for any $\tilde{v} \in (U \times V \times D)^*$ and for any $(x, \tilde{o}) \in \mathbf{P}(\tilde{v})$, there exists $s \in (U \times \hat{D})^*$ such that $|s| = |\tilde{v}|$, $\Pi_U \times V \times D(s) = \tilde{v}$, $(\tilde{o}, \tilde{d}) = \mathbf{S}(\mathbf{P})(s)$ and $\tilde{o} = \Pi_D(x)$. $\Pi_V(\tilde{o}) = \Pi_V(\tilde{d})$. Furthermore, a sequential machine realizing $\mathbf{S}(\mathbf{P})$ can be computed from any quasi-sequential deterministic transducer recognizing $\mathbf{P}$.

- For any map $\mathbf{C} : (V \times O)^* \rightarrow U$ realizable by a Moore-automaton, $\mathbf{C}$ can be viewed as a controller both for $\mathbf{P}$ and for the deterministic simple plant $\mathbf{S}(\mathbf{P})$ and in both cases the closed-loop languages coincide, i.e. $L(B(\mathbf{P}/\mathbf{C})) = L(\mathbf{S}(\mathbf{P})/\mathbf{C})$. 

15 Elimination of external inputs and non-determinism

Figure 2: Control architecture
The controller $C$ is a solution of Problem 1 for the plant $C$ and requirements $K \subseteq E_x^c \cup E_y^c$ if and only if $C$ is a solution of Problem 2 for the deterministic simple plant $S(P)$ and requirements $K$.

The proof of Theorem 12 can be found in §8. The intuition behind the construction of $S(P)$ is the following. We augment the set of disturbances with copies of external inputs. In addition, we augment the set of disturbances with elements of the set $D_{ax}$ encoding the non-deterministic choice of outputs and internal events. The latter set is finite, due to the quasi-recognizability of $P$. We augment the output space with copies of external events. If $S(P)$ receives a sequence consisting of control inputs and augmented disturbance symbols, then the corresponding output and internal events are obtained as follows. Each augmented disturbance symbol has a component encoding a disturbance symbol from the original set $D$, a component encoding an external input, and a component from $D_{ax}$ encoding a non-deterministic choice. We extract the disturbance symbols and the external input symbols from the augmented disturbances and together with the control inputs we feed them into the original plant $P$. As a result, we obtain several sequences of outputs and internal events. We use the $D_{ax}$-valued components of the augmented disturbances to choose a particular sequence of outputs and external events from the set generated by the original plant $P$. Finally, we copy the external input components of the augmented disturbances to the output channel of $S$.

**Remark 3 (Role of recognizability).** Notice that we can always replace external inputs by augmenting disturbances and outputs in the manner described above. However, in order to get rid of non-determinism, we need to assume the plant input-output behavior of the plant is quasi-recognizable. The latter is necessary in order to ensure that the number of non-deterministic choices remains finite.

**Remark 4 (Interpretation of non-determinism).** The result above is a formalization of the modeling philosophy, according to which each instance of a non-deterministic choice is viewed as a occurrence of an invisible disturbance. Note that in many systems, especially the ones arising by abstraction from a hybrid systems, non-determinism represents lack of knowledge, rather than invisible disturbance.

The results of Theorem 3 imply that in order to solve Problem 1, it is enough to restrict attention to deterministic simple plants. **In the sequel, unless stated otherwise, the word plant will always refer to a deterministic simple plant.**

## 5 Solution through supervisory control

In this section we show that Problem 2 can be reduced to classical RW control problem as follows.

1. Transform an instance of Problem 2 to an instance of a RW control problem with partial observations.
2. Synthesize a supervisor for the RW control problem using the well-known tools and algorithms from [39].
3. Extract from the supervisor a sequential controller which solves the original problem instance.

Below we elaborate on each step above and show that they are computationally effective. We will tacitly use the notation of [39].
5.1 From Problem 2 to a RW problem

First, we define the plant language of the RW control problem corresponding to Problem 2.

Definition 16 (Plant-language). Assume that the plant is modeled by a sequential map \( P : (U \times D)^* \rightarrow O^* \times E_i^* \) realizable by a sequential machine. The plant language \( L_P \subseteq (U \cup D \cup O \cup E_i)^* \) associated with the plant \( P \) is defined as

\[
L_P = \{(w, d) \in (U \cup D \cup O \cup E_i)^* | w = (u_0 d_1 o_1 \hat{o}_1) (u_1 d_2 o_2 \hat{o}_2) \cdots (u_{k-1} d_k o_k \hat{o}_k) u_k | u_i \in U,
\]

\[
u_0 \in U, d_i \in D, o_i \in O, \hat{o}_i \in E_i^*,
\]

\[\{(a, b, c, d) \in (E_i)^4 | a \cdot \hat{a}_i = (v_0 d_1 o_1 \hat{o}_1) \cdots (v_{k-1} d_k o_k \hat{o}_k) v_k \}| \]

for \( i = 1, 2, \ldots, k, k \geq 0 \}

That is, the plant language \( L_P \) consists of strings, which are made up of groups of symbols, first element of which is the control input, the second one is the disturbances, the third one is the output and the fourth one is the sequence of internal events produced by the plant.

That is, if \( w \in L_P \), then for some \( k \geq 0 \), \( w \) can be decomposed as \( w = \) \( u_1 \cdot \) \( u_2 \cdot \) \( \ldots \cdot \) \( u_k \cdot \) \( u_k \) where \( u_i = u_{i-1} d_i o_i \hat{o}_i \), such that \( d_i \) is the disturbance at step \( i \), \( o_i \) is the observable output produced at step \( i \), \( \hat{o}_i \) is the sequence of internal events at step \( i \) and finally \( u_{k-1} \) is the control input received at step \( i \). Finally, \( u_k \) is the input received at step \( k + 1 \). The intuition behind \( L_P \) is that its words keep recognizable the basic cycle of reading input and producing output. Note that any other ordering of events within \( w_i \) could have been taken, as long as this order if fixed for all words of the plant language. However, the chosen ordering makes the application of Ramadge-Wonham theory easier. The following proposition follows from standard automata theory [13, 5].

Proposition 1 (Regularity of \( L_P \)). If \( P : (U \times D)^* \rightarrow O^* \times E_i^* \) is a deterministic simple plant then \( L_P \) is regular, and a deterministic automaton recognizing \( L_P \) can be computed from any sequential machine \( M \) realizing \( P \).

The proof of the proposition is presented in §8.

Furthermore, we will need the following maps.

Notation 2 (Erasing events in \( E_i \) and \( D \)). The projection \( \theta : (U \cup D \cup O \cup E_i)^* \rightarrow (U \cup O)^* \) deletes elements of \( E_i \) and \( D \); \( \theta \) is a morphism such that \( \theta(a) = a \) if \( a \in (U \cup O) \) and \( \theta(c) = c \) for all \( c \in D \cup E_i \).

Notation 3 (Erasing events not in \( E_i \)). The morphism \( \theta_e : (U \cup D \cup O \cup E_i)^* \rightarrow E_i^* \) erases all occurrences of letters not in \( E_i \), i.e. \( \theta_e(a) = a \) if \( a \in (U \cup D \cup O) \), and \( \theta_e(a) = a \) if \( a \in E_i \).

Now we are ready to state the RW problem corresponding to Problem 2.

Problem 3 (RW counterpart of Problem 2). Assume that the specification language \( K \subseteq E_i^* \cup E_i^2 \) satisfies Assumption 1, and the plant is a deterministic simple plant \( P : (U \times D)^* \rightarrow O^* \times E_i^* \) is realizable by a sequential machine . Define the Ramadge-Wonham problem with partial observations corresponding to Problem 2 as follows.

- **Controllable and uncontrollable events**
  
  Let the alphabet be \( \Sigma = (U \cup D \cup O \cup E_i) \), \( \Sigma_c = U \) be the set of controllable events, and \( \Sigma_u = D \cup O \cup E_i \) be the set of uncontrollable events.

- **Observable and unobservable events**
  
  Let the set of observable events be \( \Sigma_o = (U \cup O) \) and the set of unobservable events be \( \Sigma_u = D \cup E_i \).
• Control requirements
  Define the language of control requirements \( K_s = \theta^{-1}(K_{safe}) \), where \( K_{safe} \) is as in Assumption 1.

• Plant language
  Let the language of the plant \( G \) be the prefix closure of the language \( L_P \) from Definition 16, and let the marked language of \( G \) be \( L_P \).

With the above alphabet \( \Sigma \), plant \( G \), requirements \( K_s \) and partitioning into controllable, uncontrollable, observable and unobservable events, find a non-blocking supervisor \( S : \theta(L_P) \to 2^\Sigma \) with partial observations such that the closed-loop system satisfies \( L_m(G/S) \subseteq K_s \).

**Proposition 2 (Regularity of \( K_s \)).** If \( K_{safe} \) is regular, then the language \( K_s \) is regular, and its automaton can easily be computed from an automaton accepting \( K \).

The proof of Proposition 2 can be found in §8. Problem 3 is a classical Ramadge-Wonham control problem with partial observations with well-known solution algorithms and tools [39]

### 5.2 From a supervisor to a controller

Below we formulate a procedure to extract a sequential controller from a supervisor solving Problem 3.

**Definition 17 (Controller associated with a supervisor).** Let \( S : \theta(L_P) \to 2^U \) be a non-blocking supervisor. A sequential controller \( C : O^* \to U \) associated with \( S \) is a sequential controller satisfying the following. For each collection of outputs \( o_i \in O, i = 0, 1, 2, \ldots, k \), define

\[
\begin{align*}
    u_{i+1} &= C(o_1 o_2 \cdots o_i) \\
    w_i &= (u_1 o_1)(u_2 o_2) \cdots (u_i o_i) \in \Sigma^*
\end{align*}
\]

where \( u_i \in U \) is the control input generated by \( C \). With the notation above, for all \( i = 0, 1, 2, \ldots, k \).

\[
\forall j \leq i : w_j \in \theta(L(G/S)) \implies u_{i+1} \in S(w_i)
\]

In other words, a controller associated with a supervisor is a sequential controller which generates inputs which, if viewed as controllable events, are enabled by the supervisor. That is, if the controller generates inputs \( u_1, u_2, \ldots, u_{k+1} \) while reading \( o_1, o_2, \ldots, o_i \), then each \( u_i \) must be enabled by the supervisor \( S \) for the string \( u_1 o_1 u_2 o_2 \cdots u_i o_i \).

**Remark 5 (Non-uniqueness of the controller associated with a supervisor).** Notice that the sequential controller \( \phi_S \) is not uniquely defined; it depends on the choice of \( u_k \) at each step.

**Remark 6 (Intuition: pick any controllable action enabled by the supervisor).** The intuition behind the construction is the following. The controller associated with the supervisor simply runs a copy of the supervisor inside. The controller is activated whenever an observed output (observable, uncontrollable event) arrives. The controller then updates the state of the supervisor by feeding it the observed uncontrollable event and the controller then picks a controllable event enabled by the supervisor and sends it to the plant. Subsequently, it updates the state of the supervisor by feeding it back the chosen controllable event. Finally, the controller becomes dormant, waiting for the next activation.
The intuitive description of the controller above is quite natural. It has been implemented for example in [37]. However, the correctness of such a construction has never been dealt with formally, to the best of our knowledge.

Note that the correctness of the construction does not follow from general Ramadge-Wonham theory. The results to be presented below show, in fact, that the construction is correct for safety specifications. However, it does not, in general, preserve non-blockingness in a classical sense. In addition, the choice of controllable and uncontrollable events realized in the controller construction above is also a step which does not follow from Ramadge-Wonham theory.

**Proposition 3.** For any non-blocking supervisor $S : \theta(L_P) \rightarrow 2^U$ implementable by a finite-state automaton, there exists an associated sequential controller $\phi_S$.

**Proof of Proposition 3.** Assume that the supervisor $S$ can be represented by the automaton $A = (Q, \Sigma, \delta, F, q_0)$. That is, for any $v \in \theta(L(G/S))$, $\delta(q_0, v)$ is defined and $u \in S(v)$ if and only if $\delta(q_0, vu)$ is defined. Define the Moore-automaton $A_c = (Q_c, O, U, \delta_c, \lambda, q_c)$.

- $Q_c = (Q \times U) \cup \{\bot\}$ where $\bot \notin (Q \times U)$.
- The initial state is $q_c = (q_0, u)$, where $u \in U$ is chosen so that $\delta(q_0, u)$ is defined.
- The state-transition map $\delta_c : Q_c \rightarrow Q_c$ is defined as follows. For all $o \in O, (q, u) \in Q \times U$, if $\delta(q, uo)$ is defined, then define $\delta_c((q, u), o) = (\delta(q, uo), \hat{u})$ where $\hat{u} \in U$ is chosen such that $\delta(q, uo\hat{u})$ is defined. If no such input $\hat{u} \in U$ exists, or $\delta(q, uo)$ is not defined, then define $\delta_c((q, u), o) = \bot$. Let $\delta_c(\bot, o) = \bot$.
- The readout map $\lambda : Q_c \rightarrow U$ is defined by $\lambda((q, u)) = u$ for all $(q, u) \in Q_c$, and $\lambda(\bot) = u \in U$ for some arbitrarily chosen $u \in U$.

Note that $q_c$ is well-defined, since $\epsilon \in L(G/S)$ and any word in $L_P$ must end with an element of $U$, and hence, due to the non-blockingness of the supervisor $S$, there exists $u \in U$ such that $u \in S(\epsilon)$, i.e. $\delta(q_0, u)$ is well-defined.

Consider the input-output map $C : O^* \rightarrow U$ realized by $A_c$. We show that $C$ satisfies Definition 17. We will show by induction that for each $i$, (7) holds. For $i = 0$, $u_1 = C(\epsilon)$ is such that $(q_0, u_1)$ is the initial state of $A_\epsilon$, and hence by definition $\delta(q_0, u_1)$ is well-defined. The latter means that $u_1 \in S(\epsilon) = S(w_0)$, i.e. (7) holds for $i = 0$. Assume that (7) holds for $i \leq k - 1$. Then, define the states $q_i$ of $A_i$, $i \leq k - 1$ recursively as follows: $q_i = \delta(q_{i-1}, o_i, u_i)$, where $o_i$ and $u_i$ are as in (6), for all $i = 1, 2, \ldots, k - 1$. Notice that $q_i = \delta(q_0, w_i)$ where $w_i$ is as in (6). Since $S$ is non-blocking and any element of $L_P$ must end in a symbol from $U$, and $w_k$ ends in $o_k$, we get that there exists $u_{k+1} \in U$ such that $u_{k+1} \in S(w_k)$, i.e. $\delta(q_0, w_k u_{k+1}) = \delta(q_k, u_{k+1})$ is defined. Hence, we get that the state $q_k$ of $A_c$ reachable from the initial state under the sequence $o_1o_2\cdots o_k$ is of the form $q_k = (q_k, u_{k+1})$. Here, $u_{k+1} \in U$ is such that $\delta(q_k, u_{k+1})$ is defined and hence $u_{k+1} \in S(w_k)$. Notice that due to our definition, $u_{k+1} = C(o_1o_2\cdots o_k)$, i.e. (7) holds for $i = k$. \hfill \Box

### 5.3 Correctness of the transformation

Now we are ready to state the main theorem, relating solutions of Problem 2 with those of Problem 3.
Theorem 4. If the supervisor $S$ is a solution to Problem 3, then any sequential controller $\phi$ associated with $S$ is a solution to Problem 2, and at least one such sequential controller exists.

Proof. The second part of the statement on existence of a sequential controller associated with $S$ follows from Proposition 3.

Let now $C$ be a sequential controller associated with $S$. We will show that $L(P/C) \subseteq K$, where $K = K_{safe} \cup \lim(K_{safe})$. That is for any $\hat{\omega} \in L(P/C)$, we have to show that if $\hat{\omega}$ is finite, then $\hat{\omega} \in K_{safe}$ and if $\hat{\omega}$ is infinite, then there exists a strictly increasing sequence of indices $k_i \in \mathbb{N}$, $i \in \mathbb{N}$ such that $\hat{\omega}_{k_i} \in K_{safe}$. Since $K_{safe}$ is prefix closed, it is enough to show that

$$\forall v \in D^*: B(P/C)(v) \subseteq K_{safe}$$

(8)

Indeed, assume that (8) holds. Assume that $\hat{\omega} \in L(P/C)$. If $\hat{\omega} \in E^*_u$, then there exists $v \in D^*$ such that $\hat{\omega} \in B(P/C)(v)$. Hence, $\hat{\omega} \in K_{safe}$. If $\hat{\omega} \in E^*_i$, then there exists an infinite sequence of disturbances $v \in D^*$, and a sequence of indices $k_i, i \in \mathbb{N}$ such that $k_{i+1} \geq k_i$ and $\sup k_i = \infty$ and $\hat{\omega}_{k_i} \in B(P/C)(v_{k_i})$. From (8) it follows then that $\hat{\omega}_{k_i} \in K_{safe}$, from which by definition of $\lim(K_{safe})$ it follows that $\hat{\omega} \in \lim(K_{safe})$. That is, (8) implies that $L(P/C) \subseteq K$.

Hence, it follows that it is enough to show that (8) holds. Notice that in order to prove (8), it is enough to show that the following holds.

$$\forall v \in D^*: B(P/C)(v) \subseteq \theta_c(L(G/S))$$

(9)

Indeed, assume that (9) holds. Then for all $\hat{\omega} \in B(P/C)(v)$, $\hat{\omega} \in \theta_c(L(G/S))$. We will show that then $\hat{\omega} \in K_{safe}$. But $S$ is a solution to Problem 3, hence $L_m(G/S) \subseteq \theta^{-1}_c(K_{safe})$. Since $S$ is non-blocking and hence for every $\sigma \in L(G/S)$ there exists an $y \in \Sigma^*$ such that $\hat{\sigma}y \in L_m(G/S)$. That is, $\theta_c(\sigma) \theta_c(y) \in K_{safe}$. Since $K_{safe}$ is prefix closed, this implies that $\theta_c(\sigma) \in K_{safe}$, i.e. $\sigma \in \theta^{-1}_c(K_{safe})$. That is, $L(G/S) \subseteq \theta^{-1}_c(K_{safe})$. Hence, if $\hat{\omega} \in \theta_c(L(G/S))$, then $\hat{\omega} \in K_{safe}$.

We complete the proof of the theorem by showing that (9) holds. To this end, recall that $\hat{\omega} \in B(P/C)(v)$, if (4)–(5) hold for $v = d_1d_2\cdots d_k$ with $d_i \in D$, $i = 1, 2, \ldots, k$, $k \geq 0$. Let $o_i$, $\hat{o}_i$ and $d_i$ as in (4–5). Let $v_0 = \epsilon$ and $v_i = v_{i-1}u_i\hat{o}_i\hat{o}_i$ for all $i = 1, 2, \ldots, k$. By induction on $i$, using that $C$ is a sequential controller associated with $S$ and that $\theta(v_i) = w_i$, (here $w_i$ is as in (6)), we can show that $v_iu_{i+1} \in L(G/S)$ for all $i = 0, 1, 2, \ldots, k$. From this, (9) follows easily. Indeed notice that $\theta_c(v_ku_{k+1}) = \hat{\omega}$. Hence, $\hat{\omega} \in \theta_c(L(G/S))$.

We conclude the proof by showing that $v_iu_{i+1} \in L(G/S)$. Indeed, for $i = 0$, $v_0 = \epsilon$ and $u_1 = C(\epsilon) \in S(\epsilon)$. Since $\epsilon \in L(G/S) \cap L_P$, we get that $u_1 \in L(G/S)$. Hence, $v_0u_1 = u_1 \in L(G/S)$. Suppose now that the statement is true for $j = 0, 1, \ldots, l$ for some $l$. Then, $v_lu_{l+1} \in L(G/S)$. Then, by the definition of feedback interconnection, $(o_1o_2\cdots o_{l+1}, \hat{o}_1\hat{o}_2\cdots \hat{o}_{l+1}) = P((u_1, d_1)(u_2, d_2)\cdots (u_{l+1}, d_{l+1}))$ and $u_{l+2} = C(o_1\hat{o}_2\cdots \hat{o}_{l+1})$. Notice that $w = v_lu_{l+1}o_{l+1}\hat{o}_{l+1} \in L_P$. Since $o_{l+1}\hat{o}_{l+1}$ consists of uncontrollable events, and $v_{l+1}u_{l+1} \in L(G/S)$, we get that $w \in L(G/S)$. B From the assumption that $C$ is a controller associated with $S$, from the fact that $\theta(w) = u_1o_1\cdots o_{l+1}u_{l+1} \in \theta(L(G/S))$, it then follows that $u_{l+2} \in S(\theta(w))$ and hence $wu_{l+2} = v_{l+1}u_{l+2} \in L(G/S)$, since $wu_{l+2} \in L_P$. 

6 Illustrating example

The following example represents a subsystem of the patient table support systems for Philips MRI scanners [37]. The structure of the MRI scanner is depicted in Fig. 3 and Fig. 4. The example deals with the motion of the table along the vertical axis. The user can initiate the movement of the table by using the tumble switch. The tumble switch has three positions: up, down and neutral. The table can move up and down. There are two sensors Max down and Max up on the top and the bottom positions of the axis. They get activated, if the table reaches the corresponding positions where the sensors are located.

We need a controller, which receives the position of the tumble switch and the reading of the up- and down-most sensors, and which achieves the following. If the tumble switch is up, then the table moves up, if the tumble switch is down, then the table moves down. If the tumble switch is neutral, then the table stops. In addition, the table is not allowed to go lower (higher) than the position of the Max down and Max up sensors respectively. In particular, if the table is moving and it is getting beyond one of the sensors, then it must be stopped. That is, the controller must actively generate (force) events, it is not enough to enable or disable events. The controller can initiate the movement of the table up, or down or it can stop it. In addition, it can issue no command at all, i.e. it is a decision of the controller to do nothing.

The control problem above can be formalized in our framework as follows.
6.1 Model of the plant

Below we will describe the model of the plant. We start with the formal definition, after which we will describe the intuition behind the modeling choices.

6.1.1 Formal model of the plant

The uncontrolled plant can be modeled as a sequential (deterministic) input-output map

\[ P_{\text{model}} : (U \times V \times D)^* \rightarrow O^* \times E^* \]

where

- \( U = \{M_d, M_u, S, \emptyset\} \) is the set of control inputs
- \( V = \{T_{\text{up}}, T_d, T_n\} \) is the set of the set of external inputs
- \( D = \{E_D, E_{\text{up}}, E_{\text{D,off}}, E_{\text{U,off}}, \perp_d\} \) is the set of disturbances
- \( O = \{\perp, D_{\text{on}}, U_{\text{on}}, D_{\text{off}}, U_{\text{off}}\} \) is the set of outputs
- \( E_i = \{D_{\text{on}}, U_{\text{on}}, C\} \cup \{M_d, M_u, S\} \cup \{e_D, e_U\} \cup \{T_{\text{up}}, T_d, T_n\} \) is the set of internal events

The meaning of the various symbols are describe below.

- \( \perp \) – no sensor event took place
- \( D_{\text{on}} \) – the Max down sensor becomes active
- \( U_{\text{on}} \) – the Max up sensor becomes active
- \( D_{\text{off}} \) – the Max down sensor ceases to be active
- \( U_{\text{off}} \) – the Max up sensor ceases to be active
- \( T_{\text{up}} \) – the tumble switch is up
- \( T_d \) – the tumble switch is down
- \( T_n \) – the tumble switch is in a neutral position
- \( M_d \) – command to move down
- \( M_u \) – command to move up
- \( S \) – command for the table to stop moving
- \( E_D \) – the table is at Max down sensor
- \( E_{\text{up}} \) – the table is at Max up sensor
- \( E_{\text{D,off}} \) – the table has moved above the position of Max down sensor (i.e. Max down sensor changed from active to inactive)
- \( E_{\text{U,off}} \) – the table has moved below the the position of the Max up sensor, (i.e. Max up sensor changed from active to inactive)
• $\perp_d$ – no disturbance

• $\emptyset$ – empty command, does not effect the behavior of the system

• $D_{on}^c$ – the table is at Max down sensor

• $U_{on}^c$ – the table is at Max up sensor

• $M_d^c$ – the table is moving down

• $M_u^c$ – the table is moving up

• $S^c$ – the table is standing still

• $e_D$ – the table is beyond (below) Max down sensor

• $C$ – the table is between the the positions of Max down and Max up sensors.

• $e_U$ – the table is beyond (above) Max up sensor

• $T_{up}^c, T_d^c, T_n^c$ are the copies on the channel $E_i$ of the external inputs $T_{up}, T_d$ and $T_n$ respectively.

We model the uncontrolled plant by the map $P_{model} : (U \times V \times D)^* \rightarrow O^* \times E_i^*$. The map $P_{model}$ is defined by presenting a sequential machine $M = (Q, (U \times V \times D), O, E_i, \delta, \lambda, q_0)$ such that $M$ realizes $P_{model}$. The parameters of $M$ are as follows. Let $Q$ be the set of functions $\phi : D \rightarrow \{0, 1\}$, where $D = \{MD, MU, I, EU, ED, DB, UB\}$, i.e. each state of $T$ can be thought of as a function assigning true or false values to the predicate symbols in $D$. The intuition behind these state-variables is the following.

1. $MD$ is true, if the table is moving to down
2. $MU$ is true, if the table is moving up
3. $I$ is true, if the table is standing still
4. $EU$ is true, if the table is at Max down sensor.
5. $ED$ is true, if the table is at Max up sensor.
6. $DB$ is true, if the table is at the bottom end of the axis.
7. $UB$ is true, if the table is at the top end of the axis.

Note that not all the state components can be observed based on the sensor data.

The state transition map $\delta : Q \times (U \times V \times D) \rightarrow Q$ is defined as follows; $\delta(q_1, (u, v, d)) = q_2$ if and only if at least one of the conditions below holds.

1. If $u = M_d$, then $q_2(I) = 0$, $q_2(MU) = 0$, $q_2(UB) = 0$, $q_2(ED) = 0$ and $q_2(MD) = 1$
2. If $u = M_u$, then $q_2(I) = 0$, $q_2(MD) = 0$, $q_2(DB) = 0$, $q_2(EU) = 0$, and $q_2(MU) = 1$
3. If $u = S$, then $q_2(I) = 1$ and $q_2(x) = 0$ for $x \in \{MD, MU\}$.
4. If $d = ED$ and $q_1(EU) = 0$, then $q_2(EU) = 1$.
5. If $d = Ed$ and $q_1(ED) = 0$, then $q_2(ED) = 1$. 

Illustrating example
6. If \( d = \mathcal{E}_{D, \text{off}} \), and \( q_1(\mathcal{E}U) = 1 \) or \( q_1(\mathcal{D}B) = 1 \), then \( q_2(\mathcal{E}U) = 0 \) and \( q_2(\mathcal{D}B) = 0 \).

7. If \( d = \mathcal{E}_{U, \text{off}} \), and \( q_1(\mathcal{E}D) = 1 \) or \( q_1(\mathcal{U}B) = 1 \), then \( q_2(\mathcal{E}D) = 0 \) and \( q_2(\mathcal{U}B) = 0 \).

8. If \( q_1(\mathcal{E}U) = 1 \) and \( q_2(\mathcal{M}D) = 1 \), then \( q_2(\mathcal{D}B) = 1 \).

9. If \( q_1(\mathcal{E}D) = 1 \) and \( q_2(\mathcal{M}U) = 1 \), then \( q_2(\mathcal{U}B) = 1 \).

In addition, \( q_2(X) \) equals \( q_1(X) \) for all \( X \in \mathcal{D} \), except the cases explicitly described above. Moreover, the following should hold for \( q_1, q_2 \): for all \( X \neq Y \in \{\mathcal{M}D, \mathcal{I}, \mathcal{M}U\} \) and for all \( i = 1, 2 \), if \( q_i(X) = 1 \), then \( q_i(Y) = 0 \), i.e. always exactly one of the predicates \( \mathcal{M}D, \mathcal{I} \) and \( \mathcal{M}U \) is true. Furthermore, for the states \( q_1, q_2 \) it holds that exactly one of the predicates \( \{\mathcal{E}D, \mathcal{E}U, \mathcal{C}\} \) is true, i.e. \( \forall X \neq Y \in \{\mathcal{E}D, \mathcal{E}U, \mathcal{C}\}, q_1(X) = 1 \implies q_i(Y) = 0 \) for \( i = 1, 2 \). Similarly, the predicates \( \mathcal{DB} \) and \( \mathcal{UB} \) cannot be true simultaneously. For the initial state \( q_0 \) is holds that \( q_0(\mathcal{I}) = 1, q_0(\mathcal{M}U) = q_0(\mathcal{M}D) = 0, q_0(\mathcal{E}U) = q_0(\mathcal{E}D) = 0 \). The output map \( \lambda \colon Q \times (\mathcal{U} \times \mathcal{V} \times \mathcal{D}) \to \mathcal{O} \times \mathcal{E}_* \) is defined as follows.

\[
\lambda(q, (u, v, d)) = (\lambda_0(q, u, v, d), \lambda_1(q, u, v, d)\lambda_2(q, u, v, d)\lambda_3(q, u, v, d)\lambda_4(q, u, v, d))
\]

where

\[
\lambda_1(q, u, v, d) = \begin{cases} 
\mathcal{M}_2 & \text{if } q_2(\mathcal{M}D) = 1 \text{ and } q_2 = \delta(q, (u, v, d)) \\
\mathcal{M}_6 & \text{if } q_2(\mathcal{M}U) = 1 \text{ and } q_2 = \delta(q, (u, v, d)) \\
\mathcal{S}^c & \text{if } q_2(\mathcal{I}) = 1 \text{ and } q_2 = \delta(q, (u, v, d))
\end{cases}
\]

\[
\lambda_2(q, u, v, d) = \begin{cases} 
\mathcal{D}_{on} & \text{if } q_2(\mathcal{E}U) = 1 \text{ and } q_2 = \delta(q, (u, v, d)) \\
\mathcal{U}_{on} & \text{if } q_2(\mathcal{E}D) = 1 \text{ and } q_2 = \delta(q, (u, v, d)) \\
\mathcal{C} & \text{if otherwaise}
\end{cases}
\]

\[
\lambda_3(q, u, v, d) = \begin{cases} 
T_{up} & \text{if } v = T_{up} \\
T_d & \text{if } v = T_d \\
T_n & \text{if } v = T_n
\end{cases}
\]

\[
\lambda_4(q, u, v, d) = \begin{cases} 
e_D & \text{if } q_2(\mathcal{D}B) \text{ and } q_2 = \delta(q, (u, v, d)) \\
e_U & \text{if } q_2(\mathcal{UB}) \text{ and } q_2 = \delta(q, (u, v, d)) \\
\epsilon & \text{otherwise}
\end{cases}
\]

\[
\lambda_0(q, u, v, d) = \begin{cases} 
\mathcal{D}_{on} & \text{if } d = \mathcal{E}_D \text{ and } q(\mathcal{E}D) = 0 \\
\mathcal{U}_{on} & \text{if } d = \mathcal{E}_U \text{ and } q(\mathcal{E}U) = 0 \\
\mathcal{D}_{off} & \text{if } d = \mathcal{E}_{D, \text{off}} \text{ and } q(\mathcal{E}D) = 1 \\
\mathcal{U}_{off} & \text{if } d = \mathcal{E}_{U, \text{off}} \text{ and } q(\mathcal{E}U) = 1 \\
\bot & \text{otherwise}
\end{cases}
\]

Note that the output map depends on the current state, current control input, external input and disturbance. Notice that there is a certain arbitrariness in the definition of the map \( \lambda \) above. Namely, for our purposes, any other choice of the relative order of \( \lambda_i \), \( i = 1, 2, 3, 4 \) could have been chosen. In fact, we could have also chosen \( E_i \) as a direct product \( \{\mathcal{D}_{on}, \mathcal{U}_{on}, \mathcal{C}\} \times \{\mathcal{M}_2, \mathcal{M}_6, \mathcal{S}^c\} \times \{\ne_D, \ne_U\} \times \{T_{up}, T_d, T_n\} \).

### 6.1.2 Intuition and modeling choices

Below we will describe the intuition behind the formal model of the plant.

Before proceeding with the detailed explanation, it is useful to remark that the underlying point of view of the system is more time-driven and conforms with the classical event-driven paradigm to a lesser extent. That is, the plant is viewed as a discrete-time dynamical system which at each step receives symbols from its input channels, changes its state and generates symbols on the output channels. Here, there are three input channels: control inputs, external inputs and disturbances. There are two output channels: outputs
and internal events. This point of view is different from the event-driven paradigm, where the plant is viewed as an event generator.

First we describe the intuition behind the choice of the collections of control inputs, external inputs, disturbances, outputs and internal events.

Control inputs The only actions the controller can generate is to stop the movement of the table, or to make the table to move down or up. Hence, the choice of the set of control inputs $U$.

External inputs The actions of the operator are naturally modeled as external inputs; these actions are visible to the controller, they may occur any time, and in principle can change the system dynamics. The only action the operator can perform is to move the tumble switch. Hence the definition of the set $V$. Notice that in the current model the operator actions do not really influence the behavior of the system. They are relevant only for specifying the control requirements.

Invisible disturbances The choice of disturbance set $D$ is a bit less intuitive. More precisely, we view the situation when the table reaches/leaves the position of the Max down respectively Max up sensor as a disturbance. From a purely physical point of view this is not entirely logical, as these events are direct results of the motion of the table. Intuitively, the latter should be part of the state description. However, we are working with an abstraction of the physical plant, where the precise form of motion is not modeled and hence can be regarded as unknown. That is, we cannot predict, based on the past inputs, when the table reaches the position of sensors. In order to predict this, continuous kinematic models of table movement are required. Therefore, we chose to model the reaching of sensor positions as disturbances. The plant model only imposes conditions on when such disturbances should be regarded as valid ones.

Outputs The only visible outputs are the activation/de-activation of the Max up and Max down sensors. Hence the choice of the set of outputs $O$.

Internal events Finally, the set of internal events can be divided into three subsets. The first one consists of events which provide information on the internal state of the plant. More specifically $e_D$ and $e_U$ indicate that the table is in the down- and up-most position on the axis. These symbols are used in the formalization of control requirements, to indicate that the system exhibits unsafe behavior. The symbols $M_{c_d}$, $M_{c_u}$ and $S^c$ indicate whether the table is moving down, up, or stays idle. The symbols $D_{cn}$ or $U_{cn}$ and $C$ tell us whether the table is in the potentially critical position at the Max down or Max up sensor, or it is in the middle of the axis, between the sensors. The symbols $\{M_{c_d}, M_{c_u}, S^c, D_{cn}, U_{cn}, C\}$ are used in the specification of control requirements to formalize that the plant is in a critical state. By a critical state we mean the situation when the table is at one of the sensors Max down and Max up. In this case, if no action is taken, the system might move beyond Max down or Max up sensors. Finally, $E_I$ contains the copies of the symbols from $V$. This is necessary in order to formalize the requirement on the reaction of the plant to the actions of the operators.

Next, we describe the intuition behind the state-variable of the plant model. We need to keep track of the following information;

- Is the table is moving down, up or standing still?
- Is the table is at one of the sensors (Max down or Max up)?

Illustrating example
• Is the table is at the outermost ends of the axis?

The variables $MD, MU, I$ answer the first question, $EU, ED$ the second question, and $DB, UB$ the third question. Each state variable can take a value 0 and 1 (or true and false) and hence can be seen as a propositional predicate. Notice that the number of different value combinations of the variables is finite, hence our plant still has a finite state-space.

The state transition rules reflect the dynamics of the plant. That is, the value of the variables $MD, MU, I$ is updated according to whether the control input is to move down, to move up, or to stop. If the disturbance symbol indicates that the table is at the down- or up-most sensor, then variable $EU$ resp. $ED$ has to be set 1. Similarly, if the table moves away from $Max$ down or $Max$ up sensor position, then the variables $EU$ and $ED$ should be set to zero. If the plant is in $EU$ or $ED$ and the table is moving down (or resp. up), then the table will reach the end of the axis in the next step, hence $DB$ or resp. $UB$ has to be set to 1. Finally, certain variables are mutually exclusive, i.e. they cannot have the same value. For example, the system cannot be in a state where $EU$ and $ED$ are simultaneously true, i.e. the table cannot be at the $Max$ up or $Max$ down sensor simultaneously. The same mutual exclusivity holds for $MD, MU, I$ or $DB$ and $UB$.

Finally, the readout map $\lambda$ describes the conditions when the sensor gets activated, i.e. a visible output is generated, and it describes which internal events are generated. Notice that in one step, a sequence of internal events, rather than one internal event is generated. In the model above we fixed a specific order of symbols in the sequence of internal events generated at each step. This choice is arbitrary, and several other choices could have been possible. Alternatively, instead of generating sequences of events, we could have also modeled the plant as a device generating a subset of internal events at each step. The reason for choosing the formulation presented above that is allows for a smaller internal event alphabet and it still fits the modeling framework.

### 6.2 Control requirements

The specification says that the events $e_U$ and $e_D$ may never take place, and, unless $e_U$ or $e_D$ is at risk of happening, $T_{up}$ should lead to moving up, $T_d$ should lead to moving down, and $T_n$ should stop the movement of the table. Recall that $e_U$ and $e_D$ are generated only when the table is beyond (above) the $Max$ up or beyond (below) the $Max$ down sensor respectively. This specification is formally stated as the language $K = K_{safe} \cup \lim(K_{safe})$, where $K_{safe} = L(A_1) \cap L(A_2)$ with $L(A_i)$, $i = 1, 2$ denoting the languages generated by the automata $A_i$, $i = 1, 2$ respectively; the automaton $A_1$ is on Fig. 5, and $A_2$ is on Fig. 6. Notice that all the states of $A_1$ and $A_2$ are marked. Informally, the language of $A_2$ says that if the tumble switch is up or down, then the table may either be idle or be moving up or down respectively. In addition $A_2$ specifies that if the tumble switch is neutral, the table must be idle.

The automaton $A_1$ describes how the plant must react to the position of the tumble switch. First of all, $A_1$ says that unless we have reached the position of $Max$ down or $Max$ up sensor, the table must be moving up or down, if the tumble switch is respectively up or down. If we have reached the position of $Max$ down or $Max$ up, then $A_1$ relaxes this requirement, and the table is allowed to stop, as it might be necessary to avoid reaching the very (top or bottom) end of the axis. The latter events, i.e. reaching the very end of bottom or top side of the axis, are events which must not occur under any circumstances. However, as soon as the table is between the sensors again, i.e. the plant
Figure 5: Automaton $A_1$

Figure 6: Automaton $A_2$
emits $C$, the requirements of the normal mode of operation must be respected, i.e. the plant again must move up if the tumble switch is up, etc.

Notice that in the final control requirements, because of the conjunction with $A_2$, the plant is never allowed to move up if the tumble switch is down, and conversely, the plant is never allowed to move down if the tumble switch is up.

The labels on the edges are pairs of external inputs and observable outputs. The symbol in the state after the space is the control input corresponding to the state. The state from which there is an ingoing arrow without an origin is the initial state; i.e. the state labeled by $S60$ is the initial one.

6.3 Control synthesis

We applied the presented control synthesis method to the plant $T$ and specification $K$. Notice that the plant is not a simple deterministic plant in our case, due to the presence of external inputs. Recall that if a sequential input-output map is realized by a sequential machine $M$, then it is quasi-recognizable. Hence, the sequential machine $M$ in fact defines a sequential and deterministic input-output map, which is quasi-recognizable, but which still includes external inputs. Recall that any sequential machine can be viewed as a quasi-sequential deterministic transducer. Hence, we can apply Theorem 3 to $M$, viewed as a quasi-sequential deterministic transducer, in order to obtain a simple deterministic plant. Since the sequential input-output relation realized by $M$ is itself deterministic, in the construction of the simple deterministic plant the set $D_{ax}$, describing the non-deterministic choice, can be omitted.

We applied the synthesis procedure of Section 5 to the simple deterministic plant obtained from $M$. That is, an automation accepting the language $L_P$ corresponding to $P = S(P_{model})$ was computed as described in the proof of Proposition 1. Likewise, an automaton $K_s$ accepting $\theta^{-1}(K_{safe})$ was constructed. Subsequently, the classical algorithm for supervisory control with partial observations were applied to $L_P$ and $K_s$ to compute a supervisory $S$. From the latter, we extracted a Moore-automaton of the desired controller using the algorithm of the proof of Proposition 3. Note that the algorithm in fact returns a set of controllers solving the control problem. The controller presented here represents one specific choice from the set of admissible controllers returns by the algorithm. We then applied the classical minimization procedure [13, 16] and obtained a Moore-automaton realization of the controller with 8 states. All the above steps were carried out using the implementation of the constructions of this paper. Due to the lack of space, we will not present the automaton recognizing $L_R$. The Moore-automaton of the controller is presented in Table 1–2. The state of the controller are \{S00, S10, S20, S30, S40, S50, S60\}, the initial state is S60. The complete specification of the state-transition and readout maps can be found in Table 1–2.

The example above is a part of an industrial case [37]. In [37], a controller was synthesized using RW theory and tested on a physical prototype. In [37] the extraction of a controller from the supervisor was done by using engineering insights about the relevant use-case.

7 Conclusions

We have presented a novel framework for control of discrete-event systems, which allows for explicit modeling of inputs and output and active generation of events by the con-
Table 1: State-transition map of the controller.

<table>
<thead>
<tr>
<th>Source state</th>
<th>Target state</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>S4O</td>
<td>S6O</td>
<td>(T_n, D_{off})</td>
</tr>
<tr>
<td>S4O</td>
<td>S0O</td>
<td>(T_d, D_{on}), (T_d, \bot)</td>
</tr>
<tr>
<td>S4O</td>
<td>S1O</td>
<td>(T_d, D_{off})</td>
</tr>
<tr>
<td>S4O</td>
<td>S5O</td>
<td>(T_{up}, D_{off})</td>
</tr>
<tr>
<td>S4O</td>
<td>S4O</td>
<td>(T_{up}, D_{on}), (T_{up}, \bot), (T_n, \bot), (T_n, D_{on})</td>
</tr>
<tr>
<td>S5O</td>
<td>S1O</td>
<td>(T_d, \bot)</td>
</tr>
<tr>
<td>S5O</td>
<td>S6O</td>
<td>(T_n, \bot)</td>
</tr>
<tr>
<td>S5O</td>
<td>S2O</td>
<td>(T_{up}, U_{on})</td>
</tr>
<tr>
<td>S5O</td>
<td>S4O</td>
<td>(T_n, D_{on}), (T_{up}, D_{on})</td>
</tr>
<tr>
<td>S5O</td>
<td>S3O</td>
<td>(T_d, U_{on}), (T_n, U_{on})</td>
</tr>
<tr>
<td>S5O</td>
<td>S5O</td>
<td>(T_d, D_{on})</td>
</tr>
<tr>
<td>S5O</td>
<td>S2O</td>
<td>(T_{up}, \bot)</td>
</tr>
<tr>
<td>S6O</td>
<td>S4O</td>
<td>(T_n, D_{on}), (T_{up}, D_{on})</td>
</tr>
<tr>
<td>S6O</td>
<td>S6O</td>
<td>(T_n, \bot)</td>
</tr>
<tr>
<td>S6O</td>
<td>S1O</td>
<td>(T_d, \bot)</td>
</tr>
<tr>
<td>S6O</td>
<td>S0O</td>
<td>(T_d, D_{on})</td>
</tr>
<tr>
<td>S6O</td>
<td>S5O</td>
<td>(T_{up}, \bot)</td>
</tr>
<tr>
<td>S6O</td>
<td>S3O</td>
<td>(T_n, U_{on}), (T_d, U_{on})</td>
</tr>
<tr>
<td>S6O</td>
<td>S2O</td>
<td>(T_{up}, U_{on}), (T_{up}, \bot)</td>
</tr>
<tr>
<td>S3O</td>
<td>S1O</td>
<td>(T_d, U_{off})</td>
</tr>
<tr>
<td>S3O</td>
<td>S3O</td>
<td>(T_d, U_{off}), (T_n, U_{on}), (T_n, U_{off}), (T_d, \bot)</td>
</tr>
<tr>
<td>S3O</td>
<td>S5O</td>
<td>(T_{up}, U_{off})</td>
</tr>
<tr>
<td>S3O</td>
<td>S2O</td>
<td>(T_{up}, U_{on}), (T_{up}, \bot)</td>
</tr>
<tr>
<td>S3O</td>
<td>S6O</td>
<td>(T_n, U_{off})</td>
</tr>
<tr>
<td>S2O</td>
<td>S1O</td>
<td>(T_d, \bot)</td>
</tr>
<tr>
<td>S2O</td>
<td>S1O</td>
<td>(T_d, U_{off}), (T_n, U_{on})</td>
</tr>
<tr>
<td>S2O</td>
<td>S5O</td>
<td>(T_{up}, U_{off}), (T_{up}, \bot)</td>
</tr>
<tr>
<td>S2O</td>
<td>S2O</td>
<td>(T_{up}, U_{on})</td>
</tr>
<tr>
<td>S2O</td>
<td>S3O</td>
<td>(T_d, U_{on}), (T_n, U_{on})</td>
</tr>
<tr>
<td>S2O</td>
<td>S1O</td>
<td>(T_n, \bot)</td>
</tr>
<tr>
<td>S2O</td>
<td>S1O</td>
<td>(T_d, \bot)</td>
</tr>
<tr>
<td>S2O</td>
<td>S5O</td>
<td>(T_{up}, \bot)</td>
</tr>
<tr>
<td>S2O</td>
<td>S0O</td>
<td>(T_d, D_{on})</td>
</tr>
<tr>
<td>S1O</td>
<td>S3O</td>
<td>(T_n, U_{on}), (T_d, U_{on})</td>
</tr>
<tr>
<td>S0O</td>
<td>S4O</td>
<td>(T_{up}, D_{on}), (T_n, D_{on})</td>
</tr>
<tr>
<td>S0O</td>
<td>S0O</td>
<td>(T_d, D_{on})</td>
</tr>
<tr>
<td>S0O</td>
<td>S5O</td>
<td>(T_{up}, \bot), (T_{up}, D_{off})</td>
</tr>
<tr>
<td>S0O</td>
<td>S6O</td>
<td>(T_n, D_{off}), (T_n, \bot)</td>
</tr>
<tr>
<td>S0O</td>
<td>S1O</td>
<td>(T_d, \bot), (T_d, D_{off})</td>
</tr>
</tbody>
</table>
Table 2: Readout map of the controller.

<table>
<thead>
<tr>
<th>State</th>
<th>Output of the controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0O</td>
<td>$\mathcal{M}_u$</td>
</tr>
<tr>
<td>S1O</td>
<td>$\mathcal{M}_u$</td>
</tr>
<tr>
<td>S2O</td>
<td>$\mathcal{M}_d$</td>
</tr>
<tr>
<td>S3O</td>
<td>$S$</td>
</tr>
<tr>
<td>S4O</td>
<td>$S$</td>
</tr>
<tr>
<td>S5O</td>
<td>$\mathcal{M}_d$</td>
</tr>
<tr>
<td>S6O</td>
<td>$S$</td>
</tr>
</tbody>
</table>

troller. We also show that under suitable assumptions the proposed control problem can be solved using the classical RW framework. We illustrated the theory by an example inspired by an industrial case.

Future research direction include tackling a more general version of the proposed control problem by using games on automata [23, 19]. In addition, we would like to investigate the possibility of reducing computational complexity of the solution. Furthermore, we plan more case studies for validating the theory on industrial examples.

Acknowledgments The authors thank R. Schiffelers and P. Collins for useful discussion and remarks.


8 Technical proofs

Proof of Theorem 1. Consider the monoid \( M = \Sigma^* \times X^* \times Y^* \). \( R \) is recognizable if \( \text{graph} \, R \) is a recognizable subset of \( M \). The latter means that there exists a finite monoid \( S \), a subset \( F \subseteq S \) and a monoid morphism \( \phi: M \to Q \) such that \( \text{graph} \, R = \phi^{-1}(F) \).

Define the submonoid \( M_s = (\Sigma \times X)^* \times Y^* \) of \( M \). Notice that \( M_s \) is indeed a sub-monoid of \( M \); \( M_s \) can be identified with the subset \( \{(u, x, y) \in M \mid |u| = |x| \} \) of \( M \). It is easy to see that \( \text{graph} \, R \) is a subset of \( M_s \). Consider the restriction \( \psi = \phi|_{M_s}: M_s \to Q \). Then it follows that \( \text{graph} \, R = \phi^{-1}(F) = \psi^{-1}(F) \subseteq M_s \). Notice that \( 1 = (\epsilon, \epsilon) \in R \), hence \( \psi(1) \in F \), where 1 is the neutral element of \( M_s \subseteq M \). Define the DFA \( T = (F, M_s, E, s_0, F) \) where the components of \( T \) are defined as follows.

- \( s_0 = \psi(1) \)
- \( E: F \times (\Sigma \times X) \times Y^* \to F \) is a partial map defined as follows. For any \( q \in F \) such that \( q = \psi(m) \) for some \( m \in R \), then for any \( u \in \Sigma, x \in X, y \in Y^* \).

\[
E(q, (u, x, y)) = \begin{cases} \psi(m(u, x, y)) & \text{if } m(u, x, y) \in F \\ \text{undefined} & \text{otherwise} \end{cases}
\]  

(10)

We will argue that \( T \) is well-defined and it is a QSTD. To see that \( T \) is well-defined, it is enough to show that the definition of the left-hand side of (10) is independent of the choice of \( m \) and the resulting map \( E \) is finite. In order to show the former, if \( \psi(m) = q = \psi(m') \) for some \( m \neq m' \in M \), then \( \psi(m(u, x, y)) = \psi(m'(u, x, y)) \). To see this, notice that \( \psi(m(u, x, y)) = \psi(m)\psi((u, x, y)) = \psi(m)\psi((u, x, y)) = \psi(m'(u, x, y)) \), since \( \psi \) is a monoid morphism.

As to the finiteness of \( E \), notice that \( E(q, (u, x, y)) \) is defined if and only if \( m(u, x, y) \in \text{graph} \, R \). Suppose \( m = (u, x, y) \). Then \( R(uy) \) is finite, and hence there exists finitely many \( (u, x, y) \in \Sigma \times X \times Y^* \) such that \( (ux, yy) \in R(uy) \). Next, we argue that \( T \) is a QSTD. Condition 1 and 2 of Definition 6 hold by construction. Hence, it is left to show that condition 3 of Definition 6 holds. To this end, if \( q = \psi(m) \), then \( m \in \text{graph} \, R \). Suppose that \( m = (u, x, y) \). Since \( R \) is a sequential input-output relation, it follows that for any \( u \in \Sigma \), there exists \( x \in X \) and \( y \in Y^* \) such that \( m_n = (uy, x, yy) \in \text{graph} \, R \). But then it follows that \( E(q, (u, x, y)) = \psi(m_n) \in F \).

It is left to show that the DFA \( T \) accepts precisely \( R \). It is easy to see that \( m \in \text{graph} \, R \) if and only if \( \psi(m) \in F \). On the other hand, \( T \) accepts a element \( m \in M_s \) if and only if there
exists a successful run of \( T \) on \( m \). The latter means that there exists a decomposition \( m = m_1 m_2 \cdots m_k \), \( m_1, m_2, \ldots, m_k \in M_s \) and states \( q_0, q_1, \ldots, q_k \in F \) such that \( q_0 = s_0 \) and for all \( i = 1, 2, \ldots, k \), \( q_i = E(q_{i-1}, m_i) \). But this means that \( q_k = \psi(m_1 m_2 \cdots m_k) = \psi(m) \in F \), i.e. \( m \in \text{graph } R \). That is, any element of \( m \in \text{graph } R \) is accepted by \( T \). Conversely, suppose that \( m \in M \) is such that \( m \) is accepted by \( T \). Then there exists a decomposition \( m = m_1 m_2 \cdots m_k \) such that \( k \geq 0, m_1, m_2, \ldots, m_k \in M \) and there exists \( q_0, q_1, \ldots, q_k \in Q \) such \( q_{i+1} = E(q_i, m_{i+1}) \) is defined for all \( i = 1, 2, \ldots, k \). But the state transition map \( E \) of \( T \) is defined only for elements of \( M_s \) and if \( q_i \in F \), then \( q_{i+1} \in F \) by the definition of \( E \). Moreover, from the definition of \( E \) we get that \( q_1 = E(q_0, m_1) = \psi(m_1), q_2 = E(q_1, m_2) = \psi(m_1 m_2) \), and so on, and hence \( q_k = \psi(m_1 m_2 \cdots m_k) = \psi(m) \). Hence, \( q_k = \psi(m) \in F \). But \( \psi^{-1}(F) = \text{graph } R \), and hence \( m \in \text{graph } R \).

\[ \square \]

**Proof of Theorem 3.** Consider the plant \( \mathbb{P} \) and let \( T = (Q, M, E, F, q_0) \) be a quasi-sequential deterministic transducer accepting \( \mathbb{P} \). Let \( k \) be the maximal cardinality of the sets \( E(q, a, .) = \{ o \mid E(q, a, o, \hat{o}) = \hat{q} \text{ for some } \hat{o} \in O, \hat{o} \in E_i^* \} \) defined for all \( q \in Q, a \in \{ U \times V \times D \} \), i.e. \( k = \max_{q \in Q} \max_{a \in \{ U \times V \times D \}} |E(q, a, .)| \). Define the set \( D_{ax} = \{ 1, 2, \ldots, k \} \). Fix an ordering of each set \( E(q, a, .) \) such that \( E(q, a, .) = \{ q_1, q_2, \ldots, q_k \} \), where \( |E(q, a, .)| = i \) and \( E(q, a, a_j, \hat{o}_j) = q_i \) for some \( a_j \in O \) and \( \hat{o}_j \in E_i^* \) and for all \( j = 1, 2, \ldots, i \). For each \( j = 1, 2, \ldots, k \), define the element \( E(q, a, j) \in Q \times O \times E_i^* \) as follows.

\[
E(q, a, j) = \begin{cases} 
(q_j, o_j, \hat{o}_j) & \text{if } j \leq i \\
(q_i, o_i, \hat{o}_i) & \text{if } j > i
\end{cases}
\]

Consider the sequential machine \( \mathcal{M} = (Q, U \times \hat{D}, O, E_i, \delta, \lambda, q_0) \), where the transition map \( \delta : Q \times U \times \hat{D} \rightarrow Q \) and \( \lambda : Q \rightarrow \hat{O} \) are defined as follows. For all \( u \in U, (v, d, i) \in \hat{D}, q \in Q \).

- \( \delta(q, u, (v, d, i)) = \hat{q} \) where \( E(q, (u, v, d), i) = (q, o, \hat{o}) \) for some \( o \in O \) and \( \hat{o} \in E_i^* \).
- \( \lambda(q, u, (v, d, i)) = ((o, v), \hat{o}) \), where \( E(q, (u, v, d), i) = (q, o, \hat{o}) \) for some \( \hat{q} \in Q \).

Define the sequential input-output map \( \mathbf{S}(\mathbb{P}) : (U \times \hat{D}) \rightarrow \hat{O}^* \times E_i^* \) as the map realized by the sequential machine \( \mathcal{M} \). It is then easy to see that \( \mathbf{S}(\mathbb{P}) \) satisfies the conditions of the theorem.

Indeed, consider a sequence \( s \in (U \times \hat{D})^* \) of the form

\[
s = (u_1, (v_1, d_1, j_1))(u_2, (v_2, d_2, j_2)) \cdots (u_k, (v_k, d_k, j_k))
\]

where \( u_i \in U, v_i \in V, d_i \in D \) and \( j_i \in D_{aux} \) for \( i = 1, 2, \ldots, k \). Define the sequence of states \( q_i = \delta(q_{i-1}, (u_i, (v_i, d_i, j_i))) \), letters \( o_i \in O \) and words \( \hat{o}_i \in E_i^* \) such that \( \lambda(q_{i-1}, (u_i, (v_i, d_i, j_i))) = ((o_i, v_i, \hat{o}_i)) \). If \( \mathbf{S}(\mathbb{P})(s) = (\hat{q}, \hat{o}) \), then \( \hat{o} = (o_1, d_1)(o_2, d_2) \cdots (o_k, d_k) \) and \( \hat{o} = \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k \). However, the definition of \( \delta \) and \( \lambda \) yields that \( E(q_{i-1}, (u_i, v_i, o_i, \hat{o}_i)) = q_i \) for \( i = 1, 2, \ldots, k \). Hence, since \( \mathbf{P} \) is recognized by \( T \), we get that \( (o_1, o_2, \ldots, o_k, \hat{o} \in R((u_1, v_1, d_1)(u_2, v_2, d_2) \cdots (u_k, v_k, d_k)) \). In other words, if \( \mathbf{S}(\mathbb{P})(s) = (\hat{q}, \hat{o}) \), then \( \Pi_{\mathbf{O}}(\hat{q}, \hat{o}) \in \mathbf{P}(\Pi_{U \times V \times D}(s)) \). Moreover, it follows that \( \Pi_{\mathbf{O}}(s) = \Pi_{\mathbf{O}}(a) = d_1 d_2 \cdots d_k \). Conversely, if \( \hat{v} = (u_1, v_1, d_1, j_1) \cdots (u_k, v_k, d_k, j_k) \) and \( (\hat{q}, \hat{o}) \in \mathbf{P} \), then from the assumption that \( T \) recognizes \( \mathbf{P} \), it follows that there exists letters \( o_1, o_2, \ldots, o_k \in O \) and words \( \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k \in E_i^* \), such that \( \hat{q} = (o_1, o_2, \ldots, o_k, \hat{o} \in R((u_1, v_1, d_1)(u_2, v_2, d_2) \cdots (u_k, v_k, d_k)) \). From the definition of \( \delta \) and \( \lambda \) it follows that \( q_i = \delta(q_{i-1}, (u_i, v_i, d_i, j_i)) \) and \( ((o_i, v_i, \hat{o}_i) = \lambda(q_{i-1}, (u_i, v_i, d_i, j_i)), i = 1, 2, \ldots, k \). But then from the definition of \( D_{aux} \) it follows that there exists \( j_1, j_2, \ldots, j_k \in D_{aux} \) such that \( q_1 = E(q_{i-1}, u_i, (v_i, d_i, j_i)) \), \( o_i = o_j, \hat{o}_i = \hat{o}_j, i = 1, 2, \ldots, k \). From the definition of \( \delta \) and \( \lambda \) it the follows that \( \hat{q}_1 = \delta(q_{i-1}, (u_i, v_i, d_i, j_i)) \) and \( ((o_i, v_i, \hat{o}_i) = \lambda(q_{i-1}, (u_i, v_i, d_i, j_i)), i = 1, 2, \ldots, k \). Hence, we get that \( \mathbf{S}(\mathbb{P})(s) = (\hat{q}, \hat{o}) \), where \( \hat{q} = (o_1, o_2, \ldots, o_k, \hat{o} \in R((u_1, v_1, d_1)(u_2, v_2, d_2) \cdots (u_k, v_k, d_k)) \) and \( (\hat{q}, \hat{o}) \in \mathbf{P} \).
(o_1, v_1)(o_2, v_2) \cdots (o_k, v_k) \in \tilde{O}^* and s is as in (11). But from the construction of s and \gamma it follows that |s| = |v| = k, \Pi_U \times V \times D(s) = \hat{v} and \Pi_O(\gamma) = \gamma and \Pi_V(\gamma) = \Pi_V(\hat{v}). Hence, the S(P) satisfies the first part of the theorem.

The second part of the theorem can be shown as follows. Notice that the set \tilde{O} equals the set V \times O, hence any controller for P is also a controller for S(P). Hence, in order to prove the second statement of the theorem, it is left to show that L(P/C) = L(S(P)/C). First, we show that L(P/C) \subseteq S(P)/C. Indeed, if \hat{\gamma} \in L(P/C), then there exists \gamma \in (V \times D)^n such that v = (v_1, d_1)(v_2, d_2) \cdots (v_k, d_k), and there exist \alpha_i \in O, \hat{o}_i \in E_i^*, u_i \in U, i = 1, 2, \ldots,

\forall i = 1, 2, \ldots: u_i = C((o_1, v_1)(o_2, v_2) \cdots (o_{i-1}, v_{i-1}))

\forall i = 1, 2, \ldots:

(\hat{o}_1 \hat{o}_2 \cdots \hat{o}_i, o_1 o_2 \cdots o_i) \in P((u_1, d_1, v_1)(u_2, d_2, v_2) \cdots (u_i, d_i))

\hat{\gamma} = \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k \quad \text{i.e., \hat{u}_{1,k} = \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k} \quad \text{k = [\hat{o}_1 \hat{o}_2 \cdots \hat{o}_k]} \quad (12)

Since T recognizes P, (12) means exactly that there exists a sequence of states q_i \in Q, such that E(q_{i-1}, u_i, v_i, d_i, o_i, \hat{o}_i) = q_i for all i = 1, 2, \ldots. But then it follows from the construction of \delta and \lambda that \delta(q_{i-1}, u_i, v_i, d_i, j_i) = q_i and \lambda(q_{i-1}, u_i, v_i, d_i, j_i) = (o_i, \hat{o}_i) for some j_i \in E_{aux, i} for all i = 1, 2, \ldots. Hence, we get that

\((o_1, v_1)(o_2, v_2) \cdots (o_k, v_k), \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k) = S(P)(((u_1, (v_1, d_1, j_1)) \cdots (u_k, (d_k, v_k, j_k)))

Hence, for \hat{D}^\omega \ni s = (v_1, d_1, j_1)(v_2, d_2, j_2) \cdots (v_k, d_k, j_k), \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k s = B(S(P)/C)(s_1, s)

for i \in \mathbb{N}, i.e. \hat{\gamma} \in L(S(P)/C). As for the other inclusion L(S(P)/C) \subseteq L(P/C), consider any \hat{\gamma} \in L(S(P)/C). Then, there exists u_i \in U, d_i D, j_i \in D_{aux}, o_i \in O and \hat{o}_i \in E_i^* for all i = 1, 2, \ldots, such that

\forall i = 1, 2, \ldots: u_i = C((o_1, v_1)(o_2, v_2) \cdots (o_{i-1}, v_{i-1}))

\forall i = 1, 2, \ldots:

\((o_1, v_1)(o_2, v_2) \cdots (o_k, v_k), \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k) =

S(P)(((u_1, (v_1, d_1, j_1)) \cdots (u_k, (v_k, d_k, j_k)))

\hat{\gamma} = \hat{o}_1 \hat{o}_2 \cdots \hat{o}_k \cdots \hat{o}_k \quad (13)

But from the first statement of the theorem it is clear that (13) is equivalent to (12). As we have seen above, the latter is equivalent to \hat{\gamma} \in L(P/C).

The third part of the theorem follows easily from the second part.

\[\square\]

Proof of Proposition 1. Denote by \Sigma = (U \cup D \cup O \cup E_i) the alphabet over which L_P is defined. Let \mathcal{M} = (Q, U \times D, O, E_i, \delta, \lambda, q_0) be a sequential machine which is a realization of P. Based on \mathcal{M} we will construct a deterministic finite state automaton \mathcal{A} accepting L_P.

Notice that the set \mathcal{U} = \{v \in E_*^+ \mid \exists q \in Q, o \in O, u \in U, d \in D, \lambda(q, u, d) = (o, v)\} is finite. Denote by \tilde{U} the set of prefixes of elements of \mathcal{U}. Notice that \tilde{U} is a finite set as well. Define the sets Q_0 = Q \times U, Q_1 = Q_0 \times D, Q_2 = Q_1 \times O \times \tilde{U}. The sets Q_i, i = 0, 1, 2 are disjoint and finite. Let q_1 be a symbol not in \bigcup_{i=0}^{2} Q_i. Define the set \hat{Q} = \{q_1\} \cup \bigcup_{i=0}^{2} Q_i. Define the partial map E : \hat{Q} \times \Sigma \rightarrow \hat{Q} as follows. E(q_1, e) = \hat{q}_2 and E(q_1, e) is defined if and only if one of the following conditions hold.

1. \hat{q}_1 = q_1 and e = u \in U and \hat{q}_2 = (q_0, u) \in Q_0
2. \( \hat{q}_1 = (q, u) \in Q_0 \) and \( e = d \in D \) and \( \hat{q}_2 = (q, u, d) \in Q_1 \\
3. \( \hat{q}_1 = (q, u, d) \in Q_1 \) and \( e = o \in O \) and \( \hat{q}_2 = (q, u, d, o, e) \in Q_2 \\
4. \( \hat{q}_1 = (q, u, d, o, v) \in Q_2 \) and \( e \in E_i \) and \( v \in \bar{U} \) and \( \hat{q}_2 = (q, u, d, o, ve) \in Q_2 \\
5. \( \hat{q}_1 = (q, u, d, o, v) \in Q_2 \) and \( e = u_1 \in U \) and \( v \in U \) and \( \hat{q}_2 = (q_2, u_1) \in Q_0 \) such that \( q_2 = \delta(q_1, (u, d)) \) and \( (o, v) = \lambda(q_1, (u, d)) \).

Define the set of accepting states as \( \hat{F} = Q \times U \). Define now the deterministic automaton \( A = (\hat{Q}, \Sigma, E, \hat{F}, q_\bot) \), i.e. the state-space of \( A \) is \( \hat{Q} \), \( A \) is defined over the alphabet of \( \Sigma \), the state-transition relation is given by \( E \), \( \hat{F} \) is the set of accepting states and \( q_\bot \) is the initial state. It is easy to see that \( A \) accepts the language \( L_P \), hence \( L_P \) is indeed regular.

We complete the proof of the proposition by showing that \( A \) indeed accepts \( L_P \). First, let \( s \in L_P \) be of the form

\[ s = (u_0d_1o_1\dot{o}_1)(u_1d_2o_2\dot{o}_2)\cdots(u_{k-1}d_{k-1}o_{k-1}\dot{o}_{k-1})(u_k) \]

(14)

We claim the following.

**Claim 1** \( E(q_{\bot}, s) \) is defined, \( E(q_{\bot}, s) = (q_k, u_k) \in \hat{F} \), where \( q_k = \delta(q_0, (u_0, d_1)(u_2, d_2)\cdots(u_{k-1}, d_k)) \).

The claim above implies that any string from \( L_P \) is accepted by \( A \). We prove **Claim 1** by induction on \( k \). If \( k = 0 \), then \( s = u_0 \) and \( E(q_{\bot}, u) = (q_0, u) \in Q_0 \). Assume that for \( \hat{s} = (u_0d_1o_1\dot{o}_1)(u_1d_2o_2\dot{o}_2)\cdots(u_{k-2}d_{k-1}o_{k-1}\dot{o}_{k-1})u_{k-1} \) it holds that \( E(q_{\bot}, \hat{s}) = \hat{q} \in \hat{F} \) and \( E(q_{\bot}, \hat{s}) \) is defined. Then \( \hat{q} = (q_{k-1}, u_{k-1}) \), where \( q_{k-1} = \delta(q_0, (u_0, d_1)\cdots(u_{k-2}, d_{k-1})) \) by the induction assumption. Then

\[ E(\hat{q}, d_k) = (q_{k-1}, u_{k-1}, d_k) \] and \( E(\hat{q}, d_ko_k) = (q_{k-1}, u_{k-1}, d_k, o_k) \)

Moreover, since \( s \in L_P \), it follows that \( (\dot{o}_k, o_k) = \lambda(q_{k-1}, u_k, d_k) \), and hence any prefix of \( \hat{o}_k \) is in \( U \). Therefor, by definition of \( E \), \( E(\hat{q}, d_ko_k\dot{o}_k) = (q_{k-1}, u_{k-1}, d_k, o_k, \dot{o}_k) \). Hence, from the definition of \( E \) it follows that \( E(\hat{q}, d_ko_k\dot{o}_k, u_k) = (\delta(q_{k-1}, u_k, d_k), u_k) \in \hat{F} \) as required.

Next, we prove that any string accepted by \( A \) belongs to \( L_P \). To this end, we will first show that a word \( v \in \Sigma^* \) induces a successful run in \( A \) from \( q_\bot \), if and only if \( v \) belongs to the prefix closure of \( L_P \). The proof is by induction on the length of \( v \). If \( v = \epsilon \), then \( e = \epsilon \in L_P \) and \( v \) trivially induces a successful run. Assume that \( v = \hat{v}a \) for some \( a \in \Sigma \) and \( \hat{v} \in \Sigma^* \). Assume that \( v \) induces a successful run in \( A \), i.e. \( E(q_{\bot}, v) = q \) is defined. Then \( E(q_{\bot}, \hat{v}) = \hat{q} \) is defined as well and \( q = E(\hat{q}, a) \). Hence \( \hat{v} \) is in the prefix closure of \( L_P \). Since \( \hat{v} \) is a prefix of a word from \( L_P \), it must satisfy exactly one of the conditions below.

- (1) \( \hat{v} = s \), or
- (2) \( \hat{v} = sd_{k+1}, \) or
- (3) \( \hat{v} = sd_{k+1}o_{k+1}, \) or
- (4) \( \hat{v} = sd_{k+1}o_{k+1}\dot{o}_{k+1} \)

where \( s \) is of the form (14), \( s \in L_P \) and \( d_{k+1} \in D \), and \( o_{k+1} \in O \) and \( \dot{o}_{k+1} \in E_i^* \) is such that \( (o_1o_2\cdots o_{k+1}, \dot{o}_1\dot{o}_2\cdots \dot{o}_{k+1}) = P((d_1, u_1)(d_2, u_2)\cdots(d_k, u_k)) \). Notice that by **Claim 1**, \( E(q_{\bot}, s) = (q_k, u_k) \) where \( q_k = \delta(q_0, (u_0, d_1)\cdots(u_{k-1}, d_k)) \). Then for case
(1), $E(q_\perp, \hat{v}) = (q_k, u_k)$, for case (2) $E(q_\perp, \hat{v}) = (q_k, u_k, d_{k+1})$, for case (3) $E(q_\perp, \hat{v}) = (q_k, u_k, d_{k+1}, o_{k+1})$ and for case (4) $E(q_\perp, \hat{v}) = (q_k, u_k, d_{k+1}, o_{k+1}, \hat{o}_{k+1})$. Hence, $q = E(\hat{q}, a)$ must satisfy one of the following cases.

- $q = (q_k, u_k, a)$ and $a \in D$, if $\hat{v}$ is as in (1),
- $q = (q_k, u_k, d_{k+1}, a)$ and $a \in O$, if $\hat{v}$ is as in (2),
- $q = (q_k, u_k, d_{k+1}, o_{k+1}a)$ and $a \in E$, if $\hat{v}$ is as in (3),
- $q = (\delta(q_k, a), a)$ and $a \in U$, if $\hat{v}$ is as in (4).

But for all the possibilities above we get that $v = \hat{v}a$ belongs to the prefix closure of $L_P$, if $\hat{v}$ belongs to the prefix closure of $L_P$.

We are now ready to prove that any word accepted by $A$ belongs to $L_P$. Notice that $s \in \Sigma^*$ is accepted by $A$, if $E(q_\perp, s)$ is defined and $E(q_\perp, s) \in \hat{F}$. By the previous paragraph, $s$ must be in the prefix closure of $L_P$. But $E(q_\perp, s) \in \hat{F}$ implies that $s$ ends in a symbol from $U$. It is easy to see from the definition of $L_P$ that if the word $s$ is a prefix of a word in $L_P$ and $s$ ends in a symbol from $U$, then $s$ itself belongs to $L_P$.

We conclude the proof by remarking that the construction of $A$ above is evidently computationally effective.

Proof of Proposition 2. From the well-known result in the theory of formal languages [13, 5], it follows that the inverse image $K_s$ of the regular language $K_{safe}$ by the morphism $\theta : \Sigma^* \to E_1^*$ is regular.