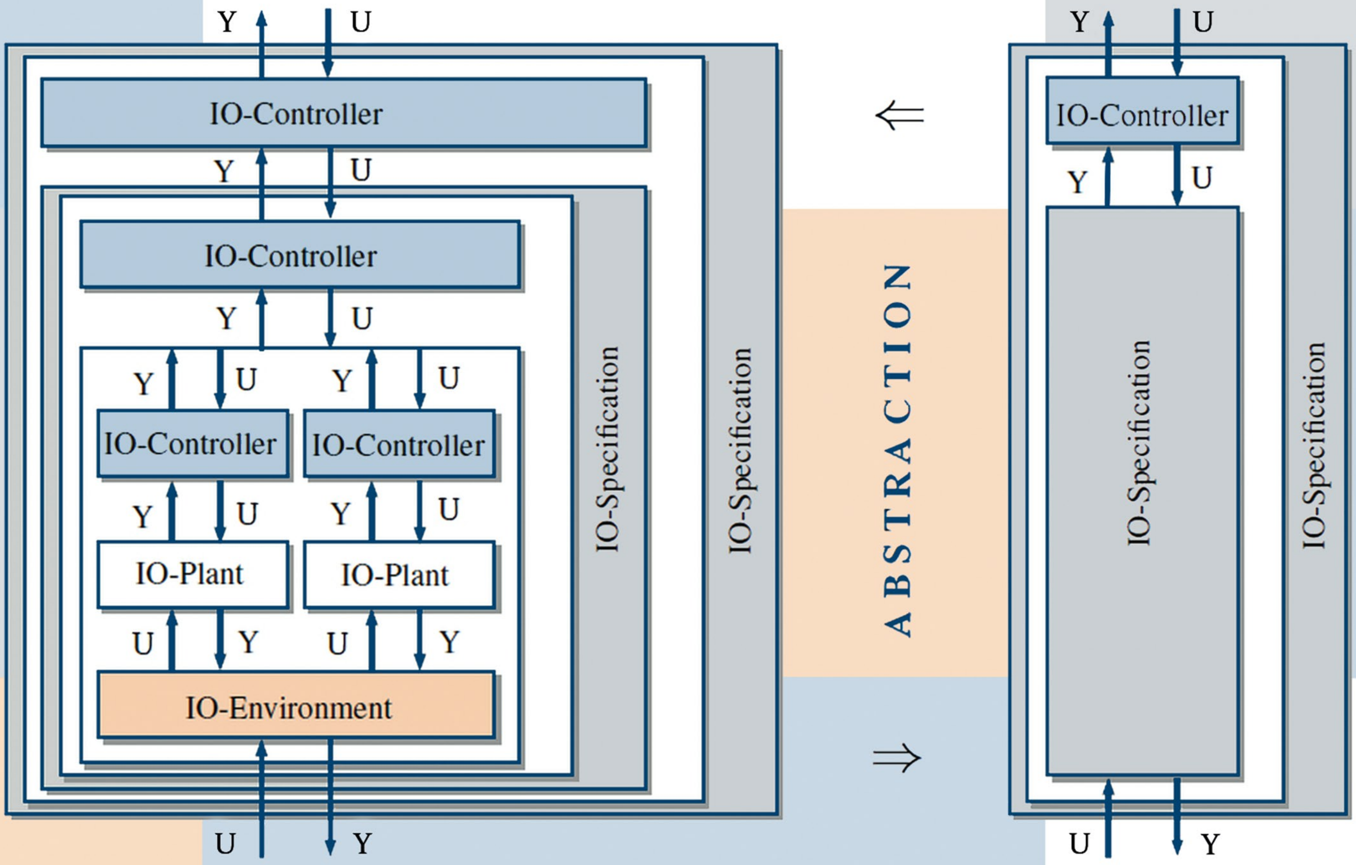


FAU Forschungen, Reihe B, Medizin, Naturwissenschaft, Technik 16

Christine Agethen

A Hierarchical Control Architecture for ω-Languages

FAU UNIVERSITY PRESS

Christine Agethen

A Hierarchical Control Architecture for ω-Languages

FAU Forschungen, Reihe B

Medizin, Naturwissenschaft, Technik

Band 16

Herausgeber der Reihe:
Wissenschaftlicher Beirat der FAU University Press

Christine Agethen

A Hierarchical Control Architecture for ω-Languages

Erlangen
FAU University Press
2017

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Verlag und Auslieferung:

FAU University Press, Universitätsstraße 4, 91054 Erlangen

Druck: docupoint GmbH

ISBN: 978-3-96147-028-0 (Druckausgabe)
eISBN: 978-3-96147-029-7 (Online-Ausgabe)
ISSN: 2198-8102

A Hierarchical Control Architecture for ω-Languages

Eine hierarchische Reglerarchitektur für ω-Sprachen

Der Technischen Fakultät

der Friedrich-Alexander-Universität Erlangen-Nürnberg

zur

Erlangung des Doktorgrades

DOKTOR-INGENIEUR

vorgelegt von

Christine Agethen

geb. Baier

aus Erlangen

Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung: 21.07.2016
Vorsitzender des Promotionsorgans: Prof. Dr. rer. nat. Peter Greil
Gutachter: Prof. Dr.-Ing. Thomas Moor
Prof. Dr.-Ing. Jörg Raisch

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftliche Mitarbeiterin am Lehrstuhl für Regelungstechnik der Friedrich-Alexander-Universität Erlangen-Nürnberg. Den zahlreichen Personen, die während dieser Zeit zum Gelingen meiner Arbeit beigetragen haben, möchte ich an dieser Stelle meinen Dank widmen.

An erster Stelle möchte ich mich besonders bei meinem Doktorvater Herrn Prof. Dr.-Ing. Thomas Moor für die ausgezeichnete Betreuung bedanken. Die tiefgehenden, stets konstruktiven Diskussionen und wertvollen Anregungen haben maßgeblich zum Gelingen der Arbeit beigetragen.

Besonderer Dank gilt auch Herrn Prof. Dr.-Ing. habil. Günter Roppenecker, dem Inhaber des Lehrstuhls für Regelungstechnik, für den stets angenehmen wissenschaftlichen Austausch, und für die Übernahme des Vorsitzes der Prüfungskommission. Ebenfalls herzlich bedanken möchte ich mich bei Herrn Prof. Dr.-Ing. Jörg Raisch für die Übernahme des Zweitgutachtens, sowie bei Herrn Prof. Dr.-Ing. Jörg Franke für das Mitwirken in der Prüfungskommission.

Allen Mitarbeitern des Lehrstuhls für Regelungstechnik möchte ich für die vielfältige Hilfsbereitschaft, die Unterstützung und das Interesse für meine Arbeit danken. Besonderer Dank gilt insbesondere den Kollegen der Forschungsgruppe Ereignisdiskrete Systeme, für die Bereitschaft zu spontanen Diskussionen, für konstruktive Ideen und Hinweise, den wissenschaftlichen Austausch und die vielen erheiternden Momente zwischendurch. Unsere gemeinsame Zeit wird mir in besonderer Erinnerung bleiben.

Abschließend möchte ich vor allem auch meiner Familie herzlich danken, deren beständiger Rückhalt und wertvolle Unterstützung maßgeblich zur Fertigstellung der vorliegenden Dissertation beigetragen haben.

Bamberg, im Juli 2016 Christine Agethen

Abstract

In the *Supervisory Control Theory* according to (Ramadge and Wonham, 1987a) the behaviour of a dynamical system is described by sequences of *discrete events*. Typical applications are production and transportation systems that are operated by programmable logic controllers. A challenge in this area is the controller synthesis for systems consisting of a large number of components involving various processes. The size of a model that describes their synchronous behaviour grows exponentially in the number of components. Hence, supervisor design using Supervisory Control Theory in the context of large-scale systems often requires a significant amount of computational effort and is not advantageous in comparison to conventional methods. In order to avoid large monolithic models, several hierarchical and modular procedures have been presented that structure the monolithic control problem vertically and horizontally into manageable synthesis sub-tasks. However, all these approaches are limited to modelling behaviours on a finite-time horizon, using $*$-languages, see e.g. (Schmidt et al., 2008), (Feng and Wonham, 2008) or (Leduc et al., 2005). Whereas synthesis procedures of the monolithic controller design also include the controller design for behaviours that evolve on an infinite-time axis, so called *ω-languages*, offering a more general field of application.

In this thesis, a hierarchical and modular control approach for ω-languages is developed that is based on the use of abstractions. Known methods regarding the monolithic controller design for ω-languages are adopted to the hierarchical context. Dynamical systems are modelled as *input/output systems*, according to the *Behavioural Systems Theory* introduced in (Willems, 1991), and already applied in the context of hierarchical supervisor design in (Perk et al., 2008). In contrast to (Perk et al., 2008), the procedure presented here is not limited to topologically closed behaviours. Hence, in addition to modelling *safety properties*, it provides the possibility to include also *liveness properties* in the models. Particularly in large-scale systems this possibility of specifying non-terminating processes is essential. For a practical perspective, algorithms to solve the presented hierarchical control problem on the basis of finite-state automata are elaborated and an example from the area of transportation systems is discussed. It demonstrates that the presented approach enables efficient controller design for large-scale systems modelled by ω-languages and reduces the respective computational effort.

Zusammenfassung

In der *Supervisory Control Theory* nach (Ramadge and Wonham, 1987a) wird das dynamische Verhalten des zu regelnde Systems mittels der Abfolge von diskreten Ereignissen beschrieben. Typische Anwendungsgebiete hierfür sind Transport- und Produktionssysteme, die mit Hilfe von speicherprogrammierbaren Steuerungen betrieben werden. Eine große Herausforderung in diesem Bereich ist der Reglerentwurf für Systeme, die aus vielen Komponenten bestehen und verschiedene Prozesse aufweisen. Die Größe des Modells zur Beschreibung des synchronen Verhaltens steigt exponentiell mit der Zahl der Komponenten an. Der Entwurf mittels der Supervisory Control Theory erfordert daher einen hohen Rechenaufwand, sodass dieses Vorgehen im Vergleich zu herkömmlichen Methoden in diesem Fall nicht von Vorteil ist. Um große monolithische Modelle zu vermeiden, wurden diverse modulare und hierarchische Vorgehensweisen vorgestellt, die den Gesamtentwurf in handhabbare Teilentwürfe horizontal und vertikal zerlegen. Diese Ansätze beschränken sich jedoch auf eine Modellbildung des gegebenen Prozesses mittels endlicher Verhalten, siehe z.B. (Schmidt et al., 2008), (Feng and Wonham, 2008) oder (Leduc et al., 2005). Die klassische Supervisory Control Theory umfasst hingegen auch den Reglerentwurf für unendlich fortlaufende Verhalten, die sog. *ω-Sprachen*, und bietet somit ein allgemeineres Anwendungsgebiet.

In der vorliegenden Arbeit werden bestehende Methoden des Reglerentwurfs für unendliche Verhalten aufgegriffen und hinsichtlich eines sowohl modularen, als auch hierarchischen abstraktionsbasierten Reglerentwurfsvorgehens erweitert. Dabei wird auf die besondere Modellierungsart mittels Ein-/Ausgangssysteme der *Behavioural System Theory* nach (Willems, 1991) zurückgegriffen, die auch bereits in (Perk et al., 2008) eingesetzt wurde. Als weitere Besonderheit ist das Verfahren im Gegensatz zum Vorgehen in (Perk et al., 2008) nicht auf topologisch abgeschlossene Verhalten beschränkt. Es eröffnet damit die Möglichkeit neben *Sicherheitseigenschaften* auch *Lebendigkeitseigenschaften* zu modellieren. Besonders im Kontext großer Systeme ist die Möglichkeit unendlich fortlaufende Prozesse zu spezifizieren essentiell. Für die praktische Anwendung des Verfahrens werden Lösungsalgorithmen auf Basis von endlichen Zustandsautomaten entwickelt und ein Anwendungsbeispiel aus dem Bereich der Transportsysteme herangezogen. Es wird dabei gezeigt, dass es mit der vorgestellten Methode möglich ist, einen effizienten Reglerentwurf für große Systeme, die mittels ω-Sprachen modelliert wurden, durchzuführen und den Rechenaufwand dadurch zu verringern.

Contents

1 Introduction

The behaviour of dynamical systems can be modelled in many different ways, depending on the objective behind the respective application. A common way is the use of a time-continuous system description based on differential equations. This approach is often used in order to analyse, as well as to manipulate the time response of a given system. Another way of modelling dynamical systems is the use of *discrete event dynamic systems*, also known as *discrete event systems*. According to (Cassandras and Lafortune, 2008), a discrete event dynamic system can be characterized by two key properties, a discrete set of states and an event-driven state transition mechanism based on a discrete set of events. In particular, such systems evolve over time only according to abrupt and irregular occurrences of asynchronous discrete events; see also (Ramadge and Wonham, 1989).

Typical examples for discrete event dynamic systems are automated transportation and manufacturing systems. In Figure 1.1 a respective laboratory model of a flexible manufacturing system is demonstrated. It consists of various actuators, such as conveyor

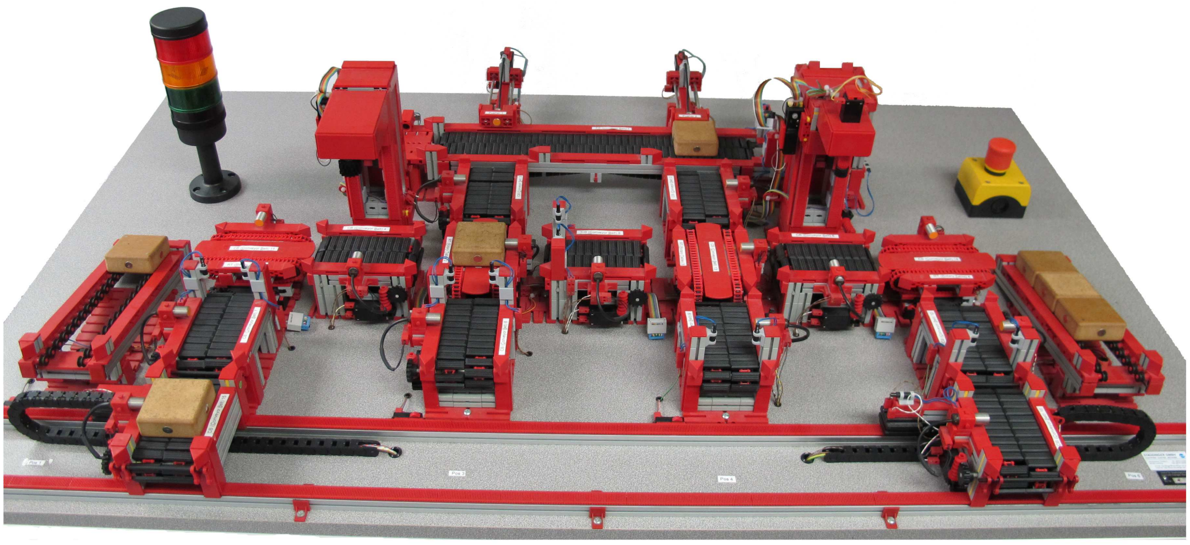


Figure 1.1: Laboratory model of a manufacturing system

belts, rotary tables and pushers, to transport work pieces through the system, and various sensors for work piece detection. In addition, the manufacturing system provides two processing machines to process the work pieces. Details regarding discrete event models of this system are given in (Moor et al., 2010). Discrete event systems are also suitable e.g. for modelling network communication (Schmidt et al., 2007), network protocols (Cieslak et al., 1986), database management systems (Lafortune, 1988), as well as genetic regulatory networks (Baldissera and Cury, 2012).

1.1 Motivation

A common property of all applications is the need for a coordinator in order to ensure a specified succession of discrete events in the given system. An example for such a coordinator, e.g. for a transportation system as in Figure 1.1, is a *programmable logic controller (PLC)*, as shown in Figure 1.2. In the late 80s, the *Supervisory Control Theory* was introduced in (Ramadge and Wonham, 1987a). It proposes a systematic and model-based approach for the controller design for discrete event systems. The basic idea is represented in Figure 1.2.

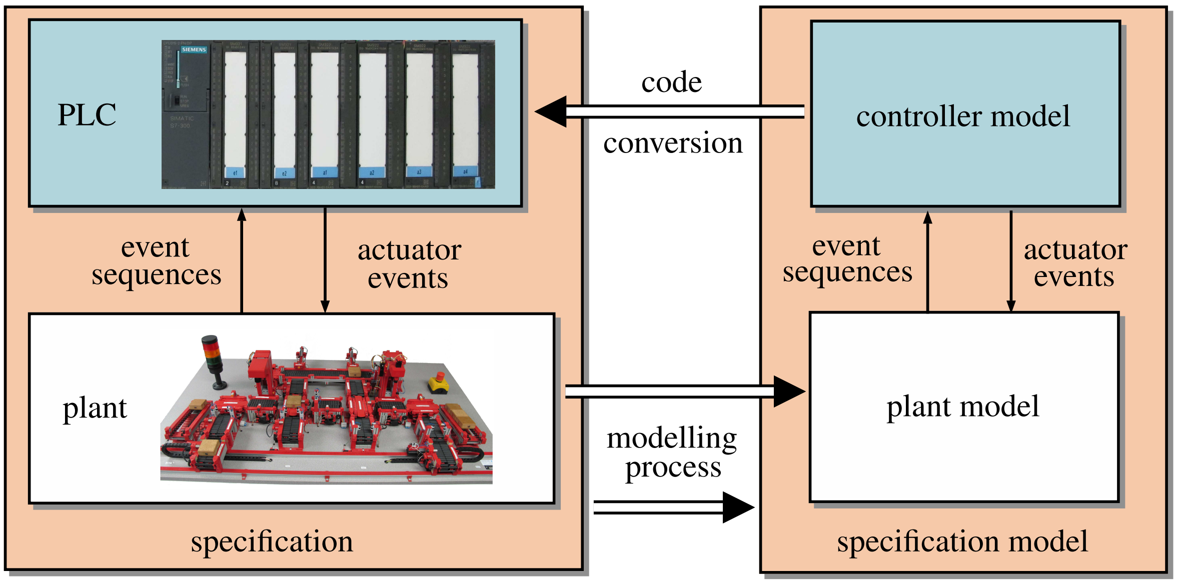


Figure 1.2: Basic concept of Supervisory Control Theory

According to Supervisory Control Theory, the uncontrolled behaviour of a given system (the plant), as well as the specified behaviour (the specification) are represented by particular models. The most common models, already applied in (Ramadge and Wonham, 1989), are *automata on finite words*, representing *finite string languages* also known as $*$*-languages*. A detailed introduction is given in (Hopcroft and Ullman, 1979). Another way to model a system behaviour, which is of particular interest for this thesis, is the use of infinite-string languages, represented by ω-automata, see also (Thomas, 1990) and (Mukund, 1996). The following example shortly demonstrates the modelling process for discrete event systems by automata.

Example 1.1.1

A typical application for modelling with discrete event systems is the conveyor belt in Figure 1.3, which forms part of the laboratory model in Figure 1.1.

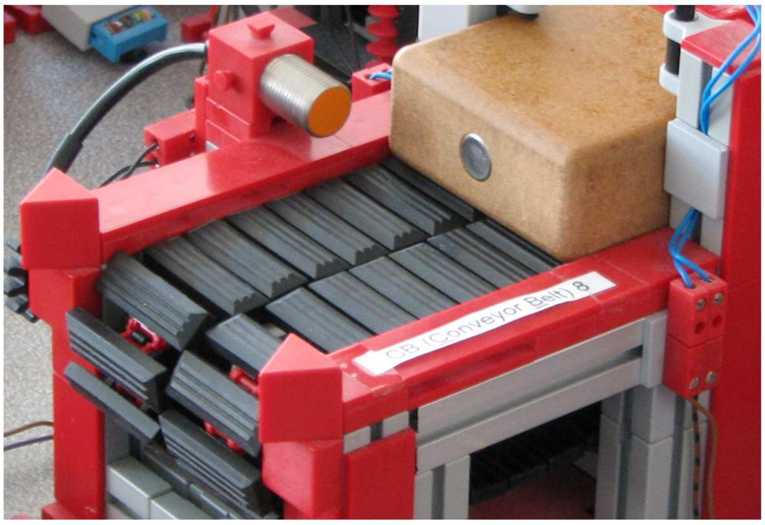


Figure 1.3: Conveyor-belt

In Figure 1.4 it is shown a finite-state automaton that describes the behaviour of the respective conveyor belt. It represents states of the conveyor belt and transitions from one state to the next, generating event sequences. State `1` is the initial state of the system behaviour. Active transitions for this state are either switching the conveyor belt on by `bm+` or off by `boff`. Turning the conveyor belt on causes a transition to state `2`. The conveyor belt contains a sensor in the middle of the belt that detects the arrival of work pieces (`wpar`) as well as the leaving of work pieces (`wplv`). The model describes that detection of work pieces is possible when the conveyor belt is on.

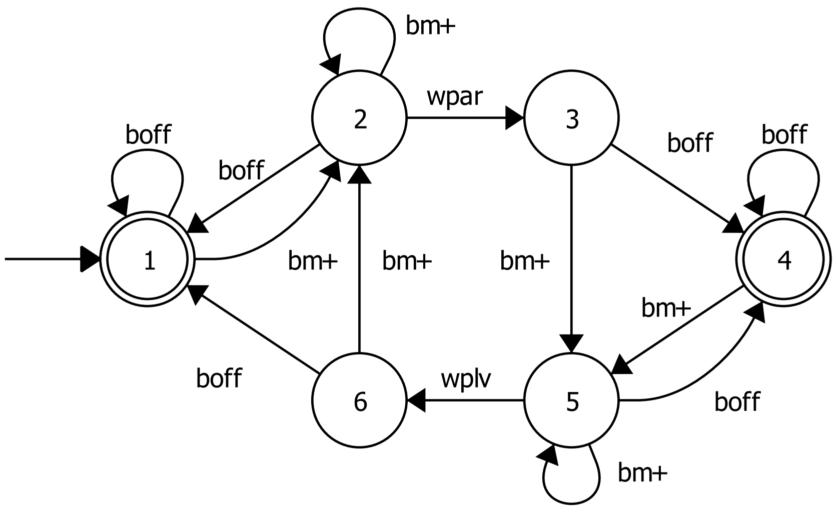


Figure 1.4: Behaviour of a conveyor belt modeled by a finite-state automaton

Depending on the behaviour that shall be described, the automaton can be interpreted as a finite-state automaton. Then, the marked states `1` and `4` accept finite event sequences that end in those states. Further, it can be interpreted as an ω-automaton, that accepts only sequences that pass those states infinitely often. $\square$

The task of the controller is to provide particular actuator events as reaction on event sequences in the plant behaviour and to guarantee thereby the enforcement of a given specification in the closed-loop system. Various algorithms are provided to be applied on respective finite-state automata, which model the behaviour of the plant and the specification, in order to systematically synthesize controller models. The synthesized models can then be implemented on the given PLC that interacts with the real system, e.g. by using code conversion.

1.1.1 Hierarchical and modular control architecture

The main advantage of Supervisory Control Theory itself is the guarantee that a synthesized controller enforces the given specification. Hence, time-consuming test phases are avoided. They would be necessary if an appropriate control is determined in a conventional way, in particular, when using trial-and-error methods. Nevertheless, one of the main challenges of Supervisory Control Theory is the treatment of large-scale systems. Since the number of system states grows exponentially in the number of system components, the respective computational effort for solving the control problem also increases exponentially; see (Gohari and Wonham, 2000) and (Rohloff and Lafortune, 2002) for detailed analysis. Consequently, several approaches were presented that exploit structural system properties in order to avoid large and complex monolithic plant models. Thereby, common engineering practise is used and the global control task is divided into sub-tasks. Modular, decentralized and hierarchical control architectures are designed with the aim to keep the models involved in synthesis procedures as small as possible. In (Ramadge and Wonham, 1987b) and (Wonham and Ramadge, 1988) first ideas regarding modular control were presented. They proposed to divide the control task into several sub-tasks, and design a controller for each sub-task. This idea was further developed in (de Queiroz and Cury, 2000a) and (de Queiroz and Cury, 2000b). Another approach is the hierarchical, abstraction-based controller design introduced by (Wong and Wonham, 1996). The combination of both ideas has been presented e.g. in (Schmidt et al., 2008; Perk et al., 2008; Feng and Wonham, 2008; da Cunha et al., 2002; Leduc et al., 2005). Some of them were proven to be successful in large-scale system applications; see e.g. (Moor et al., 2010) and (Kaess, 2014).

1.1.2 Modelling with ω-languages

In this thesis, the focus lies on the controller design for automata on infinite words that represent *infinite string languages*, also called *ω-languages* or *sequential behaviour*. In (Ramadge, 1989) it was already pointed out that an important advantage of ω-languages compared to $*$-languages is the possibility to model processes on an infinite-time horizon and to be able to distinguish between transient and the non-transient behaviour. Infinite string languages have often been used for modelling asynchronous processes and fair concurrency, e.g. in the area of concurrent computation or communication protocols. In particular, the area of *Temporal Logic* is strongly connected with the representation by ω-automata. Temporal Logics for concurrent systems has been introduced in (Pnueli and Rosner, 1989) in order to specify and verify the correctness of computer programs. Some years later, the development of *Model Checking* techniques, see e.g. (Emerson and Clarke, 1980) and (Clarke et al., 1986), enabled automated synthesis procedures, which are used in the area of *Reactive Synthesis*. The interest in combining the area of Supervisory Control with this area and Model Checking is increasing. Both approaches, the Supervisory Control Theory and the Reactive Synthesis, have been developed mostly independently and have been proven to be successful in industrial applications. Nevertheless, each has its strengths and weaknesses; see (Ziller and Schneider, 2005) and (Ehlers et al., 2014) for a detailed comparison. It seems to be promising to combine them to benefit from both. However, Temporal Logic is strongly connected to a representation by ω-automata; see e.g. (Mukund, 1997) for the strong connection between Linear Temporal Logic and *Büchi automata*. Thus, it is important to further develop supervisor synthesis methods for ω-languages. For monolithic controller design involving ω-languages, some approaches are known. In (Ramadge, 1989) the control problem for sequential behaviour under complete event observation has been discussed. In (Kumar et al., 1992) first results regarding partial observation have been presented. Whereas more general liveness properties in both cases were analysed in (Thistle and Wonham, 1994a) and (Thistle and Lamouchi, 2009). However, further developments regarding hierarchical and modular approaches to overcome the problem of computational effort for large-scale systems are still missing.

1.2 Contribution

The contribution of this thesis is an approach for the design of hierarchical and modular control architectures for ω-languages. To the author's best knowledge, up to now, there does not exist any other hierarchical approach that includes not only topologically closed ω-languages, but also more general liveness properties. In general, the approach is based on a notion of *input/output systems* known from the *Behavioural System Theory* introduced in (Willems, 1991). In (Moor et al., 2003) this behavioural input/output description was used to develop hierarchical control for a class of hybrid systems. In Perk et al. (2008) these ideas regarding hierarchical control for discrete-event systems were further elaborated. However, in (Moor et al., 2003) the focus lies only on topologically closed behaviour. The approach in (Perk et al., 2008) is focused on *prefix-closed* $*$-languages and therefore also only on topologically closed ω-languages. In the approach at hand, this limitation is dropped and a method is proposed that includes more general liveness properties.

The basic idea behind the approach is to design for each individual level of the hierarchy a separate controller according to a language inclusion specification and to use the specification itself as an abstraction of the underlying closed-loop system for the synthesis of the controller one level above. For systems consisting of several components, individual low-level controllers are designed that enforce local control objectives, and a composition of the respective local specifications shall be used as abstraction of the underlying system to design a high-level supervisor. The computational effort is expected to be reduced by alternating abstraction and controller synthesis, since the specification does not need to express how the control objective is achieved. In (Perk et al., 2008), this has been demonstrated for topologically closed languages by a transport system example. Hence, the fundamental question is, whether controller synthesis on the different levels can actually be based on abstractions of the given specifications of the levels below. In (Perk et al., 2008), it was demonstrated that, as long as all relevant languages are considered prefix-closed, the validity of the abstraction-based approach is established by a fairly simple set-inclusion argument. However, it is necessary to impose further constraints to model liveness properties possessed by the plant or required by the specification. In the case of more general liveness properties, this problem results to be more intricate due to the inclusion of liveness properties. Hence, the notion of *non-anticipating input/output systems* is introduced, which was defined originally in (Moor et al., 2010), to achieve a non-conflicting closed-loop configuration in the abstraction-based design. As a consequence, it is not necessary to impose further constraints. Liveness properties can be included naturally by using not topologically closed ω-languages.

1.3 Outline

The outline of the thesis is as follows. In Chapter 2, a short introduction into the theory of formal languages is given. Starting with finite string languages, the basic notation is extended to infinite string languages and it is illustrated how to represent them by graphs consisting of a finite number of states and state transitions. At the end of this chapter, the Supervisory Control Theory for finite string languages is described for partial event observation. It is extended by the property of completeness in the closed-loop system in order to lead over to the control problem for ω-languages. In Chapter 3, a detailed introduction into the control of ω-languages for the case of complete, as well as partial event observation is given. For practical implementations, algorithms to solve the presented control problems are proposed. Finally, a specific control problem based on the input/output description of the Behavioural System Theory is derived. A solution is developed and relevant closed-loop properties are analysed. This control problem forms the basis of the proposed hierarchical and modular approach for ω-languages. In Chapter 4, the ideas of modular and abstraction-based controller design in the context of finite string languages are revealed, in order to extend them to infinite string languages. The control problem of the previous Chapter is used to develop an abstraction-based and modular control architecture for ω-languages. Finally, a transportation system example is used to demonstrate the approach from a practical perspective and to show the potential computational benefit.

2 Preliminaries

In this thesis, the behaviour of a discrete event system, i.e. the sequences of physical events that may occur in a discrete event system, is described by formal finite and infinite string languages. The theory of formal languages is a theoretical framework that is often used in mathematics and computer science. For the representation of formal languages, finite-state automata can be used; see also (Hopcroft and Ullman, 1979) for a detailed introduction into modelling discrete event systems by formal languages. This particular framework has been introduced into the area of system control theory in (Ramadge and Wonham, 1989) and is also referred to as Supervisory Control Theory. Following the ideas of (Ramadge and Wonham, 1989), different approaches have been developed for the controller design of discrete event systems; see also (Cassandras and Lafortune, 2008), (Thistle, 1996) or (Kumar and Garg, 1995).

In this Chapter, it is given a short introduction in the theory of formal languages and the representation by finite-state automata. Further, the controller design according to the Supervisory Control Theory is described for finite string languages. The Chapter is organized as follows. In Section 2.1 basic notation for $*$-languages and for ω-languages is given. Section 2.2 illustrates representation possibilities of formal finite and infinite string languages by finite-state automata. Finally, Section 2.3 gives a general introduction into the theory of the control of discrete event systems modelled by finite string languages.

2.1 Formal Languages

To model the behaviour of a discrete event system, the mathematical framework of formal languages is used. Therein, it can be differed between processes with *finite-time behaviour* and finite termination, modelled by finite string languages, also called $*$-languages, in contrast to *infinite-time behaviours*, also called *sequential behaviours* or *ω-languages*. In the following, the two different kinds of languages are introduced, relevant properties are discussed and it is described how use languages to model processes that evolve either on a finite or on a infinite time axis.

2.1.1 Basic notation for $*$-languages

A set of physical events that occur in a discrete event system is given in form of an *alphabet* Σ. A finite sequence of events $s = \sigma_1\sigma_2...\sigma_n$, with $n \in \mathbb{N}$, is denoted a *finite string*. Further, the *Kleene-closure* Σ^* is defined as a set of finite strings $s = \sigma_1\sigma_2 \cdots \sigma_n$, $n \in \mathbb{N}$, with $\sigma_n \in \Sigma$, i.e. the set of all finite strings that can be formed by the events given in the alphabet Σ, including the *empty string* $\varepsilon \in \Sigma^*$, $\varepsilon \notin \Sigma$. The length of a string $s \in \Sigma^*$ is given by $|s| \in \mathbb{N}$. A $*$*-language* over Σ is given as a set of strings $L \subseteq \Sigma^*$. Hence, a language is a selection of strings which are possible for the given alphabet Σ.

General $*$-language properties

A language $L \subseteq \Sigma^*$ is *bounded* if $\sup\{|s| \mid s \in L\} < \infty$, or, else, the language L is denoted as *unbounded*. If for two strings $s, r \in \Sigma^*$ there exists $t \in \Sigma^*$ such that $s = rt$, it is said that r is a *prefix* of s, denoted by $r \leq s$. If, in addition, $r \neq s$, then, r is a *strict prefix* of s and it is written $r < s$. Further, the prefix of the string s with length $n \in \mathbb{N}$, $n \leq |s|$, is denoted $s^{(n)}$.

Definition 2.1.1 Prefix-closure

The *prefix-closure* (or short *closure*) of $L \subseteq \Sigma^*$ is defined by

$$\mathrm{pre}L := \{r \subseteq \Sigma^* \mid \exists s \in L : r \leq s\}. \qquad \square$$

A language $L \subseteq \Sigma^*$ is called *prefix-closed* or short *closed*, if $L = \mathrm{pre}L$, i.e. for every string contained in L, its prefixes are also contained in L.

Definition 2.1.2 Relative closedness

Given $L,\ K \subseteq \Sigma^*$, K is said to be *relatively closed w.r.t. L* if

$$K = (\mathrm{pre}K) \cap L. \qquad \square$$

If a language K is relatively closed w.r.t. a closed language, then K itself is closed. The closure operator distributes over arbitrary unions of languages. However, for the intersection of two languages $L,\ K \subseteq \Sigma^*$, it is $\mathrm{pre}(L \cap K) \subseteq (\mathrm{pre}L) \cap (\mathrm{pre}K)$, and, if equality holds, L and K are said to be *non-conflicting*. Note that this is trivially the case for $K \subseteq L$.

Another useful property, especially in the context of infinite behaviours, is the *completeness* [1].

[1] Note that this definition should not be confused with the definition of *completeness* of the Behavioural System Theory, defined e.g. in (Willems, 1991).

Definition 2.1.3 Completeness

A language $K \subseteq \Sigma^*$ is said to be *complete*, if for all $s \in \text{pre}\,K$ there exists $\sigma \in \Sigma$ such that $s\sigma \in \text{pre}\,K$. □

This property was discussed in the context of infinite behaviours e.g. in (Kumar et al., 1992). Note further that completeness is preserved under arbitrary unions of languages.

Properties regarding event observation

The physical events of a discrete event system differ regarding their observability. It is distinguished between *observable events* Σ_o and *unobservable events* Σ_uo. Hence, it is also differed between *systems under complete observation*, if only observable events are involved, and *systems under partial observation*, if also unobservable events occur in the given system. In order to hide e.g. unobservable events, a useful operator is introduced.

Definition 2.1.4 Natural projection

The *natural projection* $\text{p}_\text{o} : \Sigma^* \to \Sigma_\text{o}^*$, $\Sigma_\text{o} \subseteq \Sigma$, is defined iteratively:

$$\text{p}_\text{o}\varepsilon := \varepsilon,$$

$$\text{p}_\text{o}(s\sigma) := \begin{cases} (\text{p}_\text{o}s)\,\sigma & \text{for } s \in \Sigma^*, \sigma \in \Sigma_\text{o} \\ \text{p}_\text{o}s & \text{for } s \in \Sigma^*, \sigma \notin \Sigma_\text{o} \end{cases}$$ □

Vice versa, the set-valued inverse intersperses events from Σ_uo in strings defined on Σ_o.

Definition 2.1.5 Inverse projection

The set-valued inverse p_o^{-1} is defined by $\text{p}_\text{o}^{-1}(r) := \{s \in \Sigma^* \mid \text{p}_\text{o}(s) = r\}$ for $r \in \Sigma_\text{o}^*$. □

When applied to languages, the projection distributes over arbitrary unions, and the inverse projection distributes over arbitrary unions and arbitrary intersections. Furthermore, the closure operator commutes with projection and inverse projection.

Equivalence relations

According to (Hopcroft and Ullman, 1979), an arbitrary relation R that is *reflexive, transitive and symmetric* is called an *equivalence relation*. For an arbitrary set M, and two elements $s,t \in M$, it is written $s \equiv_\text{R} t$, for sRt. By $[s]_R := \{t \in M \mid t \equiv_\text{R} s\}$, the respective equivalence class containing $s \in M$ is denoted. A fundamental characteristic of the equivalence relation is the division of the state space in disjoint, non-empty equivalence classes. Note that, in general, the number of equivalence classes may be infinite, but in the case of regular languages, the number is finite. A particular kind of equivalence relation is the *Nerode equivalence*. It is a *right congruence*, i.e. $s \equiv_{\mathrm{R}} t$ implies $sz \equiv_{\mathrm{R}} tz$ for $z \in M$.

Definition 2.1.6 Nerode equivalence

The *Nerode equivalence* w.r.t. a language $L \subseteq \Sigma^*$ is an equivalence relation on Σ^* and defined by

$$s' \equiv_{\mathrm{L}} s' \quad \text{if for all} \quad t \in \Sigma^* \quad \text{it is} \quad s't \in L \Leftrightarrow s''t \in L. \qquad \square$$

According to this definition, two strings are equivalent, if they are equally extensible in L. Therefore, the Nerode equivalence is a useful medium for the characterization of the state space of finite automata. A well-known application is the characterization of the minimal state space under language conservation; see (Hopcroft and Ullman, 1979).

2.1.2 Basic notation for ω-languages

In contrast to the finite string setting, an *infinite string* over Σ is defined as a map $w : \mathbb{N} \to \Sigma$. The set of all infinite strings over Σ is given by $\Sigma^\omega := \{w \mid w : \mathbb{N} \to \Sigma\}$. Hence, an *$\omega$-language* is a subset $\mathcal{L} \subseteq \Sigma^\omega$ of all infinite strings over Σ. Note that, throughout this thesis, ω-languages are denoted by calligraphic letters. For a general discussion of ω-languages, refer to (Mukund, 1996; Thomas, 1990).

General ω-language properties

Given $w \in \Sigma^\omega$, the *prefix with length* $n \in \mathbb{N}$ is denoted $w^{(n)} \in \Sigma^*$ and it is written $s < w$ for a prefix $s \in \Sigma^*$ of w.

Definition 2.1.7 Prefix

The *prefix* of a language $\mathcal{L} \subseteq \Sigma^\omega$ is defined by $\operatorname{pre}\mathcal{L} := \{s \in \Sigma^* \mid \exists w \in \mathcal{L} : s < w\}$. $\square$

The prefix of any ω-language is a complete prefix-closed $*$-language and the prefix operator distributes over arbitrary unions of ω-languages. However, for the intersection of two ω-languages $\mathcal{L}, \mathcal{K} \subseteq \Sigma^\omega$, it is $\operatorname{pre}(\mathcal{L} \cap \mathcal{K}) \subseteq (\operatorname{pre}\mathcal{L}) \cap (\operatorname{pre}\mathcal{K})$, and, if equality holds, the languages are said to be *non-conflicting*. The languages $\mathcal{L}$, $\mathcal{K} \subseteq \Sigma^\omega$ are *locally non-conflicting* if $(\operatorname{pre}\mathcal{L}) \cap (\operatorname{pre}\mathcal{K})$ is complete. If two languages are non-conflicting, they are also locally non-conflicting. Note, that for $\mathcal{K} \subseteq \mathcal{L}$ both conditions are trivially satisfied. Another important language operator is the *quotient* operator, sometimes also denote the *right quotient*.

Definition 2.1.8 Quotient

For any $L \subseteq \Sigma^*$ and any $\mathcal{K} \subseteq \Sigma^* \cup \Sigma^\omega$, the *quotient* is defined by

$$\mathcal{K}/L := \{w \in \Sigma^* \cup \Sigma^\omega \mid \exists s \in L : sw \in \mathcal{K}\}. \qquad \square$$

It can be seen from the definition, that the quotient extracts all continuations of all strings contained in a given finite language in another finite or infinite language.

Definition 2.1.9 Active event set

For any language $\mathcal{L} \subseteq \Sigma^* \cup \Sigma^\omega$ and any string $s \in \operatorname{pre}\mathcal{L}$, the *active event set* of $\mathcal{L}$ after s is given by $\operatorname{elig}_{\mathcal{L}}(s) := \Sigma \cap (\operatorname{pre}\mathcal{L}/s)$. $\square$

A *monotone sequence of strings*, denoted by $(s_n) \subseteq \Sigma^*$, is a sequence $(s_n)_{n\in\mathbb{N}}$, $s_n \in \Sigma^*$, $s_n \leq s_{n+1}$ for all $n \in \mathbb{N}$. The sequence (s_n) is *bounded* if $\sup\{|s_n| \mid n \in \mathbb{N}\} < \infty$, or else, (s_n) is *unbounded*. The point-wise *limit* of a monotone sequence (s_n) is denoted by $\lim(s_n) \in \Sigma^* \cup \Sigma^\omega$. For a language $L \subseteq \Sigma^*$, the *limit* is defined by $\lim L := \{\lim(s_n) \mid (s_n) \subseteq L\} \cap \Sigma^\omega$. Note that, in general, $\operatorname{pre}\lim L \subseteq L$, if L is prefix-closed. In particular $\operatorname{pre}\lim L = L$, if and only if L is complete and prefix-closed; see (Kumar et al., 1992). Hence, $\operatorname{pre}\lim\operatorname{pre}\mathcal{L} = \operatorname{pre}\mathcal{L}$. Further, $\operatorname{pre}L = \operatorname{pre}\lim L$, if L is complete. According to (Ramadge, 1989), for a family of $*$-languages $L_a, a \in A$, it is $\cup_{a\in A} \lim L_a \subseteq \lim(\cup_{a\in A} L_a)$. Equality is given, in case of a finite union. Further, it is $\cap_{a\in A} \lim L_a \supseteq \lim(\cap_{a\in A} L_a)$. For the intersection of two $*$-languages L, $K \subseteq \Sigma^*$, with $K = \operatorname{pre}K$, it is $\lim(L \cap K) = (\lim L) \cap (\lim K)$; see also (Baier and Moor, 2012), Lemma 1.

Definition 2.1.10 Topological closure

The *topological closure* (or short *closure*) of an ω-language $\mathcal{L} \subseteq \Sigma^\omega$ is defined by

$$\operatorname{clo}\mathcal{L} := \lim\operatorname{pre}\mathcal{L}. \qquad \square$$

An ω-language is said to be *topologically closed* (or short *closed*) if $\operatorname{clo}\mathcal{L} = \mathcal{L}$. The limit of a prefix-closed $*$-language is topologically closed. Further $\mathcal{L} \subseteq \operatorname{clo}\mathcal{L}$; see (Ramadge, 1989).

Definition 2.1.11 Relatively closedness

Given two ω-languages $\mathcal{L}$, $\mathcal{K} \subseteq \Sigma^\omega$, $\mathcal{K}$ is said to be *relatively closed w.r.t.* $\mathcal{L}$, if

$$\mathcal{K} = (\operatorname{clo}\mathcal{K}) \cap \mathcal{L}. \qquad \square$$

The closure operator distributes over finite unions of ω-languages, see e.g. (Ramadge, 1989).

Properties regarding event observation

Dealing with partial event observation requires respective operators for ω-languages.

Definition 2.1.12 Natural projection of infinite strings

For the *natural projection of infinite strings*, let $w \in \Sigma^\omega$, denote $(s_n) \subseteq \Sigma^*$ an unbounded monotone sequence of prefixes of w, and define

$$\mathrm{p}_\mathrm{o}^\omega w := \lim(\mathrm{p}_\mathrm{o} s_n) \in \Sigma_\mathrm{o}^* \cup \Sigma_\mathrm{o}^\omega \qquad \square$$

This definition is based on a definition that was elaborated in the context ω-languages in (Kumar et al., 1992). The definition of the set-valued inverse is straight forward.

Definition 2.1.13 Inverse projection of infinite strings

The set-valued inverse for $v \in \Sigma_\mathrm{o}^* \cup \Sigma_\mathrm{o}^\omega$ is given by

$$\mathrm{p}_\mathrm{o}^{-\omega}(v) := \{\, w \in \Sigma^\omega \mid \mathrm{p}_\mathrm{o}^\omega(w) = v\}. \qquad \square$$

When applying the projection to languages, it is obtained

$$\mathrm{p}_\mathrm{o}^\omega \mathcal{L} = \{\mathrm{p}_\mathrm{o}^\omega w \,|\, w \in \mathcal{L}\} \subseteq \Sigma_\mathrm{o}^* \cup \Sigma_\mathrm{o}^\omega \quad \text{for} \quad \mathcal{L} \subseteq \Sigma^\omega, \text{ and}$$
$$\mathrm{p}_\mathrm{o}^{-\omega} \mathcal{L}_\mathrm{o} = \{w \in \Sigma^\omega \,|\, \mathrm{p}_\mathrm{o}^\omega w \in \mathcal{L}_\mathrm{o}\} \quad \text{for} \quad \mathcal{L}_\mathrm{o} \subseteq \Sigma_\mathrm{o}^* \cup \Sigma_\mathrm{o}^\omega.$$

Here, the projection distributes over arbitrary unions, the inverse projection over arbitrary unions and arbitrary intersections. Further, both commute with the prefix operator. Note also that $\mathrm{p}_\mathrm{o}^{-\omega}\mathrm{p}_\mathrm{o}^\omega \mathcal{L} = \{w \in \Sigma^\omega \,|\, \exists w' \in \mathcal{L} : \mathrm{p}_\mathrm{o}^\omega w = \mathrm{p}_\mathrm{o}^\omega w'\}$. In (Baier and Moor, 2012), Lemma 2, the relationship between limit and projection has been further analysed.

Lemma 2.1.1 (Baier and Moor, 2012)

Given Σ, $\Sigma_\mathrm{o} \subseteq \Sigma$, and $L = \mathrm{pre}\, L \subseteq \Sigma^*$, $L_\mathrm{o} \subseteq \Sigma_\mathrm{o}^*$, $\mathcal{L}_\mathrm{o} \subseteq \Sigma_\mathrm{o}^\omega$, then

(i) $(\mathrm{p}_\mathrm{o}^\omega \lim L) \cap \Sigma_\mathrm{o}^\omega = \lim \mathrm{p}_\mathrm{o} L$,

(ii) $\mathrm{p}_\mathrm{o}^{-\omega} \lim L_\mathrm{o} = (\lim \mathrm{p}_\mathrm{o}^{-1} L_\mathrm{o}) \cap (\mathrm{p}_\mathrm{o}^{-\omega} \Sigma_\mathrm{o}^\omega)$,

(iii) $\mathrm{clo}\, \mathrm{p}_\mathrm{o}^{-\omega} \mathcal{L}_\mathrm{o} = (\mathrm{p}_\mathrm{o}^{-\omega}\, \mathrm{clo}\, \mathcal{L}_\mathrm{o}) \cup (\mathrm{p}_\mathrm{o}^{-\omega}\, \mathrm{pre}\, \mathcal{L}_\mathrm{o})$. $\square$

2.2 Representation of formal languages

For the purpose of illustration of practical applications and the computation of solutions to the control problem, this subsection introduces finite representations for the given finite and infinite string languages. A multitude of representations can be found in the literature; see also (Hopcroft and Ullman, 1979; Thomas, 1990; Mukund, 1996). In the context of this thesis, $*$-languages are represented by *finite-state automata* and ω-languages by *Büchi automata*. Some relevant operations on the representations are illustrated which are applied in the reminder of this thesis. Note that the main results developed in the subsequent Chapters do not rely on a particular form of representation. Nevertheless, when referring to practical applications and computational procedures in this thesis, the attention is focused pragmatically on finite-state and Büchi automata. The motivation therefore is the use of the open-source C++-library libFAUDES, which is based on finite-state and Büchi automata, see also (libFAUDES, 2015).

2.2.1 Representation of $*$-languages

A $*$-language is represented by a standard finite-state automaton; see e.g. (Hopcroft and Ullman, 1979).

Definition 2.2.1 Finite-state automaton

A *finite-state automaton* is given as a five-tuple

$$G = (Q, \Sigma, \delta, Q_\mathrm{o}, Q_\mathrm{m}),$$

where Q is the set of *states*, Σ is the alphabet of events, $\delta : Q \times \Sigma \to 2^Q$ is a *transition function*, Q_o the *set of initial states* and Q_m the set of *marked states*. □

If the transition function is given as $\delta : Q \times \Sigma \to Q$ and the initial states consist of a single initial state q_o, then it is called *deterministic*. A finite $*$-language *generated* by G is given by

$$L(G) = \{s \in \Sigma^* \mid \delta(q_\mathrm{o}, s)!\},$$

where it is written $\delta(q_\mathrm{o}, s)!$, if $\delta(q_\mathrm{o}, s)$ is defined, i.e. $\delta(q_\mathrm{o}, s) \neq \emptyset$. Note that the above definition uses the *extended transition function* $\delta : Q \times \Sigma^* \to Q$; see e.g. (Hopcroft and Ullman, 1979). The extension is denoted by the same symbol δ. A finite $*$-language that is *marked* by G is defined by

$$L_\mathrm{m}(G) = \{s \in \Sigma^* \mid \delta(q_\mathrm{o}, s) \in Q_\mathrm{m} \neq \emptyset\}.$$

It represents all strings that correspond to paths ending in a marked state. Note that a $*$-language $L \subseteq \Sigma^*$ is *regular*, if it is marked by a finite-state automaton; see e.g. (Hopcroft and Ullman, 1979).

Operations on $*$-languages relevant to this thesis retain regularity, with known algorithms to obtain the respective finite state realisation.

Example 2.2.1

To illustrate the representation of $*$-languages, a trivial example is provided in Figure 2.1.

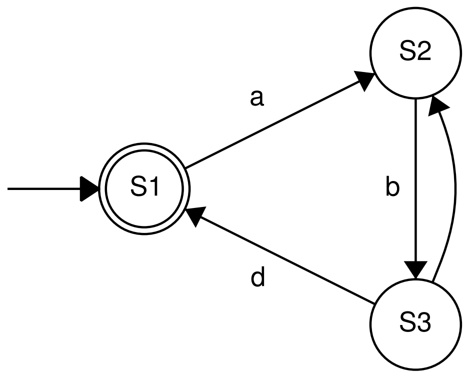


Figure 2.1: Example for the representation of languages by a finite-state automaton

The automaton consists of the state set $Q = \{\texttt{S1},\texttt{S2},\texttt{S3}\}$, the event set $\Sigma = \{\texttt{a},\texttt{b},\texttt{c},\texttt{d}\}$, the initial state $q_o = \texttt{S1}$ and the marked state $Q_m = \{\texttt{S1}\}$. The $*$-language generated by the given automaton is $L(G) = \text{pre}((ab(cb)^*d)^*)$, whereas the marked language is $L_m(G) = (ab(cb)^*d)^*$. Note that all valid strings in the marked language end with the event `d`, since the transition labelled by this event is the only one that leads to a marked state. Observe that the given automaton is deterministic. □

2.2.2 Representation of ω-languages

The basic structure of the finite-state automaton is also used for the representation of ω-languages. The only difference to the finite-state automaton used for the representation of $*$-languages lies in the language acceptance criterion. A variety of acceptance criteria are given in the literature. For the scope of this thesis, the focus lies on the Büchi and the Rabin acceptance conditions. For further details regarding other representations of ω-languages, see also (Mukund, 1996) or (Thomas, 1990).

Definition 2.2.2 Büchi automaton

A *Büchi automaton* is given as a five-tuple

$$G = (Q, \Sigma, \delta, Q_o, Q_m),$$

where Q is the set of *states*, Σ is the alphabet of events, $\delta : Q \times \Sigma \to 2^Q$ is a *transition function*, Q_o the *set of initial states* and Q_m the set of *marked states*. □

If the transition function is given as $\delta : Q \times \Sigma \rightarrow Q$ and the initial states consist of a single initial state q_o, then it is called *deterministic*. An ω-language that is *marked* by G is defined by

$$\mathcal{L}(G) := \{w \in \Sigma^\omega |\ \mathrm{Inf}(w) \cap Q_\mathrm{m} \neq \emptyset\},$$

with

$$\mathrm{Inf}(w) := \{q \in Q \mid \exists \text{ infinitely many } s < w \text{ s.th. } \delta(q_o, s) = q\}.$$

It represents all strings that correspond to paths that infinitely often pass marked states.

Example 2.2.2

The finite-state automaton in Figure 2.1 is considered and the represented ω-language is analysed. Since the automaton is deterministic, the ω-language is given as the limit of the $*$-language, i.e. $\mathcal{L}(G) = \lim(L_\mathrm{m}(G)) = \lim((ab(cb)^*d)^*) = (ab(cb)^*d)^\omega$. □

Note that $\mathcal{L} \subseteq \Sigma^\omega$ is *ω-regular*, if it is accepted by a Büchi automaton; see also (McNaughton, 1966). Moreover, if an ω-language can be represented as the limit of a regular $*$-language, it is accepted by a deterministic Büchi automaton. It is a well-known fact that there are Büchi recognizable languages that can be represented by a non-deterministic Büchi automaton, but it is not possible to represent them by a deterministic Büchi automaton; see e.g. (Mukund, 1996) for further details. Hence, non-deterministic Büchi automata languages are strictly more powerful than the deterministic ones. The deterministic *Rabin automaton* is an alternative model that recognizes all ω-regular languages.

Definition 2.2.3 Rabin automaton

A *Rabin automaton* is given as a five-tuple

$$G = (Q, \Sigma, \delta, Q_\mathrm{o}, \Omega),$$

where Q is the set of *states*, Σ is the alphabet of events, $\delta : Q \times \Sigma \rightarrow 2^Q$ is a *transition function*, Q_o the *set of initial states* and $\Omega = \{(\mathcal{F}_1, \mathcal{I}_1), ..., (\mathcal{F}_\mathrm{n}, \mathcal{I}_\mathrm{n})\}$ is a collection of *accepting pairs* $(\mathcal{F}_\mathrm{i}, \mathcal{I}_\mathrm{i})$, where $\mathcal{F}_\mathrm{i}, \mathcal{I}_\mathrm{i} \subseteq Q$. □

An ω-language that is *marked* by a Rabin automaton G is defined by

$$\mathcal{L}(G) := \{w \in \Sigma^\omega |\ \mathrm{Inf}(w) \cap \mathcal{F}_\mathrm{i} = \emptyset \text{ and } \mathrm{Inf}(w) \cap \mathcal{I}_\mathrm{i} \neq \emptyset \text{ for some } i \in 1, ..., n\}.$$

This language represents all strings that correspond to paths that for some $i \in 1, ..., n$ infinitely often pass states in $\mathcal{I}_\mathrm{i}$ and only finitely often passes states in $\mathcal{F}_\mathrm{i}$. Note that a Büchi automaton can be trivially transformed into a Rabin automaton by setting $\Omega = \{(\emptyset, Q_\mathrm{m})\}$.

2.2.3 Constructions on finite-state automata

In the sequel of this thesis, several constructions on finite-state automata are used to characterize effective solutions. Hence, it is worth presenting some of the most frequently used and interesting constructions at this point. The first operation, that shall be explained here, is the *projection operation*. It is based on a construction given in (Cho and Marcus, 1989a) and similar to the one in (Cassandras and Lafortune, 2008).

Definition 2.2.4 Projection operation

Given a finite-state automaton $G = (Q, \Sigma, \delta, q_\mathrm{o}, Q_\mathrm{m})$, and the projection $\mathrm{p_o} : \Sigma^* \to \Sigma_\mathrm{o}^*$, with $\Sigma_\mathrm{o} \subseteq \Sigma$. Let the projection $\mathrm{p_o}G := \mathrm{ac}(X, \Sigma_\mathrm{o}, \xi, x_\mathrm{o}, X_\mathrm{m})$ be defined as follows:

$$X := 2^\mathrm{Q} - \emptyset,\ x_\mathrm{o} := \{q \in \delta(q_\mathrm{o}, s) \mid \mathrm{p_o}s = \varepsilon\},$$
$$\xi(x, \sigma) := \{q \in \delta(q', s) \mid q' \in x \text{ and } \mathrm{p_o}s = \sigma\},$$
$$X_\mathrm{m} := \{x \in X \mid \exists q \in x \text{ s.th. } q \in Q_\mathrm{m}\}. \qquad \square$$

The operation ac is used to remove not accessible states; see also (Cassandras and Lafortune, 2008).

For the synchronization of two languages defined on a shared alphabet the *product operation* and for different alphabets the *synchronous composition operation* is used; see also (Cassandras and Lafortune, 2008).

Definition 2.2.5 Product operation

Given two automata $G_1 = (Q_1, \Sigma_1, \delta_1, q_{\mathrm{o},1}, Q_{\mathrm{m},1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, , q_{\mathrm{o},2}, Q_{\mathrm{m},2})$. The automaton of the product $G_1 \times G_2 := \mathrm{ac}(X, \Sigma, \xi, x_\mathrm{o}, X_\mathrm{m})$ is defined by

$$X := Q_1 \times Q_2,\ \Sigma := \Sigma_1 \cap \Sigma_2,\ x_\mathrm{o} := (q_{\mathrm{o},1}, q_{\mathrm{o},2}),$$
$$\xi((q_1, q_2), \sigma) := \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)), & \text{if } \delta_1(q_1, \sigma)! \text{ and } \delta_2(q_2, \sigma)!, \\ \text{undefined} & \text{otherwise,} \end{cases}$$
$$X_\mathrm{m} := Q_{\mathrm{m},1} \times Q_{\mathrm{m},2}. \qquad \square$$

For the languages $L_1, L_2 \subseteq \Sigma^*$, represented by G_1 and G_2, the product is given by $L_1 \times L_2 := L_1 \cap L_2$.

Definition 2.2.6 Synchronous composition operation

Given two automata $G_1 = (Q_1, \Sigma_1, \delta_1, q_{\mathrm{o},1}, Q_{\mathrm{m},1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, , q_{\mathrm{o},2}, Q_{\mathrm{m},2})$. The automaton of the synchronous composition $G_1 \parallel G_2 := \mathrm{ac}(X, \Sigma, \xi, x_{\mathrm{o}}, X_{\mathrm{m}})$ is defined as

$$X := Q_1 \times Q_2, \Sigma := \Sigma_1 \cup \Sigma_2, x_{\mathrm{o}} := (q_{\mathrm{o},1}, q_{\mathrm{o},2}),$$

$$\xi((q_1,q_2),\sigma) := \begin{cases} (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma)), & \text{if } \delta_1(q_1,\sigma)! \text{ and } \delta_2(q_2,\sigma)!, \\ (\delta_1(q_1,\sigma), q_2), & \text{if } \delta_1(q_1,\sigma)! \text{ and } \sigma \notin \Sigma_2, \\ (q_1, \delta_2(q_2,\sigma)), & \text{if } \delta_2(q_2,\sigma)! \text{ and } \sigma \notin \Sigma_1, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$X_{\mathrm{m}} := Q_{\mathrm{m},1} \times Q_{\mathrm{m},2}. \qquad \square$$

For the languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, represented by G_1 and G_2, the synchronous composition is given by $L_1 \parallel L_2 := (\mathrm{p}_1^{-1} L_1) \cap (\mathrm{p}_2^{-1} L_2)$. Note that for $\Sigma_1 \cap \Sigma_2 = \emptyset$, the synchronous composition is a so called *shuffle product*; see also (Cassandras and Lafortune, 2008).

The product construction of two Büchi automata is more involved, since the passing of the marking of each of the two automata has to be tagged in order to guarantee that the marking of the resulting automaton accepts only strings for which both original automata pass infinitely often a marked state. The basic idea is to add a marked state to the resulting automaton only after alternated passing marked states in both original automata. Hence, an additional tag is used in order to remember the marking that is waited for. The following definition is based on (Mukund, 1996).

Definition 2.2.7 ω-product operation

Given two automata $G_1 = (Q_1, \Sigma_1, \delta_1, q_{\mathrm{o},1}, Q_{\mathrm{m},1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, , q_{\mathrm{o},2}, Q_{\mathrm{m},2})$. The automaton of the ω-product $G_1 \times_\omega G_2 := \mathrm{ac}(X, \Sigma, \xi, x_{\mathrm{o}}, X_{\mathrm{m}})$ is defined as

$$X := Q_1 \times Q_2 \times \{1,2\}, \Sigma := \Sigma_1 \cap \Sigma_2, x_{\mathrm{o}} := (q_{\mathrm{o},1}, q_{\mathrm{o},2}, 1),$$

$$\xi((q_1,q_2,1),\sigma) := \begin{cases} (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma), 1), & \text{if } \delta_1(q_1,\sigma)!,\ \delta_2(q_2,\sigma)! \text{ and } q_1 \notin Q_{\mathrm{m},1}, \\ (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma), 2), & \text{if } \delta_1(q_1,\sigma)!,\ \delta_2(q_2,\sigma)! \text{ and } q_1 \in Q_{\mathrm{m},1}, \end{cases}$$

$$\xi((q_1,q_2,2),\sigma) := \begin{cases} (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma), 2), & \text{if } \delta_1(q_1,\sigma)!,\ \delta_2(q_2,\sigma)! \text{ and } q_2 \notin Q_{\mathrm{m},2}, \\ (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma), 1), & \text{if } \delta_1(q_1,\sigma)!,\ \delta_2(q_2,\sigma)! \text{ and } q_2 \in Q_{\mathrm{m},2}, \end{cases}$$

$$X_{\mathrm{m}} := Q_1 \times Q_{\mathrm{m},2} \times 2. \qquad \square$$

The ω-product of two languages $\mathcal{L}_1 \subseteq \Sigma_1^\omega$ and $\mathcal{L}_2 \subseteq \Sigma_2^\omega$, represented by G_1 and G_2, is given by $\mathcal{L}_1 \times_\omega \mathcal{L}_2 := \mathcal{L}_1 \cap \mathcal{L}_2$.

The construction of the *synchronous ω-composition operation* is straight forward and based on the construction in Definition 2.2.6 and the considerations regarding the particular marking in Definition 2.2.7.

Definition 2.2.8 Synchronous ω-composition operation

Given $G_1 = (Q_1, \Sigma_1, \delta_1, q_{o,1}, Q_{m,1})$, $G_2 = (Q_2, \Sigma_2, \delta_2, , q_{o,2}, Q_{m,2})$. The automaton of the synchronous ω-composition $G_1 \parallel_\omega G_2 := \mathrm{ac}(X, \Sigma, \xi, x_o, X_m)$ is defined by

$$X := Q_1 \times Q_2 \times \{1,2\}, \Sigma := \Sigma_1 \cup \Sigma_2, x_o := (q_{o,1}, q_{o,2}, 1),$$

$$\xi((q_1,q_2,1),\sigma) := \begin{cases} (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma), 1), & \text{if } \delta_1(q_1,\sigma)!,\ \delta_2(q_2,\sigma)! \text{ and } q_1 \notin Q_{m,1}, \\ (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma), 2), & \text{if } \delta_1(q_1,\sigma)!,\ \delta_2(q_2,\sigma)! \text{ and } q_1 \in Q_{m,1}, \\ (\delta_1(q_1,\sigma), q_2, 1), & \text{if } \delta_1(q_1,\sigma)!,\ , \sigma \notin \Sigma_2, \text{ and } q_1 \notin Q_{m,1}, \\ (\delta_1(q_1,\sigma), q_2, 2), & \text{if } \delta_1(q_1,\sigma)!,\ , \sigma \notin \Sigma_2, \text{ and } q_1 \in Q_{m,1}, \end{cases}$$

$$\xi((q_1,q_2,2),\sigma) := \begin{cases} (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma), 2), & \text{if } \delta_1(q_1,\sigma)!,\ \delta_2(q_2,\sigma)! \text{ and } q_2 \notin Q_{m,2}, \\ (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma), 1), & \text{if } \delta_1(q_1,\sigma)!,\ \delta_2(q_2,\sigma)! \text{ and } q_2 \in Q_{m,2}, \\ (q_1, \delta_2(q_2,\sigma), 2), & \text{if } \delta_2(q_2,\sigma)!,\ \sigma \notin \Sigma_1, \text{ and } q_2 \notin Q_{m,2}, \\ \delta_2(q_2,\sigma), 1), & \text{if } \delta_2(q_2,\sigma)!,\ \sigma \notin \Sigma_1, \text{ and } q_2 \in Q_{m,2}, \end{cases}$$

$$X_m := Q_1 \times Q_{m,2} \times 2. \qquad \square$$

For $\mathcal{L}_1 \subseteq \Sigma_1^\omega$ and $\mathcal{L}_2 \subseteq \Sigma_2^\omega$, represented by G_1 and G_2, the synchronous ω-composition is given by $\mathcal{L}_1 \parallel \mathcal{L}_2 := \{w \in \Sigma^\omega \mid \mathrm{p}_1^\omega w \in \mathcal{L}_1 \text{ and } \mathrm{p}_2^\omega w \in \mathcal{L}_2\}$, with $\Sigma := \Sigma_1 \dot\cup \Sigma_2$.

2.3 Supervisory control of finite string languages

The area of control theory focuses on the systematic design of a controller that interacts with a dynamical system in a way that a desired behaviour is accomplished. In the case of discrete event systems, the desired system behaviour is given in form of particular event sequences that are desired and allowed for the considered dynamical system. A common approach to solve such control problems, is the analysis of the given system behaviour based on a mathematical system description, followed by a systematic controller design, that guarantees the fulfilment of the given specifications. For discrete event systems, formal languages, which have been introduced in Subsection 2.1, are used for the mathematical description of the system behaviour. The events that occur in the given physical plant are summarized in the alphabet Σ. Events of this alphabet are concatenated to event sequences in order to describe the given system behaviour in form of a formal language $L \subseteq \Sigma^*$. Some of the given events can be disabled by a supervisor. They are called *controllable events* Σ_{c}. Events that cannot be disabled are called *uncontrollable events* Σ_{uc}. The alphabet of a discrete event system is given by the union of these disjoint alphabets, i.e. $\Sigma = \Sigma_{\mathrm{c}} \dot{\cup} \Sigma_{\mathrm{uc}}$. Further, the alphabet can be partitioned into *observable* Σ_{o} and *unobservable events* Σ_{uo}, i.e. $\Sigma = \Sigma_{\mathrm{o}} \dot{\cup} \Sigma_{\mathrm{uo}}$. This partitioning depends on the observability of the respective events for a controller that shall interact with the given system. Note that for the scope of this paper, the focus lies on the specific case where all controllable events are assumed to be observable, i.e. $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$ is assumed throughout this thesis.

Given the description of the discrete event system behaviour in form of a language $L \subseteq \Sigma^*$, the purpose of Supervisory Control Theory is the construction of a discrete event controller that enforces a specified behaviour by applying appropriate control actions on the given discrete event system. The controller, referred to as *supervisor*, is defined in form of the following map.

Definition 2.3.1 Supervisor

A *supervisor* is defined as a map $f : \Sigma^* \to \Gamma$ that specifies for each string $s \in \Sigma^*$ a *control pattern* $\Gamma := \{\gamma \subseteq \Sigma \mid \Sigma_{\mathrm{uc}} \subseteq \gamma\}$. Further, for all $s, s' \in \Sigma^*$, with $\mathrm{p_o}(s) = \mathrm{p_o}(s')$, it is $f(s) = f(s')$. □

Due to the unobservability of some events, a supervisor cannot distinguish between all strings that occur in the given system without ambiguity. For strings that are equal under projection to the observable alphabet, it is not possible to impose different control pattern and the supervisor has to enable equal control pattern. For the practical implementation, a supervisor f can be represented alternatively by a language $H \subseteq \Sigma^*$. H is said to realize a supervisor f, if for each $s \in \Sigma^*$, $\mathrm{elig}_H(s) = f(s)$, i.e. the set of active events is equal to the set of events enabled for the string s by the map f.

The resulting interaction of plant and supervisor is demonstrated in Figure 2.2. The system $L \subseteq \Sigma^*$ generates event sequences, whereas the supervisor f partially observes the event sequences. The unobservability is comparable to a mask that allows only partial information to pass. Based on the observed events sequences, the supervisor enables appropriate control pattern in order to satisfy a given specification.

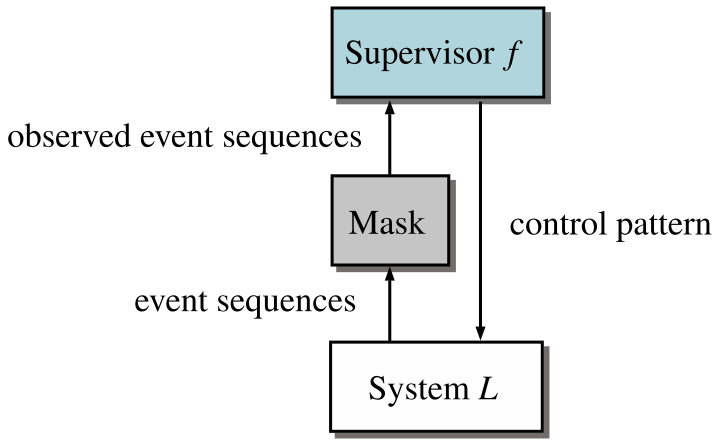


Figure 2.2: Discrete event system under supervision with partial observation

The prefix-closed behaviour of a supervisor interacting with a discrete event system L is called the *local closed-loop behaviour*. In the following, the formal definition according to (Ramadge and Wonham, 1987a) is introduced.

Definition 2.3.2 Local closed-loop behaviour

Given a discrete event system $L \subseteq \Sigma^*$ and a supervisor map $f : \Sigma^* \to \Gamma$, the *local closed-loop behaviour* [2] $(\mathrm{pre}L)_f$ for L under supervision of f, is defined iteratively by

- $\varepsilon \in (\mathrm{pre}L)_f$,
- $s\sigma \in (\mathrm{pre}L)_f$, for $s \in (\mathrm{pre}L)_f$, $s\sigma \in \mathrm{pre}L$, and $\sigma \in f(s)$,
- no other strings in $(\mathrm{pre}L)_f$. □

This definition states that, starting with the empty string ε, only those events are allowed in the closed-loop behaviour, that are possible in the given plant dynamics and enabled by the supervisor f. It is obvious by the definition that $(\mathrm{pre}L)_f \subseteq \mathrm{pre}L$ and $(\mathrm{pre}L)_f = \mathrm{pre}(\mathrm{pre}L)_f$. Hence, the closed-loop behaviour is a restriction of the uncontrolled *local system behaviour* $\mathrm{pre}L$.

[2] In the Supervisory Control Theory it is also denoted as *closed behaviour*.

From the local closed-loop behaviour the respective closed-loop behaviour on the finite-time horizon is deduced.

Definition 2.3.3 Finite-time closed-loop behaviour

It is given a discrete event system $L \subseteq \Sigma^*$, a supervisor map $f : \Sigma^* \to \Gamma$ and a respective local closed-loop behaviour $(\mathrm{pre}\, L)_f$ for L under supervision of f. Then, the *finite-time closed-loop behaviour*[3] is given by $L_f := (\mathrm{pre}\, L)_f \cap L$. $\square$

In order to guarantee a non-blocking local behaviour of the plant under supervision, the supervisor is requested to satisfy the following property.

Definition 2.3.4 Non-blocking supervisor

Given a discrete event system $L \subseteq \Sigma^*$, and a supervisor $f : \Sigma^* \to \Gamma$. The supervisor f is *non-blocking w.r.t. L*, if $(\mathrm{pre}\, L)_f = \mathrm{pre}(L_f)$. $\square$

The property of non-blocking[4] guarantees that the given plant satisfies its eventuality properties, i.e. terminates its tasks, under supervision of f. For every string in the local closed-loop behaviour it is possible to find a continuation in the local closed-loop behaviour to strings belonging to the plant behaviour.

In order to describe the given control problem formally, a language property is introduced that is necessary for characterizing those languages that can be generated in a local closed-loop behaviour.

Definition 2.3.5 Controllability

Given the $*$-languages $L,\ K \subseteq \Sigma^*$, and a set of uncontrollable events $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, K is said to be *controllable w.r.t.* $(\Sigma_{\mathrm{uc}}, L)$, if

$$((\mathrm{pre}\, K)\Sigma_{\mathrm{uc}}) \cap (\mathrm{pre}\, L) \subseteq \mathrm{pre}\, K. \qquad \square$$

Controllability guarantees that the candidate K is actually implementable and does not try to prohibit events that cannot be disabled. For a detailed discussion see (Wonham and Ramadge, 1984) and (Ramadge and Wonham, 1987a).

Another language property that includes the unobservability of events is given in the following definition.

[3] In the Supervisory Control Theory it is also denoted as *marked behaviour*

[4] The terminology *non-blocking supervisor* is shortly used for a supervisor that is *non-blocking w.r.t. L.*

Definition 2.3.6 Prefix-Normality

Given the $*$-languages $L,\ K \subseteq \Sigma^*$, and a set of observable events $\Sigma_o \subseteq \Sigma$, K is said to be *prefix-normal w.r.t.* (Σ_o, L), if

$$\mathrm{pre}\, K = (\mathrm{p}_o^{-1}\mathrm{p}_o\, \mathrm{pre}\, K) \cap (\mathrm{pre}\, L). \qquad \Box$$

For a detailed discussion of this property and its relation to the weaker condition of *observability*; see e.g. (Lin and Wonham, 1988a) and (Lin and Wonham, 1988b).

2.3.1 Control problem with partial event observation

The *control problem for $*$-languages with partial event observation* is defined as follows.

Definition 2.3.7 Control problem with partial event observation

Given a system $L \subseteq \Sigma^*$, with $\Sigma_{uc} \subseteq \Sigma$, $\Sigma_{uo} \subseteq \Sigma$, $\Sigma_c \subseteq \Sigma_o$, and a specification language $E \subseteq \Sigma^*$, find a non-blocking supervisor $f : \Sigma^* \to \Gamma$ such that $L_f \subseteq E$. $\Box$

Given a plant L interacting with a non-blocking supervisor, the properties of the closed-loop behaviour and the existence of a non-blocking supervisor map are analysed.

Proposition 2.3.1 (Lin and Wonham, 1988b)

Given a discrete event system $L \subseteq \Sigma^*$, with $\Sigma_{uc} \subseteq \Sigma$, $\Sigma_{uo} \subseteq \Sigma$, $\Sigma_c \subseteq \Sigma_o$, and a non-empty candidate $K \subseteq \Sigma^*$, then there exists a non-blocking supervisor map f, such that $K = L_f$ if and only if

L1 K is relatively closed w.r.t. L,

L2 K is controllable w.r.t. (Σ_{uc}, L), and

L3 K is prefix-normal w.r.t. (Σ_o, L). $\Box$

Detailed proofs for Proposition 2.3.1 are given in (Lin and Wonham, 1988b), as well as in (Cieslak et al., 1988) and (Cassandras and Lafortune, 2008).

In order to characterize a solution to the control problem, the set of controllable sublanguages of a given specification language $E \subseteq \Sigma^*$ is denoted by

$$[\mathrm{C}](E) := \{K \subseteq E \mid K \text{ is controllable w.r.t. } (\Sigma_{uc}, L)\}.$$

The set of all normal sublanguages of a specification $E \subseteq \Sigma^*$ is denoted by

$$[\mathrm{N}](E) := \{K \subseteq E \mid K \text{ is prefix-normal w.r.t. } (\Sigma_o, L)\}.$$

Further, the set of all relatively closed sublanguages w.r.t. L is denoted by

$$[\mathrm{F}](E) := \{K \subseteq E \mid K \text{ is relatively closed w.r.t. } L\}.$$

The set of controllable, normal and relatively closed sublanguages is denoted by

$$[\mathrm{CNF}](E) := \mathrm{C}(E) \cap \mathrm{N}(E) \cap \mathrm{F}(E).$$

In (Ramadge and Wonham, 1987a) and (Cieslak et al., 1988), it has been shown, that $[\mathrm{C}](E)$, $[\mathrm{F}](E)$ and $[\mathrm{N}](E)$ are non-empty and closed under arbitrary unions. In particular, $[\mathrm{CNF}](E)$ is also non-empty and closed under arbitrary unions. Hence, there exists a unique supremal element for $[\mathrm{CNF}](E)$, which is denoted by $\sup[\mathrm{CNF}](E)$. If $\sup[\mathrm{CNF}](E)$ is non-empty, then there exists a solution to the control problem. In particular, $\sup[\mathrm{CNF}](E)$ is the optimal solution to the control problem, denoted also the *minimally restrictive solution* $K^{\Uparrow} := \bigcup\{K \subseteq E \mid K$ satisfies L1 - L3 $\}$.

Theorem 2.3.2 (Lin and Wonham, 1988b)

Given a discrete event system $L \subseteq \Sigma^*$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{uo}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and a specification $E \subseteq \Sigma^*$. There exists a non-blocking supervisor $f : \Sigma^* \to \Gamma$ w.r.t. L such that $L_f \subseteq E$, if and only if $\sup[\mathrm{CNF}](E) \neq \emptyset$. In particular, $K^{\Uparrow} = \sup[\mathrm{CNF}](E)$. □

Following the proofs in (Cieslak et al., 1988) and (Cassandras and Lafortune, 2008), the respective supervisor to a candidate $K \subseteq L$ satisfying L1 - L3, is deduced by

$$f(s) := \Sigma_{\mathrm{uc}} \cup \{\sigma \in \Sigma_{\mathrm{c}} \mid (\exists s'\sigma \in \mathrm{pre}\,K)[\mathrm{p_o}s' = \mathrm{p_o}s]\}.$$

Useful formulas for the computation of supremal sublanguages are given in (Cassandras and Lafortune, 2008) and (Ziller and Cury, 1994a,b). In particular, the computation of $\sup[\mathrm{CNF}](E)$ on the basis of finite-state automata is described in (Cho and Marcus, 1989a), (Yoo et al., 2002) or (Moor et al., 2012).

2.3.2 Control problem with partial observation and completeness

From the control problem under partial observation on the finite-time horizon, a control problem can be derived, that focuses on the infinite-time horizon. Therefore, the limit of the system behaviour $\lim L$ is considered. It is further assumed that L is a complete language. In order to guarantee a non-blocking behaviour of the plant under supervision on the infinite-time horizon, the non-blocking property of the supervisor from Definition 2.3.4 is extended and defined on the basis of $\lim L$.

Definition 2.3.8 Non-blocking supervisor

Given a discrete event system $L \subseteq \Sigma^*$, and a supervisor $f : \Sigma^* \to \Gamma$. The supervisor f is *non-blocking w.r.t.* $\lim L$, if $(\mathrm{pre}\lim L)_f = \mathrm{pre}\lim L_f$. □

The system behaviour under supervision of a non-blocking supervisor f is described by the limit of the finite-time closed-loop behaviour $\lim L_f = (\lim(\mathrm{pre}L)_f) \cap (\lim L)$. Further note that for a complete language L, the non-blocking property on the finite-time horizon is a direct consequence of the non-blocking property on the infinite-time horizon, i.e. $\mathrm{pre}L_f = (\mathrm{pre}L)_f$. For the given system L under supervision of a non-blocking supervisor f there can be deduced particular properties for the closed-loop behaviour.

Proposition 2.3.3

Given a discrete-event system $L \subseteq \Sigma^*$, with L is a complete language, $\Sigma_{\mathrm{uo}} \subseteq \Sigma$, $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$ and non-empty candidate $K \subseteq \Sigma^*$, then there exists a supervisor f that is non-blocking w.r.t. $\lim L$, such that $\lim K = \lim L_f$ if and only if

L0 K is complete,

L1 K is relatively closed w.r.t. L,

L2 K is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, L)$,

L3 K is prefix-normal w.r.t. (Σ_{o}, L).

Proof. To show sufficiency, assume that K exhibits the above properties. According to (Ramadge and Wonham, 1987a), properties L2 and L3 guarantee the existence of a supervisor f such that $(\mathrm{pre}L)_f = \mathrm{pre}K$. For the infinite-time closed-loop behaviour, L1 implies that $\lim L_f = \lim((\mathrm{pre}L)_f \cap L) = \lim(\mathrm{pre}K \cap L) = \lim K$. In particular, f is a non-blocking supervisor for $\lim L$, since $\mathrm{pre}\lim L_f = \mathrm{pre}\lim K = \mathrm{pre}K = (\mathrm{pre}L)_f = (\mathrm{pre}\lim L)_f$. Further, for all $s, s' \in \Sigma^*$, with $\mathrm{p_o}(s) = \mathrm{p_o}(s')$, it is $f(s) = f(s')$, according to Proposition 2.3.1. To show necessity, detailed proofs for L1 - L3 are given in (Lin and Wonham, 1988b), as well as in (Cieslak et al., 1988) and (Cassandras and Lafortune, 2008). Regarding L0, select an arbitrary string $s \in \mathrm{pre}K = \mathrm{pre}L_f$. By completeness of L and the non-blocking characteristic of the supervisor, observe that $s \in (\mathrm{pre}\lim L)_f = \mathrm{pre}\lim L_f$. Hence, there exists a string $u \in \Sigma^\omega$, such that $su \in \lim L_f$. Consequently, there exists $\sigma \in \Sigma$, such that $s\sigma < su$ and, thus, $s\sigma \in \mathrm{pre}\lim L_f = \mathrm{pre}\lim K \subseteq \mathrm{pre}K$. □

The *control problem for $*$-languages with partial observation and completeness* is defined.

Definition 2.3.9 Control problem with partial observation and completeness

Given a system $L \subseteq \Sigma^*$, with L is complete, $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{uo}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and a specification language $E \subseteq \Sigma^*$, with E is complete, find a supervisor $f : \Sigma^* \to \Gamma$ that is non-blocking w.r.t. $\lim L$ such that $\lim L_f \subseteq \lim E$. □

In order to characterize a solution to the control problem, the set of complete sub-languages of a given specification language $E \subseteq \Sigma^*$ is denoted by

$$[\mathrm{O}](E) := \{K \subseteq E \mid K \text{ is complete }\}.$$

The set of complete languages is closed under arbitrary union; see also (Kumar et al., 1992). A useful formula for the computation of $\sup[\mathrm{O}](E)$ is also given therein. Further, the set of controllable, normal, relatively closed and complete sub-languages of a given specification language w.r.t. the plant language L is defined by

$$\mathrm{CNFO}(E) = \mathrm{C}(E) \cap \mathrm{N}(E) \cap \mathrm{F}(E) \cap \mathrm{O}(E).$$

Controllability, prefix-normality, relative closedness and completeness are retained under arbitrary union. In particular, a supremal sub-language that possesses the above properties exists uniquely and is denoted $\sup \mathrm{CNFO}(E)$. Hence, there exists a solution to the control problem iff $\sup[\mathrm{CNFO}](E)$ is non-empty. In particular, $\lim \sup[\mathrm{CNFO}](E)$ is the supremal solution $(\lim K)^{\Uparrow}$ to the given control problem.

Theorem 2.3.4

Given a discrete event system $L \subseteq \Sigma^*$, with L is complete, $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{uo}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and a specification $E \subseteq \Sigma^*$, with E is complete. There exists a non-blocking supervisor $f : \Sigma^* \to \Gamma$ w.r.t. $\lim L$ such that $\lim L_f \subseteq \lim E$, if and only if $\sup[\mathrm{CNFO}](E) \neq \emptyset$. In particular, $(\lim K)^{\Uparrow} = \lim \sup[\mathrm{CNFO}](E)$. $\square$

The respective supervisor to a candidate K is again deduced by

$$f(s) := \Sigma_{\mathrm{uc}} \cup \{\sigma \in \Sigma_{\mathrm{c}} : (\exists s'\sigma \in \mathrm{pre} K)[\mathrm{p_o} s' = \mathrm{p_o} s]\}.$$

A formula for the computation of the supremal sublanguage is given in (Moor et al., 2012).

3 Monolithic controller design for ω-languages

The focus of this thesis lies on the development of a hierarchical control architecture for infinite string languages. In order to describe this set-up, this Chapter introduces the basic ideas of the monolithic controller design for ω-languages and extends it to a set-up based on input/output systems, known from the Behavioural System Theory, see (Willems, 1991), and proven to be an appropriate set-up for hierarchical, abstraction based controller design, see e.g. (Moor et al., 2011) and (Perk et al., 2008).

Finite automata modelling infinite string languages have been introduced in the 60s, in publications like (Büchi, 1966) or (McNaughton, 1966) and have been used to solve decision problems in mathematical logics. In contrast to finite string languages, they describe system behaviour on the infinite-time horizon and offer the possibility to model more general liveness properties. Using ω-languages, it is possible to distinguish between transient and non-transient behaviour and to model e.g. eventual task completion or eventual feedback. The area of Temporal Logics is strongly connected to a representation of system behaviour or specified behaviour by ω-automata and a detailed classification of liveness properties is given in (Manna and Pnueli, 1990) and (Baier and Kwiatkowska, 2000). After using ω-languages in this area only for describing the behaviour of concurrent systems, see (Pnueli and Rosner, 1989), there have been developed Model Checking techniques that allow for automated synthesis procedures; see e.g. (Emerson and Clarke, 1980) and (Clarke et al., 1986). Those techniques have been proven to be successful in industrial applications. Hence, the interest in combining the area of Supervisory Control with the area of *Reactive Synthesis* and *Model Checking* is increasing in order to benefit from both approaches; see (Ziller and Schneider, 2005) and (Ehlers et al., 2014). To close the gap between them, it is important to focus more on the representation of system behaviour by infinite-string languages and to further develop existing synthesis methods for ω-languages in the area of Supervisory Control.

A first approach regarding controller synthesis for ω-languages and a first analysis regarding its application areas has already been presented in (Ramadge, 1989). Therein, the control problem for sequential behaviours under complete event observation and the characterization of respective solutions has been presented. Further, the challenge to find a supremal solution has been discussed in detail. In (Kumar et al., 1992) first results regarding controller design for sequential behaviour under partial observation are given. However, those results are limited to topologically closed languages, which are only appropriate for modelling safety properties, see also (Alpern and Schneider, 1985). Those languages can only describe that some critical configuration is not attained. However, for the description that some desired configuration is attained always eventually, i.e. for the description of liveness properties, not topologically closed behaviours have to be used. The liveness property is of particular interest in the context of hierarchical and abstraction based controller design for large scale systems. In that context, it is important to guarantee that liveness is given for the overall system of all concurrent and interacting sub-systems. More general liveness properties in the context of controller design for ω-languages have been analysed in (Thistle and Wonham, 1994a) and (Thistle and Lamouchi, 2009) and algorithm for the computation of solutions are introduced in (Thistle and Wonham, 1992) and (Thistle and Lamouchi, 2009). Further, in (Moor et al., 2011) it is described an approach for abstraction based controller design for ω-languages, which is the basis for the hierarchical controller design developed in this thesis.

The present Chapter is organized as follows. In Section 3.1 results and concepts of Supervisory Control Theory for infinite string languages are revealed. Two different control problems are considered, control problems for behaviour under complete event observation and for behaviour under partial event observation. It is illustrated how to handle the particular liveness characteristics of infinite-time behaviours and how to design a supervisor that guarantees the satisfaction of a specification that is also given as an infinite-string language. In addition to the theoretical framework, algorithms for the computation of solutions are investigated in this section. In the second part of this Chapter, a control problem for ω-languages is introduced which is based on an input/output system description from Behavioural System Theory and the control problem for sequential behaviour under partial event observation. This particular control problem is used in Chapter 4 to set up the hierarchical control framework for ω-languages.

3.1 Supervisory control of infinite string languages

In Subsection 2.3, the control problem for finite-string languages under partial event observation and completeness has been developed and it has been pointed out a way how to solve this particular control problem. In this Subsection, the control problem for infinite-string languages is described using results from Subsection 2.3.

The behaviour of the uncontrolled system is given by an infinite-string language $\mathcal{L} \subseteq \Sigma^{\omega}$. Note that the alphabet Σ consists of controllable and uncontrollable, as well as observable and unobservable events, i.e $\Sigma := \Sigma_{\mathrm{c}} \dot{\cup} \Sigma_{\mathrm{uc}} = \Sigma_{\mathrm{o}} \dot{\cup} \Sigma_{\mathrm{uo}}$. To influence the given system in that way that the fulfilment of a given specification $\mathcal{E} \subseteq \Sigma^{\omega}$ is guaranteed, a supervisor f according to Definition 2.3.1 shall be designed. The resulting closed-loop configuration is given in Figure 3.1.

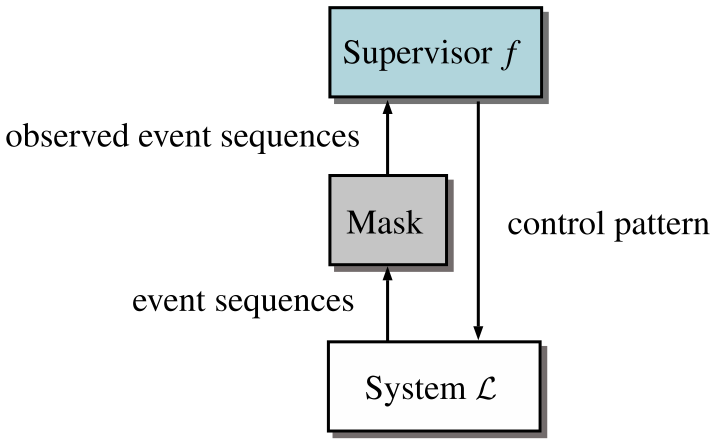


Figure 3.1: Discrete event system under supervision with partial observation

Further, the closed-loop behaviour of the plant under supervision of f on the infinite-time horizon is introduced

Definition 3.1.1 Infinite-time closed-loop behaviour

Given a discrete-event system $\mathcal{L} \subseteq \Sigma^{\omega}$, a supervisor map $f : \Sigma^{*} \to \Gamma$ and the local closed-loop behaviour $(\mathrm{pre}\,\mathcal{L})_f \subseteq \mathrm{pre}\,\mathcal{L}$, then the *infinite-time closed-loop behaviour* $\mathcal{L}_f \subseteq \Sigma^{\omega}$ for $\mathcal{L}$ under supervision of f is given by

$$\mathcal{L}_f := (\lim(\mathrm{pre}\,\mathcal{L})_f) \cap \mathcal{L}. \qquad \square$$

In order to guarantee that the local closed-loop behaviour $(\mathrm{pre}\,\mathcal{L})_f$ is complete, the supervisor f is requested to be *non-blocking w.r.t.* $\mathcal{L}$, see also Definition 2.3.8.

Definition 3.1.2 Non-blocking supervisor

Given a discrete-event system $\mathcal{L} \subseteq \Sigma^\omega$ and a supervisor $f : \Sigma^* \to \Gamma$, the supervisor is *non-blocking w.r.t.* $\mathcal{L}$, if $(\operatorname{pre}\mathcal{L})_f = \operatorname{pre}\mathcal{L}_f$. [1] $\square$

In the following, it is analysed how to design a non-blocking supervisor for a plant behaviour that is given as a sequential behaviour, with possibly uncontrollable and unobservable events. The supervisor shall guarantee that the plant satisfies a language inclusion specification that is given in form of an ω-language.

3.1.1 Control problem under complete event observation

Before analysing the control problem under partial observation, the control problem under complete observation, is considered. Hence, all events are assumed to be observable, i.e. $\Sigma_o = \Sigma$. This control problem has been discussed for the first time in detail in (Ramadge, 1989). In (Kumar et al., 1992) a relevant connection to the $*$-language control problem involving the property of completeness is illustrated in the case of closed languages, see also Subsection 2.3.2. Further, in (Thistle and Wonham, 1994a), the control problem has been further analysed regarding the characterisation of solutions including as well specifications that are relatively closed w.r.t. the plant behaviour, as more general specifications.

The *control problem for infinite languages with complete event observation* is defined as follows.

Definition 3.1.3 Control problem with complete event observation

Given a system $\mathcal{L} \subseteq \Sigma^\omega$, with $\Sigma_{uc} \subseteq \Sigma$, $\Sigma_o = \Sigma$, and a specification language $\mathcal{E} \subseteq \Sigma^\omega$, find a supervisor $f : \Sigma^* \to \Gamma$ that is non-blocking w.r.t. $\mathcal{L}$ such that $\mathcal{L}_f \subseteq \mathcal{E}$. $\square$

Considering a system behaviour $\mathcal{L} \subseteq \Sigma^\omega$ and a supervisor f that is non-blocking w.r.t. $\mathcal{L}$, the infinite-time closed-loop behaviour $\mathcal{L}_f$ results to have particular properties, summarized in Proposition 3.1.1. Further, necessary conditions for the existence of a non-blocking supervisor map f are analysed therein.

[1] In the following, the terminology *non-blocking supervisor* is shortly used for a supervisor that is *non-blocking w.r.t.* $\mathcal{L}$.

Proposition 3.1.1 (Ramadge, 1989)

Given $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{o}} = \Sigma$, and a non-empty candidate $\mathcal{K} \subseteq \Sigma^{\omega}$, then there exists a non-blocking supervisor f w.r.t. $\mathcal{L}$ such that $\mathcal{L}_f = \mathcal{K}$ if and only if

I1 $\mathcal{K}$ is relatively closed w.r.t. $\mathcal{L}$,

I2 $\operatorname{pre}\mathcal{K}$ is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \operatorname{pre}\mathcal{L})$.

Proof. To show sufficiency, assume that $\mathcal{K}$ exhibits the above properties. According to (Ramadge, 1989), non-emptiness of $\mathcal{K}$ and property I2 guarantee the existence of a supervisor f such that $(\operatorname{pre}\mathcal{L})_f = \operatorname{pre}\mathcal{K}$. With respect to the infinite-time closed-loop behaviour, I1 implies that $\mathcal{L}_f = (\lim(\operatorname{pre}\mathcal{L})_f) \cap \mathcal{L} = (\lim \operatorname{pre}\mathcal{K}) \cap \mathcal{L} = \mathcal{K}$. In particular, f is a non-blocking supervisor for $\mathcal{L}$, since $\operatorname{pre}\mathcal{L}_f = \operatorname{pre}\mathcal{K} = (\operatorname{pre}\mathcal{L})_f$. Further, for all $s, s' \in \Sigma^*$, with $\mathrm{p_o}(s) = \mathrm{p_o}(s')$, it is $f(s) = f(s')$, since all events are observable.
To show necessity, note that property I2 follows by $(\operatorname{pre}\mathcal{L})_f = \operatorname{pre}\mathcal{L}_f = \operatorname{pre}\mathcal{K}$. Regarding property I1, it is $(\operatorname{clo}\mathcal{K}) \cap \mathcal{L} = (\lim \operatorname{pre}\mathcal{K}) \cap \mathcal{L} = (\lim \operatorname{pre}\mathcal{L}_f) \cap \mathcal{L}$. Non-blocking of f w.r.t. $\mathcal{L}$ implies that $(\lim \operatorname{pre}\mathcal{L}_f) \cap \mathcal{L} = (\lim(\operatorname{pre}\mathcal{L})_f) \cap \mathcal{L}$. By the definition of $\mathcal{L}_f$, it follows that $(\lim(\operatorname{pre}\mathcal{L})_f) \cap \mathcal{L} = \mathcal{L}_f = \mathcal{K}$. Hence, $(\operatorname{clo}\mathcal{K}) \cap \mathcal{L} = \mathcal{K}$. $\square$

To analyse the existence of a solution to the control problem, the union of all closed-loop behaviours $\mathcal{K}$ that solve the control problem is defined by

$$\mathcal{K}^{\Uparrow} := \bigcup \{\mathcal{K} \subseteq \mathcal{E} \mid \mathcal{K} \text{ satisfies I1 and I2 }\}.$$

However, the set of languages that satisfy I1 is only closed under finite set union, but not under arbitrary set union. Hence, in general, the existence of a unique supremal element that satisfies I1 and I2 is not given. Only if the specification language $\mathcal{E}$ is relatively closed w.r.t. $\mathcal{L}$, a unique supremal solution exists.

Lemma 3.1.2 (Ramadge, 1989)

It is given a system $\mathcal{L} \subseteq \Sigma^{\omega}$, $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{o}} = \Sigma$ and a specification language $\mathcal{E} \subseteq \mathcal{L}$, that is relatively closed w.r.t. $\mathcal{L}$. Then, a unique supremal sub-language of $\mathcal{E}$ that satisfies I1 and I2 exists. $\square$

Hence, consider a specification $\mathcal{E} \subseteq \Sigma^{\omega}$ that is relatively closed. In addition, plant and specification languages are representable as limits of regular $*$-languages, i.e. $\mathcal{L} = \lim L$ and $\mathcal{E} = \lim E$. Further, define that L is complete. Given relative closedness of $\mathcal{E}$, it is readily verified that $\mathcal{E} = \lim(\operatorname{pre}\mathcal{E} \cap L)$. Thus, without loss of generality, it is assumed that E is relatively prefix-closed w.r.t. L. Then, the supremal solution $\mathcal{K}^{\Uparrow}$ of the control problem can be characterised on the basis of the properties L0 - L2, see also Subsection 2.3.2.

Theorem 3.1.3

Consider a control problem given by a system $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{o}} = \Sigma$ and a specification $\mathcal{E} \subseteq \mathcal{L}$. Assume that $\mathcal{E}$ is relatively topologically closed w.r.t. $\mathcal{L}$, and that both $\mathcal{L}$ and $\mathcal{E}$ can be represented as limits of regular $*$-languages $\mathcal{L} = \lim L$, $\mathcal{E} = \lim E$, where $L \subseteq \Sigma^*$ is complete and $E = (\operatorname{pre}\mathcal{E}) \cap L$. Then $\lim\sup[\mathrm{CFO}](E)$ is the supremal solution $\mathcal{K}^{\Uparrow}$ to the control problem.

Proof. To show $\lim\sup[\mathrm{CFO}](E) \subseteq \mathcal{K}^{\Uparrow}$, consider an arbitrary $K \in [\mathrm{CFO}](E)$ and denote $\mathcal{K} := \lim K$. By property L0, it is $\operatorname{pre}\mathcal{K} = \operatorname{pre}K$ and, hence, by Proposition 2.3.3 and property L2, $\mathcal{K}$ satisfies the property I2. Further, by property L1, it is $\operatorname{clo}\mathcal{K} \cap \mathcal{L} = (\lim\operatorname{pre}K) \cap (\lim L) = \lim(\operatorname{pre}K \cap L) = \lim K = \mathcal{K}$. Finally, by $K \subseteq E$, it is $\mathcal{K} \subseteq \mathcal{E}$. Consequently, $\lim\sup[\mathrm{CFO}](E) \subseteq \mathcal{K}^{\Uparrow}$. Vice versa, by Lemma 3.1.2 $\mathcal{K}^{\Uparrow} \subseteq \mathcal{E}$ and $\mathcal{K}^{\Uparrow}$ satisfies I1 and I2. Further, by Proposition 3.1.1, there exists a non-blocking supervisor f such that $\mathcal{K}^{\Uparrow} = \lim L_f$. Thus, by Proposition 2.3.3, it follows that $K := (\operatorname{pre}L)_f \cap L$ satisfies L0 - L2. By non-blocking of f, observe that further $K = (\operatorname{pre}L)_f \cap L = (\operatorname{pre}\lim L_f) \cap L = (\operatorname{pre}\mathcal{K}^{\Uparrow}) \cap L$. Finally, $K = (\operatorname{pre}\mathcal{K}^{\Uparrow}) \cap L \subseteq (\operatorname{pre}\mathcal{E}) \cap L = E$. This concludes the proof of $\mathcal{K}^{\Uparrow} \subseteq \lim\sup[\mathrm{CFO}](E)$. □

Consequently, if a non-empty solution to the given control problem exists, it is given by $\mathcal{K}^{\Uparrow} = \lim\sup[\mathrm{CFO}](E)$. A formula for the computation of $\sup[\mathrm{CFO}](E)$, is given in (Moor et al., 2012).

However, requesting relative closedness of $\mathcal{E}$ w.r.t. $\mathcal{L}$ is restrictive. In particular, the control problem developed in Subsection 3.2 is based on specifications that are, in general, not relatively closed. For the general case, it is therefore developed an alternative characterisation of a solution to the control problem.

Definition 3.1.4 ω-controllability

Given a system $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{o}} = \Sigma$, and the candidate $\mathcal{K} \subseteq \Sigma^{\omega}$. Then, $\mathcal{K}$ is said to be *ω-controllable* w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$, if for all $s \in (\operatorname{pre}\mathcal{L}) \cap (\operatorname{pre}\mathcal{K})$ there exists a $\mathcal{V}_s \subseteq \mathcal{L} \cap \mathcal{K}$ with $s \in \operatorname{pre}\mathcal{V}_s$, and

C1 $\mathcal{V}_s$ is relatively topologically closed w.r.t. $\mathcal{L}$, and

C2 $\operatorname{pre}\mathcal{V}_s$ is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \operatorname{pre}\mathcal{L})$. □

Given this alternative controllability characterisation, it has to be proven that properties I1 and I2 can be substituted by ω-controllability. Therefore, it is first shown that any $\mathcal{K} \subseteq \mathcal{E}$ that satisfies I1 and I2 is indeed ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$.

Proposition 3.1.4

Given the languages $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{o}} = \Sigma$ and $\mathcal{K} \subseteq \Sigma^{\omega}$. If $\operatorname{pre}\mathcal{K}$ is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \operatorname{pre}\mathcal{L})$ and $\mathcal{K}$ is relatively closed w.r.t. $\mathcal{L}$, then $\mathcal{K}$ is ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$.

Proof. To show ω-controllability, select an arbitrary $s \in (\operatorname{pre}\mathcal{L}) \cap (\operatorname{pre}\mathcal{K})$. Define $\mathcal{V}_s := \mathcal{K}$ and observe, that $\mathcal{V}_s$ satisfies C1 and C2 and $s \in \operatorname{pre}\mathcal{V}_s$. Further $\mathcal{V}_s = \mathcal{K} = \operatorname{clo}\mathcal{K} \cap \mathcal{L} = \mathcal{K} \cap \mathcal{L}$. Thus, $\mathcal{K}$ is ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$. □

Like I1 and I2 in the case of relatively closed specifications, the property of ω-controllability guarantees the existence of an appropriate non-blocking supervisor map f such that the behaviour of the infinite-time closed-loop system satisfies $\mathcal{L}_f \subseteq \mathcal{E}$.

Proposition 3.1.5

Given the languages $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{o}} = \Sigma$ and $\mathcal{K} \subseteq \Sigma^{\omega}$, $\mathcal{K} \neq \emptyset$. If $\mathcal{K}$ is ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$, then there exists a non-blocking supervisor $f : \Sigma^* \to \Gamma$ w.r.t. $\mathcal{L}$ such that $\mathcal{L}_f \subseteq \mathcal{K}$.

Proof. Assume that $\mathcal{K}$ exhibits the above properties. According to Definition 3.1.4, for every $s \in \operatorname{pre}\mathcal{K}$ there exists $\mathcal{V}_s \subseteq \mathcal{K}$, satisfying C1 and C2. By definition, $\varepsilon \in \operatorname{pre}\mathcal{K}$. Hence, there exists $\mathcal{V}_{\varepsilon} \subseteq \mathcal{K} \cap \mathcal{L}$ satisfying C1 and C2, i.e. $\mathcal{V}_{\varepsilon}$ is relatively closed w.r.t. $\mathcal{L}$ and $\operatorname{pre}\mathcal{V}_{\varepsilon}$ is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \operatorname{pre}\mathcal{L})$. Hence, $\mathcal{V}_{\varepsilon}$ satisfies I1 and I2. By Proposition 3.1.1, there exists a non-blocking supervisor map f such that $\mathcal{L}_f = \mathcal{V}_{\varepsilon}$. Since, $\mathcal{V}_{\varepsilon} \subseteq \mathcal{K} \cap \mathcal{L}$ it follows that $\mathcal{L}_f \subseteq \mathcal{K}$. □

The subset of ω-controllable languages of a given specification $\mathcal{E} \subseteq \Sigma^{\omega}$ is denoted by

$$[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E}) := \{\mathcal{K} \subseteq \mathcal{E} \mid \mathcal{K} \text{ is } \omega\text{-controllable w.r.t. } (\Sigma_{\mathrm{uc}}, \mathcal{L})\}.$$

In contrast to property I1, ω-controllability is preserved under arbitrary unions.

Lemma 3.1.6

It is given the language $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$. Consider a family of languages $\mathcal{K}_a \subseteq \Sigma^{\omega}$, $a \in A$, each one ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$. Then, the union $\mathcal{K} := \cup_{a \in A}\mathcal{K}_a$ is ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$, too.

Proof. Pick an arbitrary prefix $s \in (\operatorname{pre}\mathcal{K})$. By $\operatorname{pre}\mathcal{K} = \cup_{a \in A} \operatorname{pre}\mathcal{K}_a$, it is possible to choose $a \in A$ such that $s \in \operatorname{pre}\mathcal{K}_a$. Since $\mathcal{K}_a$ is ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$ by definition, it is possible to choose $\mathcal{V}_s \subseteq \mathcal{K}_a$, satisfying conditions C1 and C2 from Definition 3.1.4. Clearly, $\mathcal{V}_s \subseteq \mathcal{K}$ and ω-controllability of $\mathcal{K}$ w.r.t $(\Sigma_{\mathrm{uc}}, \mathcal{L})$ has been established. □

Hence, there exists a unique supremal element, which is denoted by $\sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E})$. In particular, the least upper bound of the closed-loop behaviours $\mathcal{K}^{\Uparrow}$ is equal to $\sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E})$.

Proposition 3.1.7 (Moor et al., 2011)

Given a system $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, and a specification language $\mathcal{E} \subseteq \Sigma^{\omega}$. Then, the union of all closed-loop behaviours $\mathcal{K}$ that solve the control problem is equal to the supremal ω-controllable sub-language of $\mathcal{E}$, i.e. $\mathcal{K}^{\Uparrow} = \sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E})$.

Proof. To show that $\mathcal{K}^{\Uparrow} \subseteq \sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E})$, pick $s \in (\operatorname{pre}\mathcal{L}) \cap (\operatorname{pre}\mathcal{K}^{\Uparrow})$. Since the prefix operator distributes over arbitrary unions, there exists a component $\mathcal{K}$ of the union representation $\mathcal{K}^{\Uparrow}$, with $s \in \operatorname{pre}\mathcal{K}$, $\mathcal{K} \subseteq \mathcal{E}$, and $\mathcal{K}$ satisfies C1 and C2. In particular, $\mathcal{K} \subseteq \mathcal{L} \cap \mathcal{K}^{\Uparrow}$. Hence, $\mathcal{K}^{\Uparrow}$ is ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$. By supremality of $\sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E})$ follows that $\mathcal{K}^{\Uparrow} \subseteq \sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E})$. To show that $\sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E}) \subseteq \mathcal{K}^{\Uparrow}$, pick an arbitrary $w \in \sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E})$ and recall that for each prefix $s < w$, there exists $\mathcal{V}_s$ satisfying C1 and C2. Based on $\mathcal{V}_s$, construct for all $s < w$ a component $\mathcal{K}$ of the union representation, in order to obtain $w \in \mathcal{K}^{\Uparrow}$. Let $\mathcal{K}_s := \{v \in \mathcal{V}_s \mid v^{(|s|)} = w^{(|s|)} \text{ and } v^{(|s|+1)} \neq w^{(|s|+1)}\}$ and consider the candidate $\mathcal{K} := (\cup_{s<w}\mathcal{K}_s) \cup \{w\}$. Observe that the union is disjoint by construction. Further, for $v \in \mathcal{K}$ and $v \neq w$, $v \in \mathcal{K}_r$, where r is the maximal prefix $r < v$ such that $r < w$. To show I1, pick an arbitrary $v \in \operatorname{clo}\mathcal{K} \cap \mathcal{L}$. If $v = w$, then it is $v \in \mathcal{K}$, by definition. If $v \neq w$, pick the maximal prefix $r < v$ such that $r < w$. Then, any $t < v$ it is $t \in \operatorname{pre}\mathcal{K}_r$ and, hence, $v \in \operatorname{pre}\lim\mathcal{K}_r$. By C1 follows that $v \in \mathcal{K}_r \subseteq \mathcal{K}$. To show I2, pick $s \in \operatorname{pre}\mathcal{K}$ and $\sigma \in \Sigma_{\mathrm{uc}}$ such that $s\sigma \in \operatorname{pre}\mathcal{L}$. If $s\sigma < w$, then $w \in \mathcal{K}$ implies that $s\sigma \in \operatorname{pre}\mathcal{K}$. If $s\sigma \not< w$, pick the maximal prefix $r < s\sigma$ such that $r < w$. Then, $s \in \operatorname{pre}\mathcal{K}_r \subseteq \operatorname{pre}\mathcal{V}_r$, and, by C2, $s\sigma \in \operatorname{pre}\mathcal{V}_r$. By definition, $s\sigma \in \operatorname{pre}\mathcal{K}_r$. This concludes the proof of $\sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E}) = \mathcal{K}^{\Uparrow}$. □

As a consequence, a solution to the control problem for infinite behaviours under complete observation does only exist, if $\sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E}) \neq \emptyset$. Note further that a unique maximal solution to the control problem exists if and only if $\sup[\mathrm{C}^{\omega}_{\mathrm{co}}]$ is relatively closed w.r.t. $\mathcal{L}$. In particular, in that case, $\sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E})$ is the unique maximal solution. Detailed formulas for the computation of $\sup[\mathrm{C}^{\omega}_{\mathrm{co}}](\mathcal{E})$ will be further illustrated in Section 3.1.3.

3.1.2 Control problem under partial event observation

First considerations regarding the control problem for infinite string languages under partial event observation are given in (Kumar et al., 1992), however only for the case of closed languages. In (Thistle and Lamouchi, 2009) the control problem under partial observation has been solved for the more general case and a computational procedure has been presented that is based on a language representation by Rabin automata. However, this procedure requires the plant behaviour to be closed. Further, it includes the case of unobservable controllable events, that is not considered in this thesis. A more general approach and a more general synthesis procedure, motivated by similar control problems in the area of program synthesis, is given in (Kupferman and Vardi, 2000). This approach is based on alternating tree automata and reduces the solvability of the control synthesis problem to the non-emptiness problem for alternating automata on infinite trees. In this thesis the control problem under partial event observation for infinite string languages is developed following the illustrations of the control problem under complete observation in Subsection 3.1.1. It is considered a system behaviour $\mathcal{L} \subseteq \Sigma^\omega$ with uncontrollable and unobservable events, i.e. $\Sigma_{\mathrm{uc}} \subseteq \Sigma$ and $\Sigma_{\mathrm{uo}} \subseteq \Sigma$. A supervisor f that is non-blocking w.r.t. $\mathcal{L}$ shall be synthesised such that a given specification language $\mathcal{E} \subseteq \Sigma^\omega$ is satisfied. The *control problem with partial event observation* is defined.

Definition 3.1.5 Control problem with partial event observation

Given a system $\mathcal{L} \subseteq \Sigma^\omega$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{uo}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and a specification $\mathcal{E} \subseteq \Sigma^\omega$, find a supervisor $f : \Sigma^* \to \Gamma$ that is non-blocking w.r.t. $\mathcal{L}$ such that $\mathcal{L}_f \subseteq \mathcal{E}$. □

Given a supervisor f that is non-blocking w.r.t. $\mathcal{L}$, the infinite-time closed-loop behaviour $\mathcal{L}_f$ results to have the properties I1 - I3, given in the proposition below. Necessary conditions for the existence of a non-blocking supervisor map f are derived.

Proposition 3.1.8

Given $\mathcal{L} \subseteq \Sigma^\omega$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma, \Sigma_{\mathrm{uo}} \subseteq \Sigma, \Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and a non-empty $\mathcal{K} \subseteq \Sigma^\omega$. Then, there exists a non-blocking supervisor map f w.r.t. $\mathcal{L}$ such that the infinite-time closed-loop system $\mathcal{L}_f = \mathcal{K}$ if and only if

I1 $\mathcal{K}$ is relatively closed w.r.t. $\mathcal{L}$,

I2 $\mathrm{pre}\,\mathcal{K}$ is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathrm{pre}\,\mathcal{L})$,

I3 $\mathrm{pre}\,\mathcal{K}$ is prefix-normal w.r.t. $(\Sigma_{\mathrm{o}}, \mathrm{pre}\,\mathcal{L})$.

Proof. To show sufficiency, assume that $\mathcal{K}$ exhibits the above properties. From I2 and I3 follows directly, that $\mathrm{pre}\,\mathcal{K}$ satisfies L2 and L3 according to Proposition 2.3.1. Properties L2 and L3 guarantee the existence of a supervisor f such that $(\mathrm{pre}\,\mathcal{L})_f = \mathrm{pre}\,\mathcal{K}$. In particular, f is a non-blocking supervisor for $\mathcal{L}$, since $\mathrm{pre}\,\mathcal{L}_f = \mathrm{pre}\,\mathcal{K} = (\mathrm{pre}\,L)_f$. For the infinite-time closed-loop behaviour, recall that, according to the definition of $\mathcal{L}_f$ and non-blocking of f w.r.t. $\mathcal{L}$, $\mathcal{L}_f = (\lim(\mathrm{pre}\,L)_f) \cap \mathcal{L} = (\lim \mathrm{pre}\,\mathcal{L}_f) \cap \mathcal{L}$. By I1, it follows that $(\lim \mathrm{pre}\,\mathcal{L}_f) \cap \mathcal{L} = (\lim \mathrm{pre}\,\mathcal{K}) \cap \mathcal{L} = \mathcal{K}$. Further, for all $s, s' \in \Sigma^*$, with $\mathrm{p_o}(s) = \mathrm{p_o}(s')$, it is $f(s) = f(s')$, according to Proposition 2.3.1. To show necessity, recall that I1 has been proven in Proposition 3.1.1. I2 and I3 are a direct consequence of $\mathrm{pre}\,\mathcal{L}_f = (\mathrm{pre}\,L)_f$ and L2, as well as L3, according to Proposition 2.3.1. □

To analyse the existence of a solution to the control problem, the union of all closed-loop behaviours $\mathcal{K}$ that solve the control problem is defined by

$$\mathcal{K}^{\Uparrow} := \bigcup \{\mathcal{K} \subseteq \mathcal{E} \mid \mathcal{K} \text{ satisfies I1 - I3 }\}.$$

However, the set of languages that satisfy I1 is only closed under finite set union, but not under arbitrary set union, as already commented in Subsection 3.1.1. Hence, in general, the existence of a unique supremal element that satisfies I1 - I3 is not given. Nevertheless, a unique supremal solution exists also in the case of partial observation, given that the specification $\mathcal{E}$ is relatively closed w.r.t. $\mathcal{L}$.

Lemma 3.1.9

It is given a system $\mathcal{L} \subseteq \Sigma^{\omega}$, $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{uo}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and a specification language $\mathcal{E} \subseteq \mathcal{L}$, that is relatively closed w.r.t. $\mathcal{L}$. Then, a unique supremal sub-language of $\mathcal{E}$ that satisfies I1 - I3 exists.

Proof. This Lemma is a direct consequence of Lemma 3.1.2 and the fact that the set of prefix-normal languages is non-empty and prefix-normality is retained under arbitrary unions. □

If the specification is relatively closed and if plant and specification languages are representable as limits of regular $*$-languages, then the supremal solution of the control problem can be characterised on the basis of the properties L0 - L3, see also Subsection 2.3.2.

Theorem 3.1.10

Consider a control problem given by a system $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{uo}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and a specification $\mathcal{E} \subseteq \mathcal{L}$. Assume that $\mathcal{E}$ is relatively topologically closed w.r.t. $\mathcal{L}$, and that both $\mathcal{L}$ and $\mathcal{E}$ are represented as limits of regular $*$-languages $\mathcal{L} = \lim L$, $\mathcal{E} = \lim E$, where $L \subseteq \Sigma^{*}$ is complete and $E = (\operatorname{pre}\mathcal{E}) \cap L$. Then, $\lim\sup[\mathrm{CNFO}](E)$ is the supremal solution $\mathcal{K}^{\Uparrow}$ to the control problem under partial observation.

Proof. To show $\lim\sup[\mathrm{CNFO}](E) \subseteq \mathcal{K}^{\Uparrow}$, consider an arbitrary $K \in [\mathrm{CNFO}](E)$ and denote $\mathcal{K} := \lim K$. By property L0, it is $\operatorname{pre}\mathcal{K} = \operatorname{pre} K$ and, hence, by Proposition 2.3.4 and properties L2 and L3, $\mathcal{K}$ satisfies the properties I2 and I3. Further, by property L1, it is $\operatorname{clo}\mathcal{K} \cap \mathcal{L} = (\lim \operatorname{pre} K) \cap (\lim L) = \lim(\operatorname{pre} K \cap L) = \lim K = \mathcal{K}$. Finally, by $K \subseteq E$, it is $\mathcal{K} \subseteq \mathcal{E}$. Consequently, $\lim\sup[\mathrm{CNFO}](E) \subseteq \mathcal{K}^{\Uparrow}$. Vice versa, by Lemma 3.1.9, $\mathcal{K}^{\Uparrow} \subseteq \mathcal{E}$ and $\mathcal{K}^{\Uparrow}$ satisfies I1 - I3. Further, by Proposition 3.1.8, there exists a non-blocking supervisor f such that $\mathcal{K}^{\Uparrow} = \lim \mathcal{L}_f$. Thus, by Proposition 2.3.3, it follows that $K := (\operatorname{pre} L)_f \cap L$ satisfies L0 - L3. By non-blocking of f, observe that furthermore

$K = (\mathrm{pre}\,L)_f \cap L = (\mathrm{pre}\lim L_f) \cap L = (\mathrm{pre}\,\mathcal{K}^{\Uparrow}) \cap L$. Finally, $K = (\mathrm{pre}\,\mathcal{K}^{\Uparrow}) \cap L \subseteq (\mathrm{pre}\,\mathcal{E}) \cap L = E$. This concludes the proof of $\mathcal{K}^{\Uparrow} \subseteq \lim\sup[\mathrm{CNFO}](E)$. □

Consequently, if in that particular case a non-empty solution to the given control problem exists, it is given by $\mathcal{K}^{\Uparrow} = \lim\sup[\mathrm{CNFO}](E)$. A formula for the computation of $\sup[\mathrm{CNFO}](E)$, is given in (Moor et al., 2012). Note that, due to natural projection operations and the construction of observer automata, the computational complexity is exponential in the number of plant and specification states.

However, in this thesis, the case of not necessarily relatively closed specifications is also considered, in particular in Chapter 4. Hence, an alternative characterisation for I1 - I3 is required to analyse the existence of solutions to the control problem.

Definition 3.1.6 ω-admissibility

Given a system $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{uo}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and the candidate $\mathcal{K} \subseteq \Sigma^{\omega}$, then, $\mathcal{K}$ is said to be *ω-admissible* w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathcal{L})$, if for all $s \in (\mathrm{pre}\,\mathcal{L}) \cap (\mathrm{pre}\,\mathcal{K})$, there exists a $\mathcal{V}_s \subseteq \mathcal{L} \cap \mathcal{K}$ with $s \in \mathrm{pre}\,\mathcal{V}_s$, and

C1 $\mathcal{V}_s$ is relatively topologically closed w.r.t. $\mathcal{L}$,

C2 $\mathrm{pre}\,\mathcal{V}_s$ is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathrm{pre}\,\mathcal{L})$,

C3 $\mathrm{pre}\,\mathcal{V}_s$ is prefix-normal w.r.t. $(\Sigma_{\mathrm{o}}, \mathrm{pre}\,\mathcal{L})$. □

It has to be analysed, whether this property is an alternative characterisation for I1 - I3. Therefore, it is first shown that any $\mathcal{K} \subseteq \mathcal{E}$ that satisfies I1 - I3 is indeed ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathcal{L})$.

Proposition 3.1.11

Given a system $\mathcal{L} \subseteq \Sigma^{\omega}$, with $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{uo}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and a candidate $\mathcal{K} \subseteq \Sigma^{\omega}$. If $\mathrm{pre}\,\mathcal{K}$ is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathrm{pre}\,\mathcal{L})$, prefix-normal w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathrm{pre}\,\mathcal{L})$ and $\mathcal{K}$ is relatively closed w.r.t. $\mathcal{L}$, then $\mathcal{K}$ is ω-admissible w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathcal{L})$.

Proof. To show ω-controllability, select an arbitrary $s \in (\mathrm{pre}\,\mathcal{L}) \cap (\mathrm{pre}\,\mathcal{K})$. Define $\mathcal{V}_s := \mathcal{K}$ and observe, that $\mathcal{V}_s$ satisfies C1 - C3 and $s \in \mathrm{pre}\,\mathcal{V}_s$. Further $\mathcal{V}_s = \mathcal{K} = \mathrm{clo}\,\mathcal{K} \cap \mathcal{L} = \mathcal{K} \cap \mathcal{L}$. Thus, $\mathcal{K}$ is ω-admissible w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathcal{L})$. □

Vice versa, it can be shown, that ω-admissibility guarantees the existence of a non-blocking supervisor f such that the control problem is solved.

Proposition 3.1.12

Given $\mathcal{L} \subseteq \Sigma^\omega$, with $\Sigma_{\text{uc}} \subseteq \Sigma$, $\Sigma_{\text{uo}} \subseteq \Sigma$, $\Sigma_{\text{c}} \subseteq \Sigma_{\text{o}}$, and $\mathcal{K} \subseteq \Sigma^\omega$, $\mathcal{K} \neq \emptyset$, such that $\mathcal{K}$ is ω-admissible w.r.t. $(\Sigma_{\text{uc}}, \Sigma_{\text{o}}, \mathcal{L})$. Then, there exists a non-blocking supervisor f such that $\mathcal{L}_f \subseteq \mathcal{K}$.

Proof. Assume that $\mathcal{K}$ exhibits the above properties. According to Definition 3.1.6, for every $s \in \operatorname{pre}\mathcal{K}$ there exists $\mathcal{V}_s \subseteq \mathcal{K}$, satisfying C1 - C3. By definition, $\varepsilon \in \operatorname{pre}\mathcal{K}$. Hence, there exists $\mathcal{V}_\varepsilon \subseteq \mathcal{K} \cap \mathcal{L}$ satisfying C1 - C3. In particular, $\mathcal{V}_\varepsilon$ satisfies I1 - I3, and by Proposition 3.1.8, there exists a non-blocking supervisor f such that $\mathcal{L}_f = \mathcal{V}_\varepsilon \subseteq \mathcal{K}$. □

The subset of ω-admissible languages of a given specification $\mathcal{E} \subseteq \Sigma^\omega$ is denoted

$$[\mathrm{C}^\omega_{\text{po}}](\mathcal{E}) := \{\mathcal{K} \subseteq \mathcal{E} \mid \mathcal{K} \text{ is } \omega\text{-admissible w.r.t. } (\Sigma_{\text{uc}}, \Sigma_{\text{o}}, \mathcal{L})\}.$$

Note that ω-admissibility is retained under arbitrary union. Hence, given a specification $\mathcal{E} \subseteq \mathcal{L}$, the supremal ω-admissible sub-language $\mathcal{K}^\Uparrow$ exists uniquely.

Lemma 3.1.13

Let $\Sigma_{\text{uc}} \subseteq \Sigma$, $\Sigma_{\text{uo}} \subseteq \Sigma$, $\Sigma_{\text{c}} \subseteq \Sigma_{\text{o}}$, $\mathcal{L} \subseteq \Sigma^\omega$, and consider a family of languages $\mathcal{K}_a \subseteq \Sigma^\omega$, $a \in A$, each one ω-admissible w.r.t. $(\Sigma_{\text{uc}}, \Sigma_{\text{o}}, \mathcal{L})$. Then, the union $\mathcal{K} := \cup_{a \in A} \mathcal{K}_a$ is ω-admissible, too.

Proof. Pick an arbitrary prefix $s \in (\operatorname{pre}\mathcal{L}) \cap (\operatorname{pre}\mathcal{K})$. By $\operatorname{pre}\mathcal{K} = \cup_{a \in A} \operatorname{pre}\mathcal{K}_a$, it is possible to choose $a \in A$ such that $s \in \operatorname{pre}\mathcal{K}_a$. Since $\mathcal{K}_a$ is considered ω-admissible, it is also possible to choose $\mathcal{V}_s \subseteq \mathcal{L} \cap \mathcal{K}_a$, $s \in \operatorname{pre}\mathcal{V}_s$ to satisfy C1 - C3. Clearly, $\mathcal{V}_s \subseteq \mathcal{L} \cap \mathcal{K}$, and ω-admissibility of $\mathcal{K}$ has been established. □

Hence, there exists a unique supremal element, which is denoted by $\sup[\mathrm{C}^\omega_{\text{po}}](\mathcal{E})$. Further, a unique supremal solution of the control problem exists iff $\sup[\mathrm{C}^\omega_{\text{po}}]$ is relatively closed w.r.t. $\mathcal{L}$. In particular, in that case, $\sup[\mathrm{C}^\omega_{\text{po}}](\mathcal{E})$ is the unique maximal solution.

3.1.3 Algorithmic solutions for infinite-string control problems

When it comes to practical applications, algorithms on the basis of finite-state automata are required that compute solutions to the presented control problems. For the case of control problems with $*$-languages, as well as ω-languages with relatively closed specifications, respective algorithms and useful references have been mentioned in Subsections 3.1.1 and 3.1.2. The purpose of this part of the thesis, is the illustration and the development of algorithms for the infinite string language control problem involving more general liveness properties. The basic concept follows the ideas in (Thistle and Wonham, 1992, 1994b), where an algorithm for the control problem under complete observation has been presented. Therefore, an alternative characterisation of ω-controllability, which was first introduced in (Thistle and Wonham, 1994b), is given in Subsection 3.1.3.1. In Subsection 3.1.3.2, an algorithm to solve the control problem is formulated on the basis of the *fixpoint theory*. In Subsection 3.1.3.3, the algorithm for the control problem under complete observation is illustrated, which has been developed originally in (Thistle and Wonham, 1992). The respective results are applied in Subsection 3.1.3.4 to develop a comparable heuristic algorithm for the control problem under partial observation.

3.1.3.1 Alternative ω-controllability characterisation

Following the ideas in (Thistle and Wonham, 1994a), an alternative definition of controllability in the context of ω-languages is introduced, which is later on used to characterize the algorithmic implementation. It is based on a particular prefix-characterisation, called *controllability-prefix.*

Definition 3.1.7 Controllability-prefix

For the languages $\mathcal{L} \subseteq \Sigma^\omega$, $\Sigma_o = \Sigma$, $\Sigma_{uc} \subseteq \Sigma$, and $\mathcal{K} \subseteq \Sigma^\omega$, the *controllability-prefix* $\mathrm{cpco}_{\mathcal{L}}(\mathcal{K})$ is defined as the set of strings $s \in \mathrm{pre}\,\mathcal{K}$ for which there exists $\mathcal{V}_s \subseteq \mathcal{K}/s$ with

A0 $\mathcal{V}_s \neq \emptyset$,

A1 $\mathcal{V}_s$ is relatively closed w.r.t. $\mathcal{L}/s$,

A2 $\mathrm{pre}\,\mathcal{V}_s$ is controllable w.r.t. $(\Sigma_{uc}, \mathrm{pre}(\mathcal{L}/s))$. □

The controllability-prefix represents the set of all strings in $\mathrm{pre}\,\mathcal{K}$ whose infinite extensions can be controlled to belong to $\mathcal{K}$. In particular the controllability-prefix is an alternative characterisation of the ω-controllability according to Definition 3.1.4.

Proposition 3.1.14

Given the languages $\mathcal{L} \subseteq \Sigma^\omega$, with $\Sigma_{uc} \subseteq \Sigma$ and $\Sigma_o = \Sigma$, and $\mathcal{K} \subseteq \Sigma^\omega$, then $\mathcal{K}$ is ω-*controllable w.r.t.* $(\Sigma_{uc}, \mathcal{L})$, if and only if $\mathrm{cpco}_{\mathcal{L}}(\mathcal{K}) = \mathrm{pre}\,\mathcal{K}$.

Proof. To show sufficiency, select an arbitrary $s \in (\mathrm{pre}\,\mathcal{L}) \cap (\mathrm{pre}\,\mathcal{K})$ and show that there exists $\mathcal{V}_s \subseteq \mathcal{L} \cap \mathcal{K}$ such that $s \in \mathrm{pre}\,\mathcal{V}_s$ and C1 and C2 are satisfied. The set of prefixes of s are defined as $T := \{t \in \Sigma^* \mid t \leq s\}$. Observe that $s \in \mathrm{cpco}_{\mathcal{L}}(\mathcal{K})$ and $\mathrm{pre}\,\mathcal{K} = \mathrm{cpco}_{\mathcal{L}}(\mathcal{K})$ implies that $t \in \mathrm{cpco}_{\mathcal{L}}(\mathcal{K})$. Hence, for every $t \in T$ there exists $\mathcal{V}_t \subseteq (\mathcal{L} \cap \mathcal{K})/s$ such that A0 - A2. Based on this considerations, the candidate $\mathcal{V}_s := \bigcup_{t \in T} t\mathcal{V}_t$ is defined for s. Observe that $s \in \mathrm{pre}\,\mathcal{V}_s$ and $\mathcal{V}_s \subseteq \mathcal{L} \cap \mathcal{K}$ by definition. Regarding C1, observe that $\mathcal{V}_s \subseteq (\lim \operatorname{pre} \mathcal{V}_s) \cap \mathcal{L}$ is always true. To show the reverse direction, pick $w \in (\lim \operatorname{pre} \mathcal{V}_s) \cap \mathcal{L}$. Then, there exists $t \in T$ such that $w \in \mathcal{L} \cap (\lim \operatorname{pre} t\mathcal{V}_t)$. Since t is of finite length, the existence of an increasing sequence $(v_\text{n}) \in \operatorname{pre} \mathcal{V}_t$ such that $t \lim(v_\text{n}) = w$ can be deduced. By A1 follows that $\lim(v_\text{n}) \in (\lim \operatorname{pre} \mathcal{V}_t) \cap \mathcal{L}/t = \mathcal{V}_t$. Hence, $w \in t\mathcal{V}_t \subseteq \mathcal{V}_s$. Regarding C2, select an arbitrary $\hat{s} \in \operatorname{pre} \mathcal{V}_s$ and $\sigma \in \Sigma_\text{uc}$ such that $\hat{s}\sigma \in \operatorname{pre} \mathcal{L}$. Since $\hat{s} \in \operatorname{pre} \mathcal{V}_s$, there exists $t \in T$ such that $\hat{s} \in t \operatorname{pre} \mathcal{V}_t$. Due to controllability of $\operatorname{pre} \mathcal{V}_t$ w.r.t. $(\Sigma_\text{uc}, \operatorname{pre} \mathcal{L}/t)$, it follows that $\hat{s}\sigma/t \in \operatorname{pre} \mathcal{V}_t$. Thus, $\hat{s}\sigma \in \operatorname{pre} \mathcal{V}_s$. Thus, sufficiency has been shown. To show necessity, pick an arbitrary $s \in \operatorname{pre} \mathcal{K}$ and show that there exists $\mathcal{V}_s \subseteq \mathcal{K}/s$ such that A0 - A2 are satisfied. Due to ω-admissibility, according to Definition 3.1.4, there exists $\hat{\mathcal{V}}_s$ satisfying C1 and C2. The candidate $\mathcal{V}_s := \hat{\mathcal{V}}_s/s$ is selected. Since $\hat{\mathcal{V}}_s \subseteq \Sigma^\omega$ and $s \in \operatorname{pre} \hat{\mathcal{V}}_s$, there exists $v \in \Sigma^\omega$ such that $v \in \mathcal{V}_s$ and, hence, $\mathcal{V}_s \neq \emptyset$, satisfying A0. A1 and A2 follow directly from results of Proposition 4.3 in (Thistle and Wonham, 1994a) regarding controllability and relative closure under quotient operator. Finally, equality of both properties has been shown. $\square$

The particular definition of the controllability-prefix can now be used to define a procedure to compute a solution of the control problem under complete observation.

3.1.3.2 Fixpoint theory

For the elaboration of a synthesis procedure a special notation for extremal fixpoints of monotone operators on lattices is used in (Thistle and Wonham, 1994a), inspired by its application in the area of model checking in (Emerson and Lei, 1986). This particular framework, the so called *μ-Calculus*, has been developed in order to test automata for different properties. Some properties can be tested state by state. More involved properties have to be tested by a complex tree analysis. The fixpoint theory offers a framework for a respective recursive computation procedure. Hence, basic concepts of the fixpoint theory are illustrated in the following.

An operator $f : (2^X)^k \to 2^X$ is said to be monotone if, for $1 \leq i \leq k$ and any subsets $X_i, X_i' \subseteq X$,

$$X_i \subseteq X_j \quad \Rightarrow \quad f(X_1, ..., X_i, ..., X_k) \subseteq f(X_1, ..., X_i', ..., X_k),$$

i.e. it preserves inclusion. Further, the operator f is said to be $\bigcup$*-continuous*, if for any i, with $1 \leq i \leq k$, and a non-decreasing sequence $X_i^0 \subseteq X_i^1 \subseteq X_i^2 ...$, it is

$$\bigcup_{j=0}^{\infty} f(X_1, ..., X_i^j, ..., X_k) = f(X_1, ..., \bigcup_{j=0}^{\infty} X_i^j, ..., X_k).$$

The operator f is said to be $\bigcap$-*continuous*, if for any i, with $1 \leq i \leq k$, and a non-increasing sequence $X_i^0 \supseteq X_i^1 \supseteq X_i^2 ...$, it is

$$\bigcap_{j=0}^{\infty} f(X_1, ..., X_i^j, ..., X_k) = f(X_1, ..., \bigcap_{j=0}^{\infty} X_i^j, ..., X_k).$$

According to (Emerson and Lei, 1986), both continuities imply monotonicity. Further, for a finite set X any monotone operator is $\bigcup$-continuous and $\bigcap$-continuous. From the results in (Tarski, 1955) follows that such an operator has *least* and *greatest fixpoints*. The *least fixpoint* $Y \subseteq X$ of an operator such that $f(Y) = Y$ is denoted by $\mu Y.f(Y)$. The *greatest fixpoint* $Y \subseteq X$ of an operator such that $f(Y) = Y$ is denoted by $\nu Y.f(Y)$, respectively. According to (Tarski, 1955), these solutions are known to satisfy the following properties.

Proposition 3.1.15 (Tarski-Knaster)

Let $f : 2^X \rightarrow 2^X$ be a monotone operator. Then f has least and greatest fixpoints, i.e.

(a) $\mu Y.f(Y) = \bigcap \{Y' \subseteq X : Y' = f(Y)\} = \bigcap \{Y' \subseteq X : Y' \supseteq f(Y')\}$,

b) $\nu Y.f(Y) = \bigcup \{Y' \subseteq X : Y' = f(Y)\} = \bigcup \{Y' \subseteq X : Y' \subseteq f(Y')\}$,

c) If f is $\bigcup$-continuous, then $\mu Y.f(Y) = \bigcup_{i=0}^{\infty} f^i(\emptyset)$,

d) If f is $\bigcap$-continuous, then $\nu Y.f(Y) = \bigcap_{i=0}^{\infty} f^i(X)$. □

Given a finite set X and a monotone operator f, the least fixpoint of f can be obtained by a recursive iteration starting with $X_0 = \emptyset$ and computing $X_{i+1} = f(X_i)$ until $X_j = X_{j-1}$ for some j. The greatest fixpoint can be received by a similar iteration, but starting with $X_0 = X$. Hence, the greatest fixpoint can be used to synthesise the greatest set of states, where some property is always satisfied. Whereas the least fixpoint determines the greatest set of states, from where some property is always reached eventually.

3.1.3.3 Algorithmic solution in the case of complete observation

For the computation of a solution to the control problem in the case of complete event observation an iterative procedure is proposed in (Thistle and Wonham, 1992) that computes a representation of the controllability-prefix (according to Definition 3.1.7). For this computation, it is convenient to represent all necessary information in just one automaton. In Thistle and Wonham (1992) a so called Rabin-Büchi automaton is used, representing two liveness properties. The first one is the liveness property of the plant itself, given in form of the Büchi acceptance criterion. The second one represents the eventuality properties of the specification, given in form of a Rabin acceptance criterion. However, in this thesis only Büchi automata are considered. Therefore, just another Büchi acceptance criterion is used for the eventuality properties of the specification and the *extended Büchi automaton* is defined as follows.

Definition 3.1.8 Extended Büchi automaton

An *extended Büchi automaton* is given as a six-tuple $\mathrm{A} := (X, \Sigma, \delta, x_\mathrm{o}, X_\mathrm{K}, X_\mathrm{L})$, where X is the set of states, Σ is the alphabet of events, $\delta : X \times \Sigma \to X$ is a transition function, x_o the initial state, X_K the set of marked states according to the specification $\mathcal{E} \subseteq \Sigma^\omega$ and X_L the set of marked states according to the respective plant $\mathcal{L} \subseteq \Sigma^\omega$. □

The extended Büchi automaton represents two different ω-languages. The first one is the ω-language accepted by the Büchi automaton $\mathrm{A_K} := (X, \Sigma, \delta, x_\mathrm{o}, X_\mathrm{K})$, given by

$$\mathcal{L}(\mathrm{A_K}) := \{w \in \Sigma^\omega |\ \mathrm{Inf}\,(w) \cap X_\mathrm{K} \neq \emptyset\}.$$

It represents a liveness property associated with the given specification. Further, the extended Büchi automaton represents the ω-language accepted by the Büchi automaton $\mathrm{A_L} := (X, \Sigma, \delta, x_\mathrm{o}, X_\mathrm{L})$,

$$\mathcal{L}(\mathrm{A_L}) := \{w \in \Sigma^\omega |\ \mathrm{Inf}\,(w) \cap X_\mathrm{L} \neq \emptyset\},$$

which is used to represent the liveness properties of the plant. Given two Büchi automata $\mathrm{A_L}$ and $\mathrm{A_K}$, representing plant and specification language, it is straight forward to receive an extended Büchi automaton, applying the synchronous ω-composition operation in Definition 2.2.8 to $\mathrm{A_K}$ and $\mathrm{A_L}$ and tracking the set of states in the resulting automaton A that contain a marked state of $\mathrm{A_L}$. For the discussion of strings that do not begin in the initial state, define $\mathrm{A_x}$ to be the automaton obtained by setting the initial state of the automaton A to x. Following (Thistle and Wonham, 1992), the subset of controllable states under complete event observation is defined.

Definition 3.1.9 Controllability state subset under complete observation

Given an extended Büchi automaton $\mathrm{A} := (X, \Sigma, \delta, x_\mathrm{o}, X_\mathrm{K}, X_\mathrm{L})$, with $\mathrm{A_L} = (X, \Sigma, \delta, x_\mathrm{o}, X_\mathrm{L})$ and $\mathrm{A_K} = (X, \Sigma, \delta, x_\mathrm{o}, X_\mathrm{K})$, the *controllability state subset* F^A is defined as the set of all $x \in X$ for which there exists a supervisor $f^\mathrm{A} : \Sigma^* \to \Gamma$ such that

(i) for any $s \in \Sigma^*$ generated by $\mathrm{A_x}$ under f^A, there exists $v \in \Sigma^\omega$ such that sv is accepted by $\mathrm{A_{L,x}}$ under f^A,

(ii) every $w \in \Sigma^\omega$ accepted by $\mathrm{A_{L,x}}$ under f^A is accepted by $\mathrm{A_{K,x}}$. □

The controllability state subset represents the set of states from where the automaton can be controlled to satisfy the liveness properties given by the specification. Based on the μ-Calculus approach, in (Thistle and Wonham, 1992) and (Thistle, 1995) an algorithm for the computation of the controllability state subset is developed. For convenience, the algorithm is simplified here to the control problem given by the extended Büchi automaton instead of treating Rabin-Büchi automata. Basically, the fix-point calculus is a recursive procedure applying in each step particular operators, either in order to compute a greatest or a least fix-point. The algorithm in (Thistle and Wonham, 1992) is a nested construct of several such operators computing greatest and least fix-points. The fundamental and innermost operator that analyses the given states regarding a particular controllability property, is the *one-step operator*.

Definition 3.1.10 One-Step operator

Let $\mathrm{A} := (X,\Sigma,\delta,x_o,X_\mathrm{K},X_\mathrm{L})$ be an extended Büchi automaton. The *one-step operator* $\Theta^\mathrm{A} : (2^X)^2 \to 2^X$ is given by

$$\Theta^\mathrm{A}(X_1,X_2) := \{x \in X \mid (\exists \gamma \in \Gamma)[(\exists \sigma \in \gamma)[\delta(x,\sigma)! \text{ and } \delta(x,\sigma) \in X_1] \text{ and } (\forall \sigma \in \gamma)[\delta(x,\sigma)! \Rightarrow \delta(x,\sigma) \in X_1 \cup X_2)]]\}. \qquad \square$$

Note that $\Theta(X_1,X_2)$ is a monotone operator, i.e. it preserves inclusion. This operator forms the core of the algorithm in the inner-most loop. It determines the state subset from which the automaton can be controlled to enter a given domain $X_1 \cup X_2$ by the next transition without being prevented from entering a given target state set X_1. The next operator, the *inverse dynamics operator*, calls the one-step operator and iterates the target, as well as the domain state set.

Definition 3.1.11 Inverse dynamics operator

Let $\mathrm{A} := (X,\Sigma,\delta,x_o,X_\mathrm{K},X_\mathrm{L})$ be an extended Büchi automaton. The *inverse dynamics operator* $\tilde{\Theta} : 2^X \to 2^X$ is given by

$$\tilde{\Theta}(X_1) := \nu X_2.\mu X_3.[\Theta(X_1 \cup (X_3 \backslash X_\mathrm{L}), X_2 \backslash X_\mathrm{L})]. \qquad \square$$

The inverse dynamics operator determines the set of states from which the automaton can be controlled to enter the target state set no later than passing a plant marking X_L. The last operator is the *reachability operator*.

Definition 3.1.12 Reachability operator

Let $\mathrm{A} := (X,\Sigma,\delta,x_o,X_\mathrm{K},X_\mathrm{L})$ be an extended Büchi automaton. The *reachability operator* $\Phi : 2^X \to 2^X$ is given by

$$\Phi(X_1) := \mu X_2.[\tilde{\Theta}(X_1 \cup X_2)]. \qquad \square$$

The difference between this operator and the underlying operator is the addition of $X_L \backslash X_K$ states to the target state set, given that they satisfy the condition in the innermost loop. Hence, the set of states is determined from which the automaton can be controlled to reach the target states after passing no more than finitely often states in $X_L \backslash X_K$. In the last step, the fix-point characterization of the controllability state set is given.

Definition 3.1.13 Fix-point characterization of the controllability state set

Let $\mathrm{A} := (X, \Sigma, \delta, x_o, X_K, X_L)$ be an extended Büchi automaton. Then, the fix-point characterization of the controllability state set $C^{\mathrm{A}} \subseteq X$ is given by

$$C^{\mathrm{A}} := \mu X_1 . \nu X_2 . [\Phi(X_1 \cup (X_2 \cap X_K))]. \qquad \square$$

According to (Thistle and Wonham, 1992), this algorithm computes the controllability state subset under complete observation.

Theorem 3.1.16 (Thistle and Wonham, 1992)

Given an extended Büchi automaton $\mathrm{A} := (X, \Sigma, \delta, x_o, X_K, X_L)$ accepting $\mathcal{L} \subseteq \Sigma^\omega$ by $(X, \Sigma, \delta, x_o, X_L)$ and $\mathcal{E} \subseteq \Sigma^\omega$ by $(X, \Sigma, \delta, x_o, X_K)$. Then, $C^{\mathrm{A}} = F^{\mathrm{A}}$. $\square$

Further information regarding detailed proofs are given in (Thistle and Wonham, 1992). For the context of this thesis, the particular construction of a state feedback map is of further interest, since it can be used to derive a supervisor that solves the control problem under complete observation.

Proposition 3.1.17 (Thistle and Wonham, 1992)

Given an extended Büchi automaton $\mathrm{A} := (X, \Sigma, \delta, x_o, X_K, X_L)$ accepting $\mathcal{L} \subseteq \Sigma^\omega$ by $(X, \Sigma, \delta, x_o, X_L)$ and $\mathcal{E} \subseteq \Sigma^\omega$ by $(X, \Sigma, \delta, x_o, X_K)$. Then, there exists a feedback map $\phi^{\mathrm{A}} : C^{\mathrm{A}} \to \Gamma$ such that for any $x \in C^{\mathrm{A}}$

(i) for any $w \in \Sigma^\omega$ and any path $\pi : \mathrm{pre}\, w \to X$ of w in A_x, the conditions $\forall s\sigma \in \mathrm{pre}\, w$:
$\sigma \in \phi^{\mathrm{A}}(\pi(s))$ and $\mathrm{Inf}(w) \cap X_L \neq \emptyset$ together imply $\mathrm{Inf}(w) \cap X_K \neq \emptyset$, and

(ii) there exists $v \in \Sigma^\omega$ and a path $\pi : \mathrm{pre}\, v \to X$ of v in A_x such that $\forall s\sigma \in \mathrm{pre}\, v$:
$\sigma \in \phi^{\mathrm{A}}(\pi(s))$ and $\mathrm{Inf}(v) \cap X_L \neq \emptyset$. $\square$

The particular state feedback map, can be used to construct an automaton, that represents a solution to the ω-language control problem under complete observation; see also (Thistle and Wonham, 1994a) for a similar construction.

In the discrete event system library (libFAUDES, 2015), the respective algorithm has been implemented. In particular, two different versions are available. The first one computes the supremal ω-controllable sub-language according to the algorithm in (Thistle and Wonham, 1994a). The algorithm is implemented as a nested program in accordance with the definition given as a μ-Calculus. Hence, it can be split in several sub-routines for the different operators that have been introduced before. In the following, each sub-routine is considered by itself and described by pseudo code for simplification reasons. Further, it is assumed that an extended Büchi automaton $\mathrm{A} := (X, \Sigma, \delta, x_o, X_\mathrm{K}, X_\mathrm{L})$ is given. For the representation of the algorithm, define $\mathrm{A} :=$ `Candidate`, $X :=$ `full`, $X_\mathrm{K} :=$ `CandMarking`, $X_\mathrm{L} :=$ `PlantMarking`, $\Sigma_\mathrm{c} :=$ `CAlph` and $|X| :=$ `fsz`.

Fix-point characterization of the controllability state set

The fix-point characterization of the controllability state set $C^\mathrm{A} \subseteq X$ is given by the mu-nu-iteration

$$C^\mathrm{A} := \mu X_1 . \nu X_2 . [\Phi(X_1 \cup (X_2 \cap X_\mathrm{K}))].$$

The μ-formula can be represented by an iteration starting with $X_{1,0} = \emptyset$ and iterating

$$X_{1,i+1} := X_{1,i} \cup \nu X_2 . [\Phi(X_{1,i} \cup (X_2 \cap X_\mathrm{K}))],$$

until $X_{1,i+1} = X_{1,i}$. The inner ν-calculus can be computed by an iteration starting with all candidate states $X_{2,0} := X$ and iterating

$$X_{2,i+1} := X_{2,i} \cap \Phi(X_{1,i} \cup (X_{2,i} \cap X_\mathrm{K})),$$

until $X_{2,i+1} = X_{2,i}$.

Define $X_1 :=$ `resolved` and $X_2 :=$ `initialK`. Start with `resolved` $= \emptyset$ [2] and iterate [3]

$$\texttt{resolved} = \texttt{resolved} + \nu X_2 . [\Phi(\texttt{resolved} \cup (X_2 \cap \texttt{CandMarking}))],$$

with the inner loop starting with `initialK = full` and iterating

$$\texttt{initialK} = \texttt{initialK} \cap \Phi(\texttt{resolved} \cup (\texttt{initialK} \cap \texttt{CandMarking})).$$

As given in Figure 3.2, the iterations are implemented by two loops. The breaking condition for the inner loop is either that no more states are available in `initialK`

[2] Note that the function `Clear()` deletes all elements of the respective set.

[3] Note that the function `Insert(X)` inserts all given elements in `X` into the respective set.

or that the number of states in `initialK` is not changing any more.[4] The breaking condition for the outer loop is either that maximum number of states (`fsz`) has been reached or that the number of states in `resolved` is not changing any more. At the end of the algorithm, all states that have not been found and stored in `resolved` are deleted from the `Candidate` state set by the funktion `DelStates( )`.

```
resolved.Clear();

do  \\ mu-iteration
  rsz = resolved.Size();
  initialK = full;

  do  \\ nu-iteration
    Idx iKsz = initialK.Size();
    targetLstar = (initialK * CandMarking) + resolved;
    initialK.RestrictSet(Phi(targetLstar));
  while((initialK.Size()!=iKsz) and (initialK.Size()!=0))

  resolved.InsertSet(initialK);
while((resolved.Size()!=rsz) and (resolved.Size()!=fsz))

Candidate.DelStates(full - resolved);
```

Figure 3.2: Algorithmic implementation of the characterization of the controllability state set

Reachability operator

In this part of the algorithm, the formula of the reachability operator

$$\Phi(X_1) = \mu X_2.[\tilde{\Theta}(X_1 \cup X_2)]$$

has to be evaluated. The respective iteration is given by $X_{2,0} = \emptyset$ and

$$X_{2,i+1} = X_{2,i} \cup \tilde{\Theta}(X_1 \cup X_{2,i}),$$

until $X_{2,i+1} = X_{2,i}$. Define $X_1 :=$ `targetLstar` and $X_2 :=$ `initialL`. Start with `initialL` $= \emptyset$ and iterate `initialL = initialL + thetaTilde(targetLstar + initialL)`. The iteration continues until either the maximal number of states (`fsz`) of the candidate is reached or the size of `initialL` is not increasing any more. The implementation of the reachability operator is given in Figure 3.3.

[4] Note that the function `Size()` determines the number of elements of the respective set.

```
initialL.Clear();

do  \\ mu-iteration
  Idx iLsz = initialL.Size();
  targetL = targetLstar + initialL;
  initialL.InsertSet(thetaTilde(targetL));

while((initialL.Size()!=iLsz) and (initialL.Size()!=fsz))
```

Figure 3.3: Algorithmic implementation of the reachability operator

Inverse dynamics operator

In this part of the algorithm, a combination of a ν- and a μ-calculus has to be evaluated. Observe that the ν-calculus in

$$\tilde{\Theta}(X_1) := \nu X_2.\mu X_3.[\Theta(X_1 \cup (X_3 \backslash X_{\mathrm{L}}), X_2 \backslash X_{\mathrm{L}})]$$

can be computed by an iteration starting with all candidate states $X_{2,0} := X$ and iterating

$$X_{2,i+1} := X_{2,i} \cap \mu X_3.[\Theta(X_1 + (X_3 - X_{\mathrm{L}}), X_{2,i} - X_{\mathrm{L}})],$$

until $X_{2,i+1} = X_{2,i}$. The inner μ-calculus

$$\mu X_3.[\Theta(X_1 + (X_3 - X_{\mathrm{L}}), X_{2,i} - X_{\mathrm{L}})]$$

can be represented by the iteration starting with $X_{3,0} = \emptyset$ and iterating

$$X_{3,j+1} := X_{3,j} \cup \Theta(X_1 + (X_{3,j} - X_{\mathrm{L}}), X_{2,i} - X_{\mathrm{L}}),$$

until $X_{3,i+1} = X_{3,i}$.

Define $X_1 :=$ `targetL`, $X_2 :=$ `domainL`, $X_3 :=$ `target1` and $X_L :=$ `PlantMarking`. Start with `domainL:=full` and iterate

`domainL` $=$ `domainL` $\cap$ $\mu X_3.[\Theta($`targetL` $+$ $(X_3$ $-$ `PlantMarking`$),$`domainL` $-$ `PlantMarking`$)],$

with an inner μ-iteration, starting with `target1:=`$\emptyset$ and iterating

`target1` $=$ `target1` $+$ `theta`$($`targetL` $+$ $($`target1` $-$ `PlantMarking`$),$`domainL` $-$ `PlantMarking`$)].$

The implementation of the reachability operator is given in Figure 3.4.

```
domainL = full;

do   \\ nu-iteration
  dLsz = domainL.Size();
  domain = domainL - PlantMarking;
  target1.Clear();

    do   \\ mu-iteration
      t1sz = target1.Size();
      target = targetL + (target1 - PlantMarking);
      target1.InserSet(theta(target ,domain));
    while((target1.Size()!=t1sz) and (target1.Size()==fsz))

  domainL = RestrictSet(target1);

while((domainL.Size()!=dLsz) and (domainL.Size()!=0))
```

Figure 3.4: Algorithmic implementation of the inverse dynamics operator

One-Step operator

The innermost loop of the algorithm, contains the one-step operator

$$\Theta(X_1,X_2) := \{x \in X \mid (\exists\gamma \in \Gamma)[(\exists\sigma \in \gamma)[\delta(x,\sigma)! \text{ and } \delta(x,\sigma) \in X_1] \text{ and } (\forall\sigma \in \gamma)[\delta(x,\sigma)! \Rightarrow \delta(x,\sigma) \in X_1 \cup X_2)]]\}.$$

The respective implementation is given in Figure 3.5.

```
theta.Clear();

for(iteration over all states) {
  pass = false;  fail = false;

  for(iteration over all transitions of current state) {
    if(transition ends in target) {pass = true; continue;}
    if(transition ends in domain) continue;
    if(transition event not in CAlph)) {fail = true; break;}
  }
  if(pass && !fail) theta.Insert(current state);
}
```

Figure 3.5: Algorithmic implementation of the one-step operator

It has to be iterated over all candidate states and all transitions for each state. By `theta`, the state set is defined that collects all states that fulfil the condition of the one-step operator. Further define $X_1 := $ `target` and $X_2 := $ `domain`. In order to pass the evaluation, three conditions have to be analysed. The first condition tests if the transition ends in the `target`. At least one transition per state has to end in the `target`. If the condition is satisfied, the variable `pass` is set to `true`. The last two `if`-conditions test whether all other transitions end in the `domain` and if not, whether the transition is controllable. If both are not satisfied, the variable `fail` is set to `true`, which indicates that the state does not satisfy the condition of the one-step operator. If the state satisfies all conditions, it is added to `theta`.

The second algorithm in (libFAUDES, 2015) is similar to the first one, with the difference that it records for each analysed state $x \in X$ those events Σ that have to be enabled only finitely often in order to guarantee that the liveness properties of the given specification are satisfied. This information is later on used to construct a particular solution of the control problem. Observe that enabling those tracked events only finitely often also includes not enabling them at all and deleting them from the control pattern for the respective state. Using this idea, a feedback map is constructed that maps to each state those events that have to be disabled eventually. The respective implementation in the one-step operator is given in Figure 3.6.

```
theta.Clear();

for(iteration over all states) {
  pass = false; fail = false;
  disable.Clear();

  for(iteration over all transitions of current state) {
    if(transition ends in target) {pass = true; continue;}
    if(transition ends in domain) continue;
    if(transition event not in CAlph)) {fail = true; break;}
    disable.Insert(event of current transition);
  }
  if(pass && !fail){
    theta.Insert(current state);
    if(current state is not yet listed in controls1)
      controls1[current state] = disable;
  }
}
```

Figure 3.6: Alg. implementation of the one-step operator recording eventually disabled events

To record those events, it is defined the event set `disable`. An event σ is inserted in `disable` if $\sigma \in \Sigma_c$, $\delta(x,\sigma)!$ and $\delta(x,\sigma) \notin$ `target` $\cup$ `domain`. When all transitions for a state x have been analysed and the state satisfies the conditions of the one-step operator, the events in `disable` are listed together with the state x in the state feedback map `controls1`. The feedback map is only updated when the state satisfies the condition of the one-step operator for the first time, i.e. when it it is not yet listed in `controls1`. Further, the construction of the feedback map has to be aligned with the construction of the algorithm itself and with the state sets that are found during the different iteration steps. Hence, each time `target1` is cleared and set to `target1=∅`, the feedback map `controls1` is also cleared. Further, after finishing the computation of the inverse dynamics operator, i.e. when updating `initialL` by the result of the inverse dynamics operator by `initialL = initialL + thetaTilde(targetL)`, see also Figure 3.4, `controls1` is merged to a superordinate feedback map `controls1L`. Pseudo code of the respective implementation is given in Figure 3.7.

```
for(iteration over all states x in controls1) {
  if(current x is not yet listed in controls1L)
    controls1L[current x] = disabled events of current x in
                            controls1;
}
```

Figure 3.7: Implementation of transferring feedback map `controls1` to `controls1L`

The same procedure has to be done for `control1L`. When `initialL` is cleared, then `control1L` has to be cleared. Further, `control1L` is transferred to the superordinate feedback map `control1X`, when `resolved` is updated, i.e. `resolved = resolved + initialK`, see Figure 3.2. At the end of the algorithm, `control1X` is used to construct a control feedback map for all states in C^{A}. Therefore, only those events are enabled which are either uncontrollable, or controllable and active according to the candidate, but not to be disabled according to `control1X`. Applying this control feedback map, a particular solution can be constructed.

3.1.3.4 Algorithmic solution in the case of partial event observation

For the following considerations, the assumption of complete event observation is dropped. To our best knowledge, this situation has only been addressed in (Thistle and Lamouchi, 2009). Therein, an algorithm has been presented that solves the control problem under partial observation. However, the algorithm is not applicable to a pure Büchi automata framework and the plant is considered to have no eventuality properties. For both reasons, it is not applicable to the framework used in this thesis and an algorithm has to be developed that computes a solution for the particular situation that is considered here. In the following, an algorithm is proposed that is strongly connected to the results presented just before to solve the control problem under complete event observation. A similar fix-point iteration is elaborated that considers the restrictions in the control actions due to partial event observation.

Treating algorithms for control problems with partial observation, an appropriate state space for the computation has to be constructed; see also Cho and Marcus (1989a) and Cho and Marcus (1989b). A fundamental role plays the ability to divide the constructed state space into appropriate, disjoint equivalence classes. A basic instrument is applied, which is often used in control problems with partial event observation and which provides an appropriate state partition, the so called *state-partition automaton*. It is given by the synchronous composition of the original automaton with it's *observer automaton* (Cassandras and Lafortune, 2008). A more detailed description of the state-partition automaton can also be found in (Cho and Marcus, 1989a) or (Cho and Marcus, 1989b), where this kind of automaton is called *M-recognizable*[5]. Further, it is defined the set of states $X_{\mathrm{Obs}}(r) := \{x \in X \mid x \in \delta(x_{\mathrm{o}}, s), \mathrm{p_o}s = r\}$, which can be reached in the automaton by strings that are equal under observation. If the given automaton G is a state-partition automaton, then for all $r_1, r_2 \in \mathrm{p_o}L(G)$, it is $X_{\mathrm{Obs}}(r_1) \cap X_{\mathrm{Obs}}(r_2) = \emptyset$, whenever $X_{\mathrm{Obs}}(r_1) \neq X_{\mathrm{Obs}}(r_2)$. Hence, in that case, $X_{\mathrm{Obs}}(r)$ defines equivalence classes in X. As a consequence, for every state $x \in X$, there exists an unambiguous mapping and, thus, pairwise disjunct state sets as states of the projected automaton. For further interesting property analysis of the state-partition automaton see (Jirásková and Masopust, 2012). It is further defined for any automaton $\mathrm{A} := (X, \Sigma, \delta, x_{\mathrm{o}}, X_{\mathrm{L}})$ and any state $x \in X$, the set of states that are reached by indistinguishable strings as

$$\mathrm{Obs}(x) := \{x' \in X \mid (\exists s, s' \in \Sigma^*)[\delta(x_o, s) = x \text{ and } \delta(x_{\mathrm{o}}, s') = x' \text{ and } \mathrm{p_o}s = \mathrm{p_o}s']\}.$$

Since $\mathrm{Obs}(x)$ is the set of states that cannot be distinguished from x, equal control pattern have to be enabled for all those states. This idea is included in the following considerations for the computation of the controllability state subset under partial event observation. Hence, the input automaton is assumed to be a state partition automaton.

The computation of the ω-admissible sub-language that is implemented in (libFAUDES, 2015) is an extension of the computation of the ω-controllable sub-language, described in Subsection 3.1.3.3. The implementation is still experimental and does possibly not compute the supremal ω-admissible sub-language. It is based on the considerations of recording events in a state feedback map, which have to be enabled only finitely often in order to guarantee the liveness properties of the given specification. The variant developed here records those control patterns and only accepts a new state if the corresponding pattern complies with all recorded patterns that correspond to the same observation. Hence, the result is a subset of the ω-controllable sub-language.

[5] In (Cho and Marcus, 1989a), the observation function is represented by the *mask M*.

The algorithmic implementation regarding the μ-/ν-iterations is similar to the one given in Subsection 3.1.3.3 and according to the μ-calculus given therein. In addition, a procedure is implemented to record feedback maps `controls1`, `controls1L` and `controls1X`, based on the ideas described in Subsection 3.1.3.3, which can be used to construct a particular solution of the control problem. During the construction of the state partition automaton, a map is constructed that defines for each state x in the `candidate` the respective state in the observer automaton. In the algorithm, this map is called `OberverStatesMap`. The feedback maps `controls1`, `controls1L` and `controls1X` of the algorithm for partial event observation are not recording the feedback for each state x, but for the respective states `x_ObsG = ObserverStatesMap[current state x]` in the observer automaton. Further, the feedback map is listing not only events that shall be disabled, but also events that have to be enabled eventually. In particular, it records for each state x the set of events from which at least one event has to be enabled. This information is used in a next step to compare

```
theta.Clear();  controlsT.clear();

for(iteration over all candidate states x) {
  x_ObsG = ObserverStatesMap[current state x];
  pass = false;
  fail = false;
  disable.Clear();  enable.Clear();

  for(iteration over all transitions of current state x) {
     if(transition ends in target)
       {enable.Insert(event of current transition);
        pass = true; continue;}
     if(transition ends in domain) continue;
     if(transition event not in CAlph){ fail = true; break;}
     disable.Insert(event of current transition);   }

  if(pass && !fail) {//initialize with existing patterns
    if(x_ObsG is not yet listed in controlsT) {
       if(x_ObsG is already listed in controls1)
          controlsT[x_ObsG].merge(controls1[x_ObsG]);
       if(x_ObsG is already listed in controls1L)
          controlsT[x_ObsG].merge(controls1L[x_ObsG]);
       if(x_ObsG is already listed in controls1X)
          controlsT[x_ObsG].merge(controls1X[x_ObsG]); }
    controlsT[x_ObsG].disable_all.InsertSet(disable);
    controlsT[x_ObsG].enable_one.insert(enable)  }
}
```

Figure 3.8: Find new candidate states and accumulate required controls

the set of enabled and disabled events for all states in Obs(x) and to find a common control pattern. The test whether a state satisfies all necessary conditions is therefore extended into three steps. In the first step, possible candidate states are determined and respective control pattern are recorded in the additional feedback map `controlsT`, used only in the innermost loop and cleared, every time `theta` is cleared, see Figure 3.8.

In a second step, the respective control pattern `controlsT` is tested for conflicts with existing control pattern found for other states in Obs(x). If no conflict exists, the control pattern is accepted and added (`merge()`) to `controls1`, see Figure 3.9.

```
for(iteration over all states x listed in controlsT) {
    if(conflict exists) {controlsT.erase(x); continue;}
    controls1[x].merge(control pattern in controlsT of x);}
```

Figure 3.9: Test for conflicts in control pattern and merge new controls to `controls1`

In the third step, the one step-operator is evaluate by `controls1`. Therefore, for each evaluated state, the disabled events according to `controls1` are stored in `disable` and used for the evaluation. Hence, the evaluation for each state x is simplified to evaluating if at least one transition ends in the `target` and the other transitions end either in `domain` $\cup$ `target` or can be disabled for all states in Obs(x) according to `controls1`. If a state passes the whole evaluation, it is inserted in the state set `theta`, see Figure 3.10.

```
for(iteration over all candidate states x) {
    pass = false;
    fail = false;
    x_ObsG = ObserverStatesMap[current state x];
    disable = events stored to be disabled in controls1[x_Obs];

    for(iteration over all transitions of current state) {
        if(transition event is listed in disable) continue;
        if(transition ends in target) {pass = true; continue;}
        if(transition ends in domain) {continue;}
        fail = true;
    }
    if(pass && !fail) theta.Insert(current state x);
}
```

Figure 3.10: Evaluation of the one-step operator condition by means of `controls1`

The rest of the algorithm is comparable to the steps described in Subsection 3.1.3.3. The feedback map `controls1X` can be used again to compute a particular solution to the control problem. For further details regarding the implementation, see (libFAUDES, 2015). Note that it has been given just a simplified sketch of the basic idea of the algorithm that is implemented in the libFAUDES and used successfully for the computation of solutions, e.g. for the example in Chapter 4.4.

3.2 Control problem based on input/output systems

The general idea of modelling discrete event systems by input/output systems is based on the Behavioural Systems Theory founded by J. C. Willems. In a series of publications, i.e. (Willems, 1991), Willems introduced a novel approach to system and control theory focusing on the set of trajectories of a dynamical system instead of modelling phenomena with a specific set of equations. Hence, instead of concentrating on internal details how a system behaves, e.g. in form of differential equations, the focus lies on the interaction of the system with its environment. As demonstrated in Figure 3.11, an input/output system is given by a kind of a black-box with inputs denoted by U, as well as outputs denoted by Y.

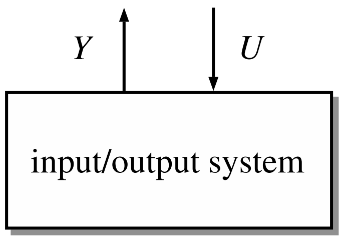


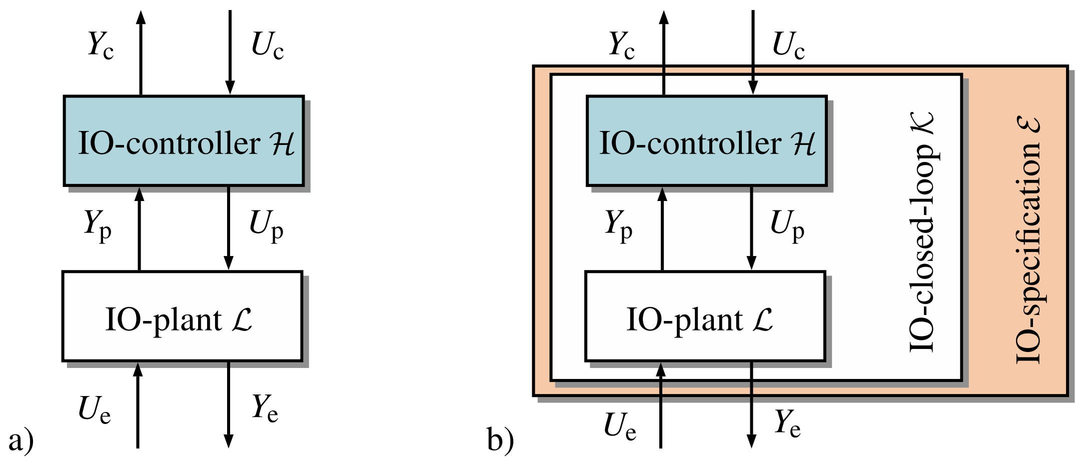
Figure 3.11: Input/output system

The input/output based framework inspired the development of solutions to supervisory control problems of hybrid systems applying an abstraction based approach; see (Moor and Raisch, 1999). A modular and a hierarchical extension of this framework is given in (Moor et al., 2001) and (Moor et al., 2003), respectively. In (Perk et al., 2006) and (Perk et al., 2008) the approach is further extended from hybrid systems to the application in hierarchical controller design for discrete event systems. Therein, all discrete events occurring in a dynamical system are divided into input and output events, also called *control and measurement events*, and a strict alternation of input and output events is required. The idea of hiding internal processes and abstracting the system behaviour to its external interaction possibilities is used in (Perk et al., 2006) to construct systematically a multi-level control architecture for topologically closed discrete event systems. As discussed in the introduction, topologically closed behaviours are in principal, not convenient for modelling liveness or eventuality properties, relevant for hierarchical controller design. Hence, in order to guarantee liveness, in (Perk et al., 2006) a particular notion of liveness was introduced, involving the output events of each subsystem. Nevertheless, a physical phenomenon may provide inherently particular eventuality properties, as well as a specification may require a desired configuration to be reached eventually. These natural liveness properties can be expressed easily by ω-languages, given that the limitation to topologically closed languages is dropped. According to (Moor et al., 2011), this is indeed possible in the abstraction based context of input/output systems. In particular, the given system is required to satisfy an additional specific controllability condition in order to guarantee liveness in the closed-loop configuration of a controller that was synthesised on the basis of abstractions. This result is used in this thesis to set up a hierarchical, abstraction based, and modular control approach for ω-languages. Note that basic ideas of this approach have been published in (Baier and Moor, 2012) and (Baier and Moor, 2015).

The section is structured as follows. In Section 3.2.1 properties of the IO-control-problem are introduced. In Subsection 3.2.2 properties of the IO-plant are presented. Subsection 3.2.3 introduces the structure of IO-specifications. A solution to the IO-control-problem is developed in Subsection 3.2.3 and the resulting closed-loop properties are analysed in Subsection 3.2.5. The partitioning of the events into input and output events might seem to complicate the modelling process. However, for the results on abstraction-based controller design in Chapter 4, the input/output structure is a crucial prerequisite, as also demonstrated in (Perk et al., 2006) or (Baier and Moor, 2015). Hence, throughout this section the modelling process is demonstrated in the context of a transportation system. Simple examples illustrate how to receive models for the IO-plant and the IO-specification with input/output event alternation from a given ad hoc model.

3.2.1 Properties of the IO-control-problem

The main idea behind the monolithic control problem for input/output systems is demonstrated in Figure 3.12 a) and b).



Y_c
U_c
IO-controller $\mathcal{H}$
Y_p
U_p
IO-plant $\mathcal{L}$
U_e
Y_e
a)
Y_c
U_c
IO-controller $\mathcal{H}$
Y_p
U_p
IO-plant $\mathcal{L}$
IO-closed-loop $\mathcal{K}$
IO-specification $\mathcal{E}$
U_e
Y_e
b)

Figure 3.12: a) Interaction of IO-plant and IO-controller b) IO-control-problem

In comparison to the closed-loop configuration introduced in Chapter 3, the discrete event system is referred to as *IO-plant* and the supervisor is denoted as *IO-controller*. Further, IO-plant and IO-controller interact with each other, with an *IO-environment* and with high-level coordinators via input events U and output events Y. The interaction of IO-plant and IO-controller is called the *IO-closed-loop*. The mentioned IO-environment models physical dependencies to other IO-plants, whereas high-level coordinator integrate the IO-closed-loop into a hierarchical control architecture. Further details regarding the closed-loop system in a hierarchical and modular control context are presented in Chapter 4. The *IO-specification* is also given by input and output events, but focusing on the interaction of the closed-loop system with the IO-environment and a high-level IO-controller.

The solution to the control problem under consideration is a controller that synchronizes alternating symbols from the *internal plant alphabet* $\Sigma_{\mathrm{p}} := U_{\mathrm{p}} \dot{\cup} Y_{\mathrm{p}}$ with the plant, forming a closed-loop configuration. The controller interacts with a high-level operator, while the plant provides the interaction with the low-level environment. It is taken the perspective that the operator seeks to affect the environment according to high-level commands from U_{c}. The controller is meant to implement each high-level command on the plant by applying suitable events from U_{p}, while monitoring the plant responses ranging in Y_{p}. Eventually, the controller shall provide a high-level feedback event from Y_{c} to the operator, in order to receive the subsequent high-level command. Thus, a specification referring to the *overall alphabet* is meant to relate *high-level control events* $\Sigma_{\mathrm{c}} := U_{\mathrm{c}} \dot{\cup} Y_{\mathrm{c}}$ with *low-level plant events* $\Sigma_{\mathrm{e}} := U_{\mathrm{e}} \dot{\cup} Y_{\mathrm{e}}$, and, thereby, formally define the consequences of high-level commands; see also Figure 3.12 b). Summing up, the control problem under consideration consists of an IO-plant, an IO-Specification and three ports for system interconnection. The purpose of the control problem is the design of an *IO-controller* $\mathcal{H} \subseteq \Sigma_{\mathrm{cp}}^{\omega}$ such that the closed-loop behaviour satisfies the IO-specification. The relevant parameters are summarized as a control problem.

Definition 3.2.1 IO-control-problem

An ***IO-control-problem*** consists of the following components

$\Sigma := \Sigma_{\mathrm{p}} \dot{\cup} \Sigma_{\mathrm{e}} \dot{\cup} \Sigma_{\mathrm{c}}$, the *overall alphabet*,

$\Sigma_{\mathrm{c}} := U_{\mathrm{c}} \dot{\cup} Y_{\mathrm{c}}$, the *high-level control events*,

$\Sigma_{\mathrm{p}} := U_{\mathrm{p}} \dot{\cup} Y_{\mathrm{p}}$, the *internal plant events*,

$\Sigma_{\mathrm{e}} := U_{\mathrm{e}} \dot{\cup} Y_{\mathrm{e}}$, the *low-level plant events*,

$\mathcal{L} \subseteq (\Sigma_{\mathrm{p}} \dot{\cup} \Sigma_{\mathrm{e}})^{\omega}$, the *IO-plant behaviour*, and

$\mathcal{E} \subseteq \Sigma^{\omega}$, the *IO-specification*.

Throughout this thesis, the IO-control-problem is referred by $(\Sigma, \mathcal{L}, \mathcal{E})$. Further, it is

$\Sigma_{\mathrm{pe}} := \Sigma_{\mathrm{p}} \dot\cup \Sigma_{\mathrm{e}}$, the *plant alphabet*,

$\Sigma_{\mathrm{cp}} := \Sigma_{\mathrm{c}} \dot\cup \Sigma_{\mathrm{p}}$, the *controller alphabet*,

$\Sigma_{\mathrm{ce}} := \Sigma_{\mathrm{c}} \dot\cup \Sigma_{\mathrm{e}}$, the *external alphabet*,

$\Sigma_{\mathrm{uc}} := U_{\mathrm{c}} \dot\cup Y_{\mathrm{p}} \dot\cup \Sigma_{\mathrm{e}}$, the *uncontrollable events*, and

$\Sigma_{\mathrm{o}} := \Sigma_{\mathrm{c}} \dot\cup \Sigma_{\mathrm{p}}$, the *observable events*. □

Projections from strings or infinite strings over Σ, to any of the above subsets of Σ, are denoted $\mathrm{p}_{\_}$ and $\mathrm{p}^{\omega}_{\_}$, respectively, with a subscript to indicate the respective range; e.g., p_{pe} for the projection from Σ^* to Σ^*_{pe}.

3.2.2 Properties of the IO-plant

The IO-plant given in Figure 3.13 is considered to be part of a large-scale system consisting of several plants and several controllers.

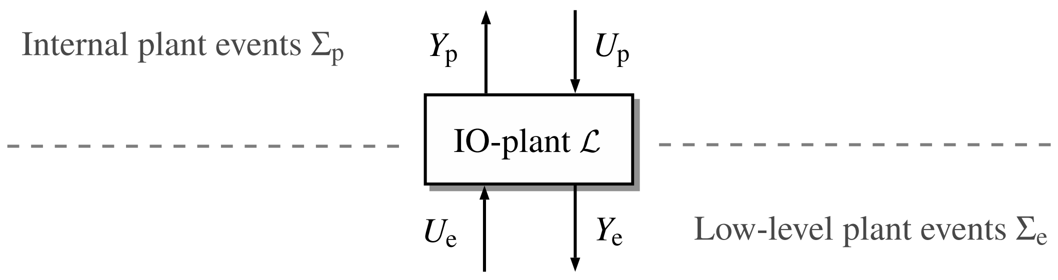

Internal plant events Σ_{p}
Y_{p}
U_{p}
IO-plant $\mathcal{L}$
U_{e}
Y_{e}
Low-level plant events Σ_{e}

Figure 3.13: IO-plant

Thus, it provides two different types of event interfaces for possible interaction. The first interface is given by the *internal plant alphabet*, denoted by $\Sigma_{\mathrm{p}} := U_{\mathrm{p}} \dot\cup Y_{\mathrm{p}}$, the second one by the *low-level plant alphabet* $\Sigma_{\mathrm{e}} := U_{\mathrm{e}} \dot\cup Y_{\mathrm{e}}$. By $\Sigma_{\mathrm{pe}} := \Sigma_{\mathrm{p}} \dot\cup \Sigma_{\mathrm{e}}$ the respective *plant alphabet* is denoted. The plant behaviour $\mathcal{L} \subseteq \Sigma^{\omega}_{\mathrm{pe}}$ is required to exhibit an strictly alternating input and output event structure.

Further, the plant has to accept any input event from the controller and from the environment, when expecting an input event from the respective part. To characterize this specific acceptance condition of input events, it is referred to the notion of a locally free input; see also (Perk et al., 2006).

Definition 3.2.2 Locally free input

For a language $L \subseteq \Sigma^*$, the alphabet $U \subseteq \Sigma$ is a *locally free input*, if

$$(\forall s \in \Sigma^*, \mu, \mu' \in U)\,[s\mu \in \operatorname{pre} L \Rightarrow s\mu' \in \operatorname{pre} L]. \qquad \square$$

It is not restrictive, but rather cosmetic, to require locally free inputs. Ad hoc models often lack these structural property since some inputs are considered to be useless in some system states. Nevertheless, considering e.g. programmable logic controller, note that at any moment it is possible to enable or disable actuator events. Hence, requesting locally free inputs just requires the model to contain more details of the natural plant behaviour. Locally free inputs can be realised by adding some *error states* to which these inputs lead. These *error states* can not be left again by any of the future traces, and, therefore signalise useless input events. Observe that an admissible controller will not activate one of these useless inputs. The plant behaviour $\mathcal{L}$ is required to possess the alternating input/output structure, as well as locally free inputs in order to refer to it as an *IO-plant*.

Definition 3.2.3 IO-plant

Given an alphabet $\Sigma_{\text{pe}} = \Sigma_{\text{p}} \dot\cup \Sigma_{\text{e}}$, the discrete event system $\mathcal{L} \subseteq \Sigma_{\text{pe}}^{\omega}$ is an *IO-plant*, if

P1 $\mathcal{L} \subseteq ((Y_{\text{p}}U_{\text{p}})^*(Y_{\text{e}}U_{\text{e}})^*)^{\omega} \subseteq \Sigma_{\text{pe}}^{\omega}$.

P2 $\operatorname{pre}\mathcal{L}$ possesses locally free inputs U_{p} and U_{e}. $\square$

In the following, the IO-plant is always requested to satisfy these conditions. A trivial example further illustrates structural properties of the IO-plant.

Example 3.2.1 IO-plant

The example system is a conveyor belt, as in Figure 1.3, that forms part of a laboratory model of a flexible manufacturing system. It consists of a simple conveyor belt to transport work pieces and a sensor in the middle of the belt to detect work pieces. Actuator events are provided to turn the belt motor on and off, denoted by `bm+` and `boff`. The sensor events `wpar` and `wplv` indicate the arrival of a work piece at the conveyor belt sensor or its leaving of the sensor, respectively. Actuator and sensor events are motivated by edges on the digital signals used to physically control the conveyor belt component by a programmable logic controller. For possible interaction with components placed next to the conveyor belt, the events `enter` and `exit` are defined, describing the entering of a work piece from the left or the exiting process to the right. Note that these events do not correspond to digital signals and are therefore not observable by the controller due to the lack of a respective sensor to detect these events. Further, the event `receipt` models possible feedback from the next component to inform about the reception of a work piece. In Figure 3.14 an ad hoc model is given for the physical plant behaviour, considering only one belt direction to keep it simple. Note that in the case that one actuator event is enabled in a state, all actuator events are enabled in that state, modelling the actual situation that a controller may enable each actuator event. In comparison to the physical conveyor belt model in (Baier and Moor, 2015), the plant behaviour is already reduced to model only feasible actuator input, i.e. actuator events before `enter` and `exit` are excluded, to keep the example simple. In contrast, the possibility of another work piece to enter right after the last work piece has exited the conveyor belt, is included in this model.

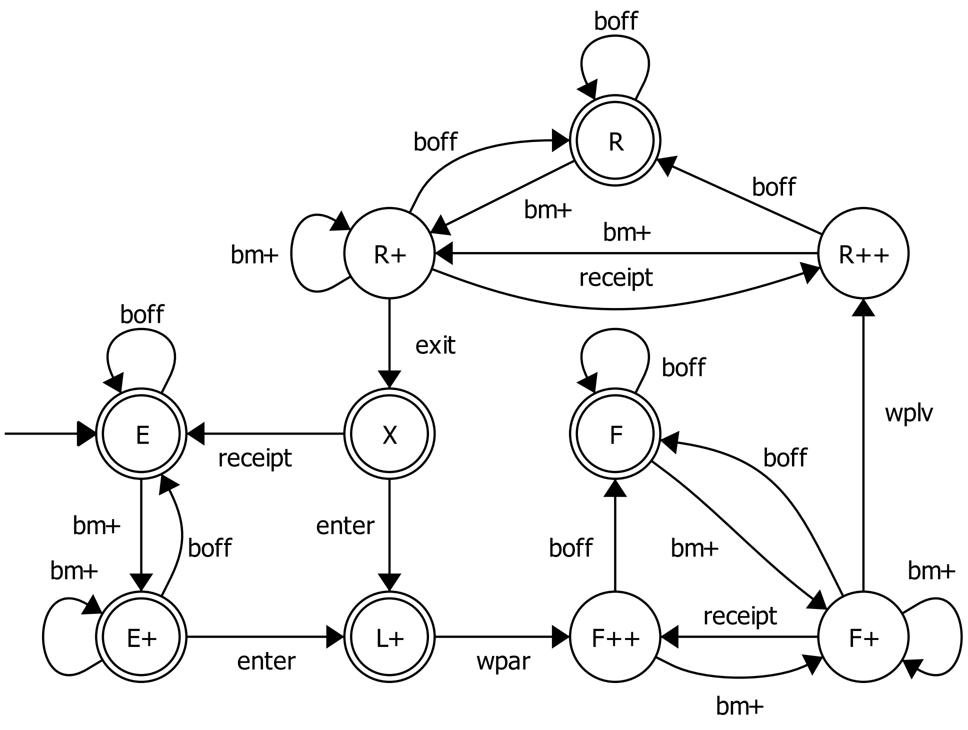


Figure 3.14: Physical ad hoc model of the conveyor belt

The physical model can be transformed to an IO-plant by replacing the occurrence of individual events by pairs of input and output events; see Figure 3.15. In Table 3.1, all events are summarized and categorized into respective alphabets. For this purpose, the additional output event `idle` is introduced to formally provide feedback when no sensor event occurred. The events `enter` and `exit` are replaced by pairs of request and acknowledgement. Here, `enter` is modelled by `get` to indicate the attempt to get a work piece from the left, which may succeed or fail, indicated by a subsequent positive acknowledge `pack` or negative acknowledge `nack`, respectively. Likewise, `exit` is modelled by `put`, followed by `pack` or `nack`. Provided that the belt motor is on, the

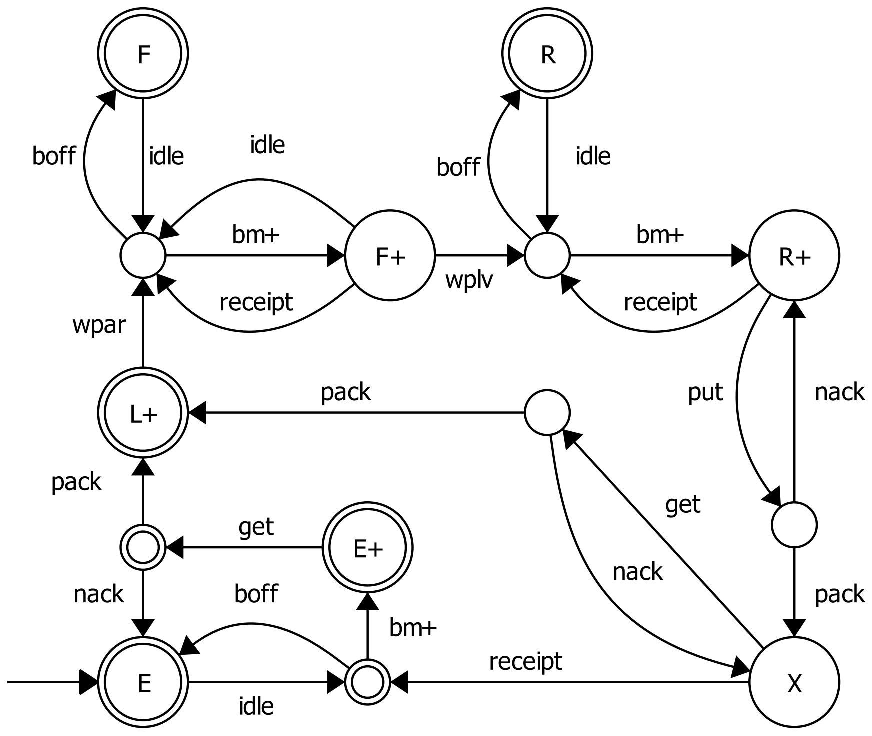


Figure 3.15: Conveyor belt, IO-plant model

Y_e:	`get, put`	attempt to get/put a workpiece from/to the environment
U_e:	`pack, nack`	acknowledgement of recent `get`/`put`
Y_p:	`idle,wpar,wplv receipt`	plant sensors with dummy `idle` if nothing else is to report
U_p:	`bm+,boff`	plant actuator to operate belt motor

Table 3.1: List of events of the conveyor belt IO-plant model

eventual occurrence of sensor events is modelled. It can be observed, that the conveyor belt model is not a topologically closed behaviour and, thus, contains eventuality properties it can be taken advantage of during a controller synthesis procedure. Regarding the IO-plant properties, observe that the event ordering, IO-plant property P1, can be proven easily by a language inclusion test. Locally free inputs, IO-plant property P2, is verified on a per state basis. □

3.2.3 Properties of the IO-specification

To characterise the desired plant behaviour, a language inclusion specification $\mathcal{E} \subseteq \Sigma^\omega$ is provided. The *overall alphabet* $\Sigma := \Sigma_\mathrm{p} \dot{\cup} \Sigma_\mathrm{e} \dot{\cup} \Sigma_\mathrm{c}$ consists of the internal plant alphabet, the low-level plant alphabet and the *high-level control alphabet* $\Sigma_\mathrm{c} := U_\mathrm{c} \dot{\cup} Y_\mathrm{c}$ that provides events for the interaction of the required controller with high-level components, i.e. superordinate controller in a hierarchical control framework. Considering pure monolithic controller design, it would be possible to just request a specification based on the alternation of input and output events. However, in order to design a suitable configuration for a hierarchical control architecture, the resulting closed-loop system has to satisfy particular properties. First of all, the resulting closed-loop system shall be used as IO-plant for the next higher controller design. Thus, the external closed-loop behaviour is required to possess again the plant properties P1 and P2. In particular, the external closed-loop must accept any external input events from U_c and U_e. Further, in order to guarantee liveness, it must persistently provide high-level feedback Y_c and shall relate external to internal signals. These considerations are already included in the structural properties of the IO-specification.

Definition 3.2.4 IO-specification

Given an alphabet $\Sigma = \Sigma_\mathrm{p} \dot{\cup} \Sigma_\mathrm{e} \dot{\cup} \Sigma_\mathrm{c}$, the discrete event system specification $\mathcal{E} \subseteq \Sigma^\omega$ is an *IO-specification*, if

E1 $\mathcal{E} \subseteq (((Y_\mathrm{p}U_\mathrm{p})^* (Y_\mathrm{e}U_\mathrm{e})^*)^* (Y_\mathrm{p}(Y_\mathrm{c}U_\mathrm{c})^+ U_\mathrm{p}))^\omega$.

E2 $\mathrm{pre}\,\mathcal{E}$ possesses locally free inputs U_c and U_e. □

Note that property E1 guarantees as well the alternating input/output structure, as the persistent feedback to the high-level, since events from Σ_c have to appear always eventually. Further note that in the intended usage, only events from Σ_e and Σ_c are necessary to specify the required behaviour and it is left to the controller design to add by synthesis respective control and measurement actions from the internal plant alphabet. The modelling process of an adequate specification is further illustrated by an example specification for the conveyor belt.

Example 3.2.2 IO-specification

In order to state a control objective, the operator output events `full` and `empty` are introduced to indicate whether or not a work piece is known to be available. Further, the operator input events `wpr` and `wpd` are used for the request and delivery of a work piece, respectively. The intended semantics of the newly introduced events is defined by relating them to the environment events via a specification automaton; see Figure 3.16. In Table 3.2, all events are summarized and categorized into respective alphabets.

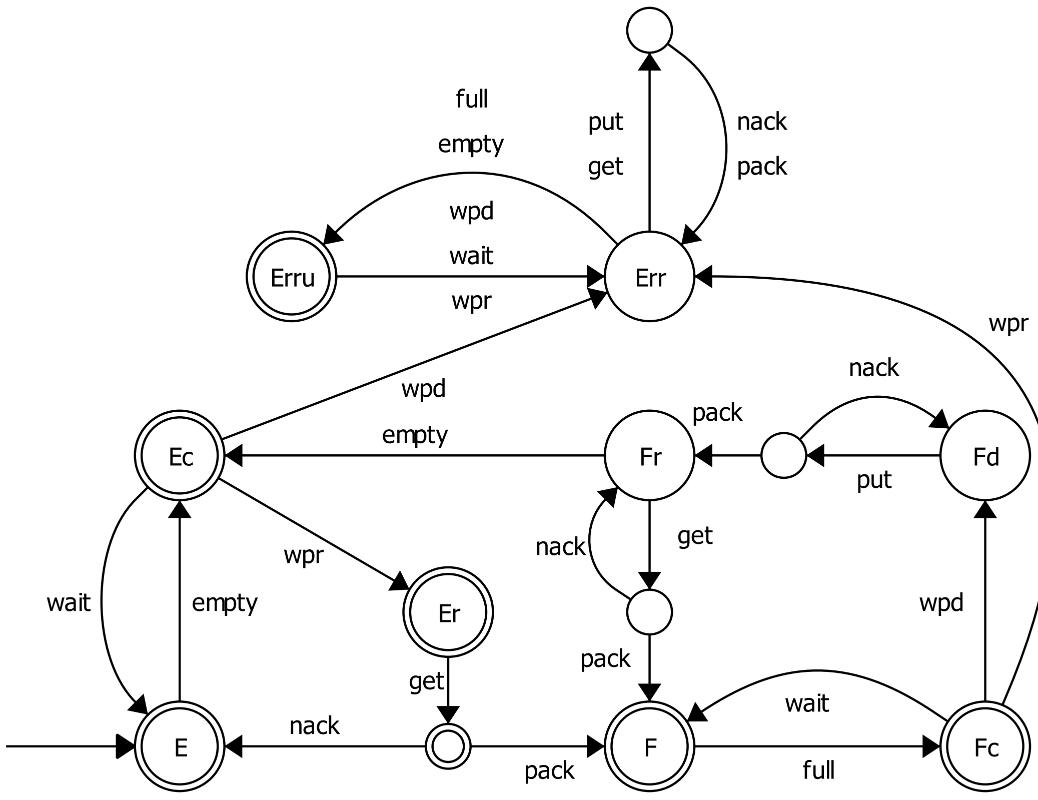


Figure 3.16: IO-specification for a conveyor belt

Y_e:	`get`, `put`	attempt to get/put a workpiece from/to the environment
U_e:	`pack`, `nack`	acknowledgement of recent `get`/`put`
Y_c:	`full`, `empty`	high-level measurement events
U_c:	`wpd`, `wpr`, `wait`	high-level actuator events

Table 3.2: List of events of the IO-specification of the conveyor belt

Note that the specification automaton exclusively refers to the alphabet Σ_{ce} and that it is left to the synthesis procedure to figure out how to drive the plant by interleaving events from Σ_p. Technically, the depicted automaton realises the projection $p^{\omega}_{ce}\mathcal{E}$ of the formal specification $\mathcal{E}$. In particular, there is no need to compute $p^{\omega}_{ce}\mathcal{E}$ from $\mathcal{E}$, avoiding a potentially exponential growth in the state count. The event ordering required by condition E1 is again verified by a language inclusion specification. The liveness requirement to eventually provide feedback to the operator is confirmed by inspecting all strictly connected components without an Y_c event and by verifying them not to include a marked state. Regarding locally free inputs E2, error states have been introduced to obtain a locally free U_{c}. A more specific error behaviour would have been possible. However, for the purpose of a subsequent design stage in a hierarchical control system, the proposed error behaviour will render the error states unreachable for any sensible high-level specification. Observe further, that the given specification is not topologically closed. □

3.2.4 Solution to the control problem

Given a control problem $(\Sigma, \mathcal{L}, \mathcal{E})$ with an IO-plant $\mathcal{L}$ and an IO-specification $\mathcal{E}$ as introduced before, an adequate controller $\mathcal{H} \subseteq \Sigma_{\mathrm{cp}}^{\omega}$ shall be synthesised. For the subsequent discussion, it turns out convenient to raise $\mathcal{L} \subseteq \Sigma_{\mathrm{pe}}^{\omega}$ to the overall alphabet Σ by applying the inverse projection to the overall alphabet. To avoid the creation of artificial liveness properties thereby and in order to further guarantee the intended event order, a particular construction is applied that already includes the event ordering of the specification.

Definition 3.2.5 Full IO-plant behaviour

Given an IO-plant $\mathcal{L} \subseteq \Sigma_{\mathrm{pe}}^{\omega}$. The *full IO-plant behaviour* is given by

$$\mathcal{L}_{\Sigma} := \left(\mathrm{p}_{\mathrm{pe}}^{-\omega}(\mathcal{L} \cup \mathrm{pre}\,\mathcal{L})\right) \cap \left(\mathrm{clo}\left(\left(Y_{\mathrm{p}}(Y_{\mathrm{c}}U_{\mathrm{c}})^{*}U_{\mathrm{p}}\right)^{*}\,(Y_{\mathrm{e}}U_{\mathrm{e}})^{*}\right)^{\omega}\right) \qquad \square$$

Since $\mathcal{L}$ is an IO-plant, $\mathcal{L}_{\Sigma}$ possesses locally free inputs U_{c}, $U_{\mathrm{p}} \dot{\cup} Y_{\mathrm{c}}$ and U_{e} by construction. Analysing the given control problem, observe that a control problem for sequential behaviours with not necessarily relatively closed specifications is given. Further, in comparison to the control problem in Section 3.1, for a given IO-control-problem, the controller is given in form of a language $\mathcal{H}$ instead of the description as a supervisor map f. Note that both representations can be converted into each other, compare e.g. (Cassandras and Lafortune, 2008). In equivalence to the full IO-plant behaviour, the controller behaviour w.r.t. the overall alphabet is denoted by $\mathcal{H}_{\Sigma} := \mathrm{p}_{\mathrm{cp}}^{-\omega}\mathcal{H} \subseteq \Sigma^{\omega}$. An admissible solution to the IO-control-problem $(\Sigma, \mathcal{L}, \mathcal{E})$ can now be characterised.

Definition 3.2.6 Solution to the IO-control-problem

Given an IO-control-problem $(\Sigma, \mathcal{L}, \mathcal{E})$, consider a candidate controller $\mathcal{H} \subseteq \Sigma_{\mathrm{cp}}^{\omega}$. $\mathcal{H}$ is a ***solution*** to the control problem, if

H1 $\mathcal{H}$ enforces the specification $\mathcal{E}$, i.e., $\mathcal{L}_{\Sigma} \cap \mathcal{H}_{\Sigma} \subseteq \mathcal{E}$.

H2 $\mathcal{H}_{\Sigma}$ is ω-admissible w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathcal{L}_{\Sigma})$. □

If $\mathcal{H}$ is a solution, H1 and E1 imply that the *IO-closed-loop behaviour* is

$$\mathcal{K} := \mathcal{L} \parallel \mathcal{H} := (\mathrm{p}_{\mathrm{pe}}^{-\omega}\mathcal{L}) \cap (\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathcal{H}) = \mathcal{L}_\Sigma \cap \mathcal{H}_\Sigma.$$

The last equality follows from the particular event ordering in $\mathcal{L}_\Sigma$ and $\mathcal{E}$. Observe that $\mathcal{L}_\Sigma \cap \mathcal{H}_\Sigma \subseteq \mathcal{E} \subseteq (((Y_\mathrm{p}U_\mathrm{p})^*\,(Y_\mathrm{e}U_\mathrm{e})^*)^*(Y_\mathrm{p}(Y_\mathrm{c}U_\mathrm{c})^+U_\mathrm{p}))^\omega$. Thus, it is $(\mathrm{p}_{\mathrm{pe}}^{-\omega}\,\mathrm{pre}\,\mathcal{L}) \cap \mathcal{L}_\Sigma \cap \mathcal{H}_\Sigma = \emptyset$. H2 implies that $\mathcal{L}_\Sigma$ and $\mathcal{H}_\Sigma$ are non-conflicting; see Lemma A.1 in the appendix. Hence, $(\mathrm{pre}\,\mathrm{p}_{\mathrm{pe}}^{-\omega}\mathcal{L}) \cap (\mathrm{pre}\,\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathcal{H}) = \mathrm{pre}(\,(\mathrm{p}_{\mathrm{pe}}^{-\omega}\mathcal{L}) \cap (\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathcal{H})\,)$. Thereby, plant $\mathcal{L}$ and controller $\mathcal{H}$ form a non-conflicting closed-loop system.

3.2.5 Properties of the closed-loop behaviour

The characterisation of solutions for the IO-control problem implies particular properties of the closed-loop system. One of these properties, the *ω-normality*, is introduced in the following Definition.

Definition 3.2.7 ω-normality

Given $\mathcal{L}$, $\mathcal{K} \subseteq \Sigma^\omega$, and a set of observable events $\Sigma_\mathrm{o} \subseteq \Sigma$, $\mathcal{K}$ is said to be *ω-normal w.r.t.* $(\Sigma_\mathrm{o}, \mathcal{L})$, if

$$\mathcal{K} = (\mathrm{p}_\mathrm{o}^{-\omega}\mathrm{p}_\mathrm{o}^{\omega}\mathcal{K}) \cap \mathcal{L}. \qquad \square$$

Note that ω-normality shows certain similarities with prefix-normality. As in the case of ∗-languages, ω-normality is retained under arbitrary union. Moreover, provided that $\mathcal{K}$ is relatively closed w.r.t. $\mathcal{L}$, prefix-normality of $\mathrm{pre}\,\mathcal{K}$ w.r.t. $(\Sigma_\mathrm{o},\ \mathrm{pre}\,\mathcal{L})$ implies normality of $\mathcal{K}$ w.r.t. $(\Sigma_\mathrm{o}, \mathcal{L})$.

Lemma 3.2.1 (Baier and Moor, 2012)

Let $\mathcal{K}$ be relatively closed w.r.t. $\mathcal{L} \subseteq \Sigma^\omega$ and $\Sigma_\mathrm{o} \subseteq \Sigma$, then $\mathcal{K}$ is ω-normal w.r.t. $(\Sigma_\mathrm{o}, \mathcal{L})$ if $\mathrm{pre}\,\mathcal{K}$ is normal w.r.t. $(\Sigma_\mathrm{o},\ \mathrm{pre}\,\mathcal{L})$.

Proof. $\mathcal{K} = (\mathrm{clo}\,\mathcal{K}) \cap \mathcal{L} = (\lim\mathrm{pre}\,\mathcal{K}) \cap \mathcal{L} = (\lim((\mathrm{p}_\mathrm{o}^{-1}\mathrm{p}_\mathrm{o}\,\mathrm{pre}\,\mathcal{K}) \cap (\mathrm{pre}\,\mathcal{L}))) \cap \mathcal{L} = (\lim\mathrm{pre}\,\mathrm{p}_\mathrm{o}^{-\omega}\mathrm{p}_\mathrm{o}^{\omega}\mathcal{K}) \cap (\lim\mathrm{pre}\,\mathcal{L}) \cap \mathcal{L} = (\mathrm{clo}\,\mathrm{p}_\mathrm{o}^{-\omega}\mathrm{p}_\mathrm{o}^{\omega}\mathcal{K}) \cap \mathcal{L} \supseteq (\mathrm{p}_\mathrm{o}^{-\omega}\mathrm{p}_\mathrm{o}^{\omega}\mathcal{K}) \cap \mathcal{L} \supseteq \mathcal{K}$. Equality implies that $\mathcal{K}$ is ω-normal. $\square$

It is now possible to analyse the properties of the closed-loop system that result from the interaction of the IO-plant and the IO-controller.

Proposition 3.2.2 (Baier and Moor, 2015)

If $\mathcal{H}$ is a solution to the control problem $(\Sigma, \mathcal{L}, \mathcal{E})$, where $\mathcal{L}$ is an IO-plant, then the IO-closed-loop behaviour $\mathcal{K} = \mathcal{L}_\Sigma \cap \mathcal{H}_\Sigma$ satisfies K1–K5:

K1 $\mathcal{K}$ enforces the specification $\mathcal{E}$, i.e., $\mathcal{K} \subseteq \mathcal{E}$,

K2 $\mathcal{K}$ is ω-admissible w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathcal{L}_\Sigma)$,

K3 $\mathcal{K}$ is ω-normal w.r.t. $(\Sigma_{\mathrm{o}}, \mathcal{L}_\Sigma)$,

K4 $\operatorname{pre}\mathcal{K}$ is normal w.r.t. $(\Sigma_{\mathrm{o}}, \operatorname{pre}\mathcal{L}_\Sigma)$,

K5 $\operatorname{pre}\mathcal{K}$ possesses locally free inputs U_{c} and U_{e}.

Proof. K1 and K2 are immediate consequences of H1 and H2. For K3 observe that

$$\begin{aligned}\mathcal{K} \subseteq &(\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathrm{p}_{\mathrm{cp}}^{\omega}\mathcal{K}) \cap \mathcal{L}_\Sigma = (\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathrm{p}_{\mathrm{cp}}^{\omega}(\mathcal{H}_\Sigma \cap \mathcal{L}_\Sigma)) \cap \mathcal{L}_\Sigma \subseteq \\ &(\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathrm{p}_{\mathrm{cp}}^{\omega}\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathcal{H}) \cap (\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathrm{p}_{\mathrm{cp}}^{\omega}\mathcal{L}_\Sigma) \cap \mathcal{L}_\Sigma = \mathcal{H}_\Sigma \cap \mathcal{L}_\Sigma = \mathcal{K}.\end{aligned}$$

K4 is obtained by

$$\begin{aligned}\operatorname{pre}\mathcal{K} &\subseteq (\mathrm{p}_{\mathrm{cp}}^{-1}\mathrm{p}_{\mathrm{cp}}\operatorname{pre}\mathcal{K}) \cap (\operatorname{pre}\mathcal{L}_\Sigma) = \\ &(\mathrm{p}_{\mathrm{cp}}^{-1}\mathrm{p}_{\mathrm{cp}}\operatorname{pre}(\mathcal{H}_\Sigma \cap \mathcal{L}_\Sigma)) \cap (\operatorname{pre}\mathcal{L}_\Sigma) \subseteq \\ (\mathrm{p}_{\mathrm{cp}}^{-1}\mathrm{p}_{\mathrm{cp}}\operatorname{pre}\mathcal{H}_\Sigma) &\cap (\mathrm{p}_{\mathrm{cp}}^{-1}\mathrm{p}_{\mathrm{cp}}\operatorname{pre}\mathcal{L}_\Sigma) \cap (\operatorname{pre}\mathcal{L}_\Sigma) = \\ (\operatorname{pre}\mathcal{H}_\Sigma) &\cap (\operatorname{pre}\mathcal{L}_\Sigma) = \operatorname{pre}(\mathcal{H}_\Sigma \cap \mathcal{L}_\Sigma) = \operatorname{pre}\mathcal{K}.\end{aligned}$$

For the penultimate equality, recall that H2 implies that $\mathcal{L}_\Sigma$ and $\mathcal{H}_\Sigma$ are non-conflicting. Regarding K5, pick $s,\ r \in \operatorname{pre}\mathcal{K}$, $\mu,\ \mu' \in U_{\mathrm{e}}$, and $\nu,\ \nu' \in U_{\mathrm{c}}$, such that $s\mu \in \operatorname{pre}\mathcal{K}$ and $r\nu \in \operatorname{pre}\mathcal{K}$. Observe that $s\mu,\ r\nu \in \operatorname{pre}\mathcal{K} \subseteq \operatorname{pre}\mathcal{L}_\Sigma$. According to P2 it follows that $s\mu' \in \operatorname{pre}\mathcal{L}_\Sigma$. Furthermore, the locally free input U_{c} of $\operatorname{pre}\mathcal{L}_\Sigma$ implies that $s\nu' \in \operatorname{pre}\mathcal{L}_\Sigma$. From ω-admissibility of $\mathcal{H}_\Sigma$ w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L}_\Sigma)$ and $s,\ r \in \operatorname{pre}\mathcal{H}_\Sigma$ follows that $s\mu',\ r\nu' \in \operatorname{pre}\mathcal{H}_\Sigma$. Recall again that $\mathcal{L}_\Sigma$ and $\mathcal{H}_\Sigma$ are non-conflicting, to obtain $s\mu',\ r\nu' \in (\operatorname{pre}\mathcal{L}_\Sigma) \cap (\operatorname{pre}\mathcal{H}_\Sigma) = \operatorname{pre}\mathcal{K}$. □

Vice versa, any ω-language that satisfies properties K1–K3 can be shown to be a solution to the control problem.

Proposition 3.2.3 (Baier and Moor, 2015)

Given a control problem $(\Sigma, \mathcal{L}, \mathcal{E})$, consider a closed-loop candidate $\mathcal{K} \subseteq \mathcal{L}_\Sigma$. If $\mathcal{K}$ satisfies K1–K3, then the controller $\mathcal{H} = \mathrm{p}_{\mathrm{cp}}^{\omega}\mathcal{K}$ solves the control problem $(\Sigma, \mathcal{L}, \mathcal{E})$.

Proof. Note that K2, by Lemma A.2 from the Appendix, implies that $(\operatorname{pre}\mathcal{L}_\Sigma) \cap (\operatorname{pre}\mathcal{K})$ is prefix-normal w.r.t. $(\Sigma_{\mathrm{o}}, \operatorname{pre}\mathcal{L}_\Sigma)$, and, together with $\mathcal{K} \subseteq \mathcal{L}_\Sigma$, K4 is obtained. According to K1 and K3, it is $\mathcal{L}_\Sigma \cap \mathcal{H}_\Sigma = \mathcal{L}_\Sigma \cap (\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathrm{p}_{\mathrm{cp}}^{\omega}\mathcal{K}) = \mathcal{K} \subseteq \mathcal{E}$, hence, $\mathcal{H}$ satisfies H1. To establish H2, pick an arbitrary $s \in (\operatorname{pre}\mathcal{L}_\Sigma) \cap (\operatorname{pre}\mathrm{p}_{\mathrm{cp}}^{-\omega}\mathrm{p}_{\mathrm{cp}}^{\omega}\mathcal{K})$. Here, K4 implies $s \in \operatorname{pre}\mathcal{K}$. According to K2, it is possible to choose $\mathcal{V}_s \subseteq \mathcal{L}_\Sigma \cap \mathcal{K} \subseteq \mathcal{L}_\Sigma \cap \mathcal{H}_\Sigma$ such that $s \in \operatorname{pre}\mathcal{V}_s$, $\operatorname{pre}\mathcal{V}_s$ is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \operatorname{pre}\mathcal{L}_\Sigma)$, prefix-normal w.r.t. $(\Sigma_{\mathrm{o}}, \operatorname{pre}\mathcal{L}_\Sigma)$, and relatively closed w.r.t. $\mathcal{L}_\Sigma$. Hence, $\mathcal{H}_\Sigma$ is ω-admissible and satisfies H2. □

Hence, in order to design an adequate controller, a closed-loop system has to be synthesised that satisfies K1–K3. The algorithms presented in Subsection 3.1.3 can be applied to find a candidate. Further, an appropriate IO-controller that solves the control problem can be deduced directly by projection of the closed-loop system to the alphabet Σ_{cp}. For the sake of completeness, the solution process is demonstrated by solving the control problem of the conveyor belt example.

Example 3.2.3 Solution to the conveyor belt IO-control-problem

The IO-plant model of the conveyor belt in Figure 3.15 together with the IO-specification in Figure 3.16 are considered. To solve the given IO-control-problem, the algorithm for partially observed ω-languages in Subsection 3.1.3 is applied and a closed-loop candidate $\mathcal{K}$ is synthesised that satisfies K1–K3 and is relatively closed w.r.t. the plant behaviour. This implies that the respective controller $\mathcal{H}$ is closed and can be deduced directly by the projection of $\mathrm{clo}\,\mathcal{K}$ to Σ_c. The result is a controller with 36 states.

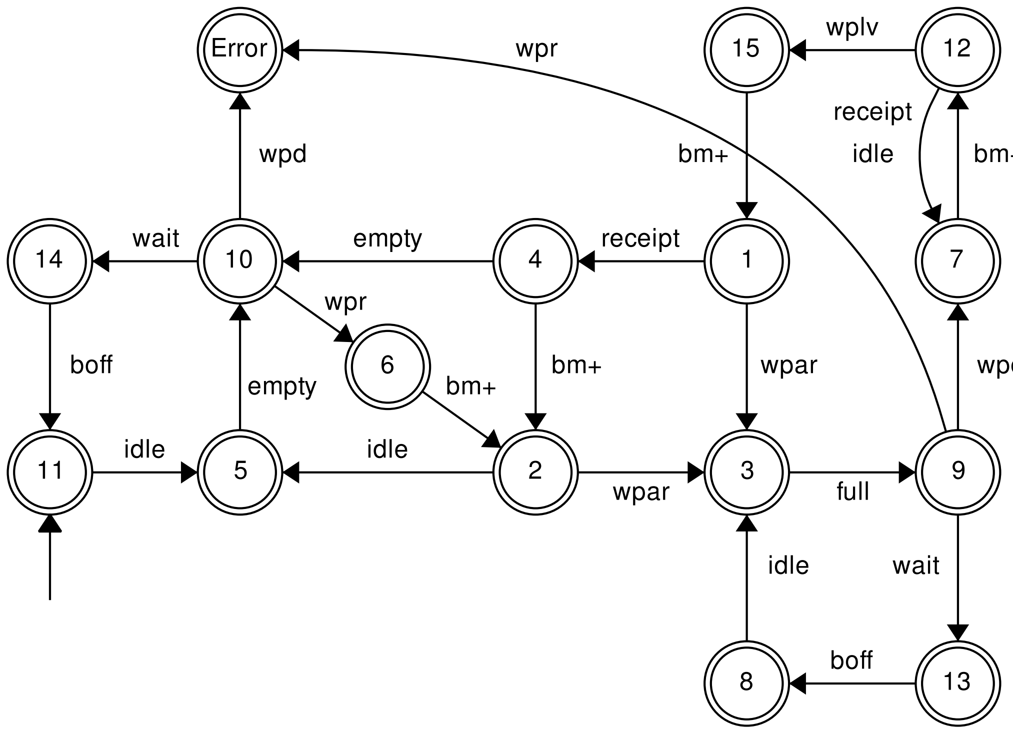


Figure 3.17: IO-controller of the conveyor-belt

In Figure 3.17 a sub-language of this controller is shown. Note that for the sake of clarity, high-level control inputs that lead to the error behaviour of the specification are just represented by a symbolical state, which is named `Error`. □

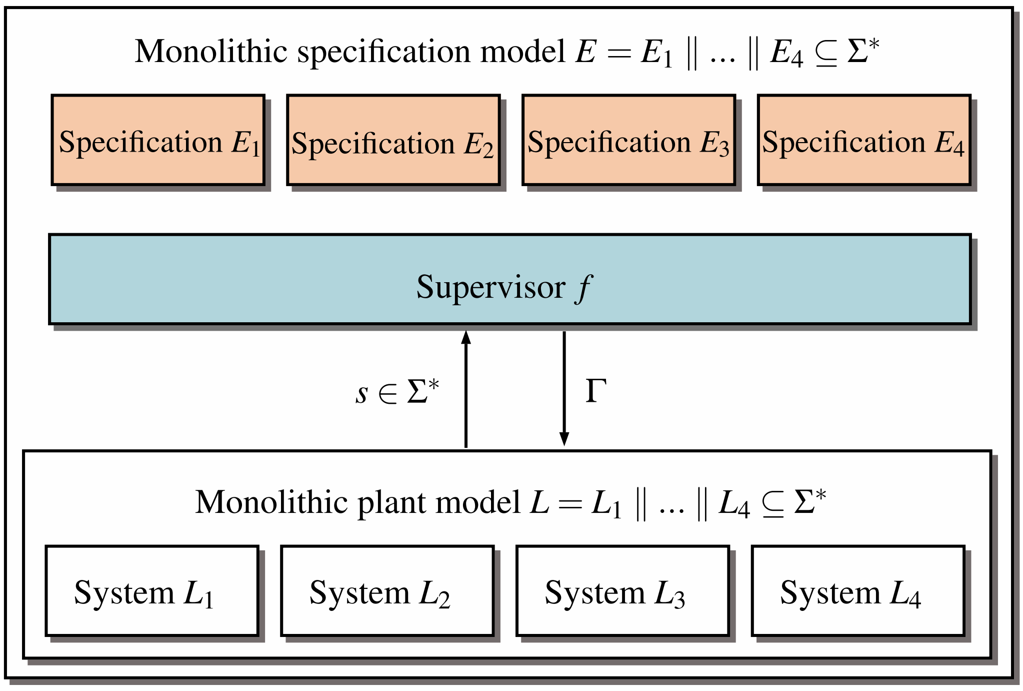
3.3 Conclusion - Monolithic controller design for ω-languages

In this Chapter, the general control problem for ω-languages has been introduced on the basis of the control problems for $*$-languages described in the Preliminaries. For the case of relatively closed specifications, the control problem for ω-languages can be transferred easily to an equivalent control problem for $*$-languages. Using known algorithms for $*$-languages, the computation of solutions is straight forward. In the case of more general specifications, a similar transfer to a $*$-language problem is not possible. In particular, an alternative characterisation for the set of possible closed-loop behaviours has to be defined, since the closed-loop property of relative closure w.r.t. the plant eventuality properties is not preserved under arbitrary unions. This alternative characterisation was first introduced in (Moor et al., 2011) and describes the set of finite strings that can be controlled to satisfy the given specification on the infinite-time horizon. This characterisation can be extended to solve the control problem under partial observation. To support the practical computation of solutions, several algorithmic implementations have been described. In particular a solution algorithm for the control problem under complete observation is described which was introduced in (Thistle and Wonham, 1992). It is given in form of a μ-Calculus, a Temporal Logic that is often used to describe complex properties of automata and which is based on a property characterisation using greatest and least fix-points. Further, it has been described how this computation can be extended to compute solutions for the control problem under partial observation.

In the second part of this chapter, the control problem for sequential behaviours has been adapted to an input/output system description, according to the Behavioural System Theory introduced in (Willems, 1991). In order to prepare an appropriate framework for the design of hierarchical and modular control architectures, a basic control problem for input/output systems is defined. Therefore, plant and specification of the given control problem are required to satisfy some fundamental structural properties. The solution of the control problem results into a closed-loop configuration with external signals satisfying again the particular properties, which have been required from the underlying plant. This fact opens the possibility to reuse the closed-loop system as a plant and to build up a hierarchical and modular control architecture. Thus, the presented procedure enables for the first time the design of hierarchical, abstraction-based, and modular control architectures for ω-languages, without the limitation to topologically closed languages.

4 Hierarchical and modular controller design for ω-languages

From a practical perspective it is not sufficient to offer a procedure that just solves the given discrete event control problem, without considering applicability of the procedure and the computational effort. In applications of the field of transport or manufacturing systems, the number of components running synchronously often grows significantly. Thus, a monolithic supervisor often has to interact simultaneously with a large number of system components; see Figure 4.1.



Monolithic specification model $E = E_1 \parallel \ldots \parallel E_4 \subseteq \Sigma^*$
Specification E_1
Specification E_2
Specification E_3
Specification E_4
Supervisor f
$s \in \Sigma^*$
Γ
Monolithic plant model $L = L_1 \parallel \ldots \parallel L_4 \subseteq \Sigma^*$
System L_1
System L_2
System L_3
System L_4

Figure 4.1: Monolithic design for large-scale systems

To synthesise a respective supervisor, the behaviour of the whole system, the monolithic system model, has to be determined. Hence, it is necessary to compute the synchronous composition (see Subsection 2.2.3) of all separate plant models. On the other hand, a large-scale system often requires a large number of specifications, which have to be composed to a single model as well. Using the composed system model and the composed specification model, a single controller for all components has to be synthesised; see Figure 4.1. The number of system states grows exponentially in the number of components and specifications. Therefore, the time required to solve the control problem can increase exponentially. In (Gohari and Wonham, 2000) a detailed complexity analysis is provided. As a consequence, the computational effort often results to be unreasonable for practical applications. This is the motivation for various approaches to offer a work around to the known problem of the combinatorial explosion of large scale systems.

A widely-used approach to handle the computational effort of supervisor design for large-scale systems, is the *modular controller design*. The basic idea behind is the division of the control task into several sub-tasks, and, the use of several sub-controllers; see for example (Wonham and Ramadge, 1988). Another very common approach, which can be combined with the first one, is the *abstraction-based design*. Instead of employing the actual monolithic plant model, a simplified model is systematically extracted and used for the synthesis procedure. The challenge in using this method, is to be able to guarantee the required closed-loop properties and to generate an admissible solution by using only the abstracted model.

In this Chapter, a procedure is presented to design an abstraction-based hierarchical and modular control architecture, for a control problem given by ω-languages. In Section 4.1, a short introduction into the idea behind modular and abstraction-based controller design is given and some existing approaches for $*$-languages are analysed. In Section 4.2, a procedure is introduced to design abstraction-based IO-controller for the input/output-based framework described in the previous Chapter. In Section 4.3, a modular control design procedure for input/output-systems is given. Finally, in Section 4.4, the efficiency of hierarchical and modular controller design for input/output-systems is demonstrated by means of a transport systems example.

4.1 Hierarchical and modular controller design for $*$-languages

Given a large scale system and given the fact that the number of states growth exponentially by the number of components involved, the monolithic controller design reaches easily its limits. As a consequence, several approaches have been developed for the control problem involving $*$-languages in order to avoid the computation of the monolithic plant and/or specification model. In the following, the focus lies on those approaches developed for $*$-languages, that show certain similarities to the hierarchical and modular control architecture that is developed in this thesis for ω-languages.

4.1.1 Modular controller design

The first idea to avoid the computation of large monolithic models was the application of modular control. In (Ramadge and Wonham, 1987b) and (Wonham and Ramadge, 1988) a procedure has been presented that, instead of synthesising a monolithic controller, uses several decentralized sub-controllers, splitting the control task into several sub-tasks in accordance with the separate specifications; see Figure 4.2.

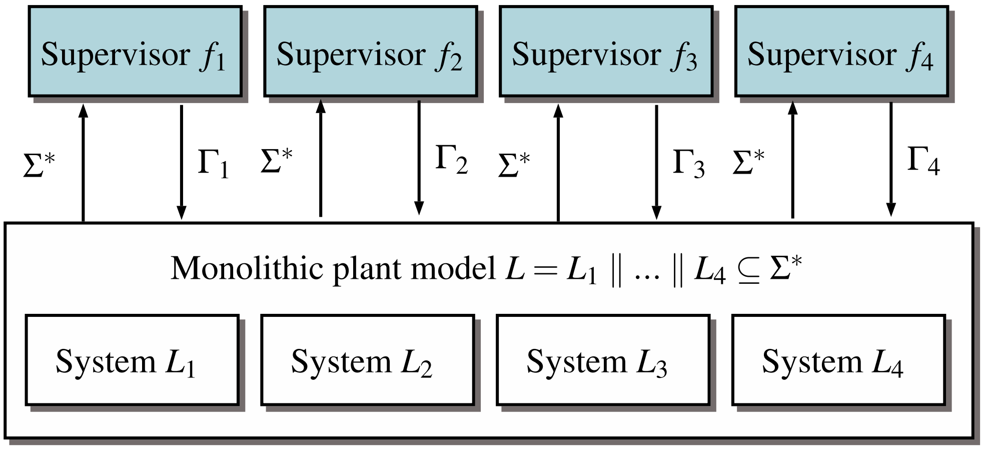


Figure 4.2: Modular controller design

This procedure often reduces computational effort, in particular in the case of prefix-closed languages. Otherwise a test whether the involved supervisors are nonconflicting has to be performed that requires the construction of the composition of the given specifications.

Some years later, an alternative procedure has been presented in (de Queiroz and Cury, 2000a) and (de Queiroz and Cury, 2000b) that exploits the fact that large scale systems in general consist of several sub-systems and their specifications often refer only to some of these subsystems. As a consequence, the control problem is divided into local control problems involving only parts of the whole system. By that, the construction of the monolithic system model can be avoided and the computational effort can be reduced; see also Figure 4.3.

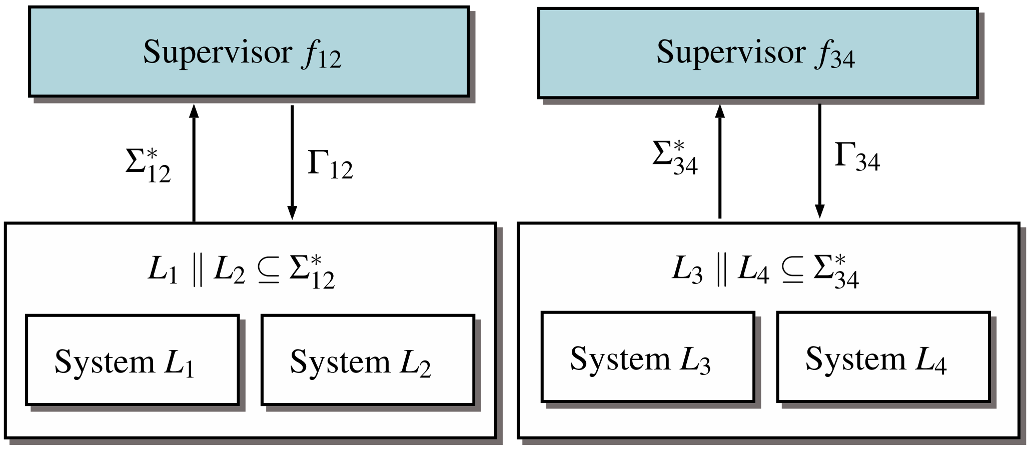


Figure 4.3: Modular controller design

Further, the property of non-conflicting has to be tested only locally on the basis of the involved subsystems. In many cases, this procedure reduces the computational effort, since the number of states of the sub-components is smaller.

4.1.2 Hierarchical abstraction-based controller design

Another procedure to handle the problem of the growing computational effort for large-scale systems is the use of a hierarchical control architecture combined with abstraction-based controller design. In contrast to the horizontal organisation of the modular control design presented before, the hierarchical design focuses on exploiting vertical modularities in the given system. The hierarchical structure can either be build in a *top-down* or in a *bottom-up design*. The top-down design proposes to start with the control on the highest level and to incorporate or deduce step-by-step control details for the subordinate levels; see e.g. (Ma and Wonham, 2006). More common in the area of supervisory control of $*$-languages is the bottom-up design; see e.g. (Schmidt et al., 2008; Perk et al., 2008; Feng and Wonham, 2008; da Cunha et al., 2002; Leduc et al., 2005). It proposes to start with the detailed physical plant model and to build step-by-step a hierarchy of superordinate controllers. Thereby, the degree of details shall be reduced from one level to the next, in order to reduce the computational effort. Like demonstrated in Figure 4.4, the basic idea is to use abstractions L^{hi} of the original plant L for the design of a high-level controller f^{hi} that can be realised by a low-level controller f and to keep the grade of details as small as necessary for the design of the superordinate controller.

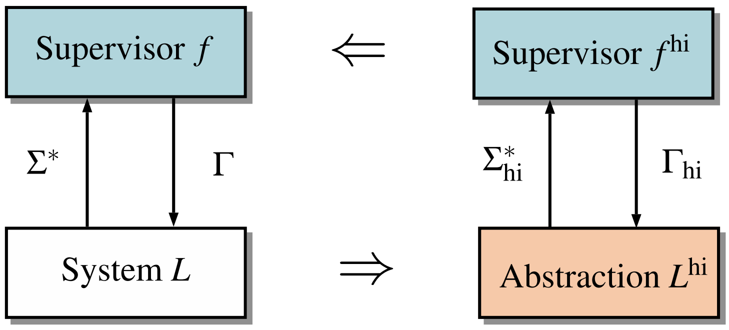


Figure 4.4: Abstraction-based controller design

A first analysis of requirements regarding the relation between high-level and low-level control problem, can be found in (Zhong and Wonham, 1990). Therein, the concept of *hierarchical consistency* is introduced within a two-level framework. a controller is designed based on the abstract high-level model. Hence, the concept of *hierarchical consistency* ensures the existence of a controller that is able to realize the control task synthesised on the high-level in a closed-loop system on the low-level. In (Wong and Wonham, 1996) this concept was extended in order to cope not only with the challenge of control consistency, but also with the problem of non-blocking in the hierarchical framework. Depending on the design of the high-level model, a controller designed for the high-level control task may be blocking with respect to the low-level plant. Hence, the notion of an *observer* is introduced as a sufficient condition for non-blocking hierarchical controller design. In the sequel, several variations of the observer property have been analysed on the basis of natural projections, like e.g. the *natural observer* (Schmidt et al., 2008; Feng and Wonham, 2006, 2008), or the less restrictive *consistency for the purpose of controller design*; see (Moor et al., 2013; Moor, 2014). In general, the natural projection does not guarantee a reduction of the state space. In particular, like discussed in (Wong, 1998), growth of the state-space is possible. In that case, the abstraction results into a control problem of even higher computational effort. The only known abstraction that guarantees that the state space is not growing is the abstraction using the natural observer; see (Wong, 1998). Another interesting topic in the abstraction-based design is the question, whether the high-level supervisor guarantees maximally permissiveness on the low-level. By the conditions *output control consistency* (Zhong and Wonham, 1990), or the less restrictive *local* *control consistency* (Schmidt and Breindl, 2011), maximally permissive control can be guaranteed. Algorithms to compute appropriate abstractions that guarantee non-blocking and/or maximal permissiveness are given e.g. in (Feng and Wonham, 2010) and (Schmidt and Breindl, 2011).

4.1.3 Hierarchical abstraction-based and modular controller design

The combination of modular controller design with abstraction-based controller design is an intuitive consequence. Figure 4.5 demonstrates just one of several possibilities to combine both approaches.

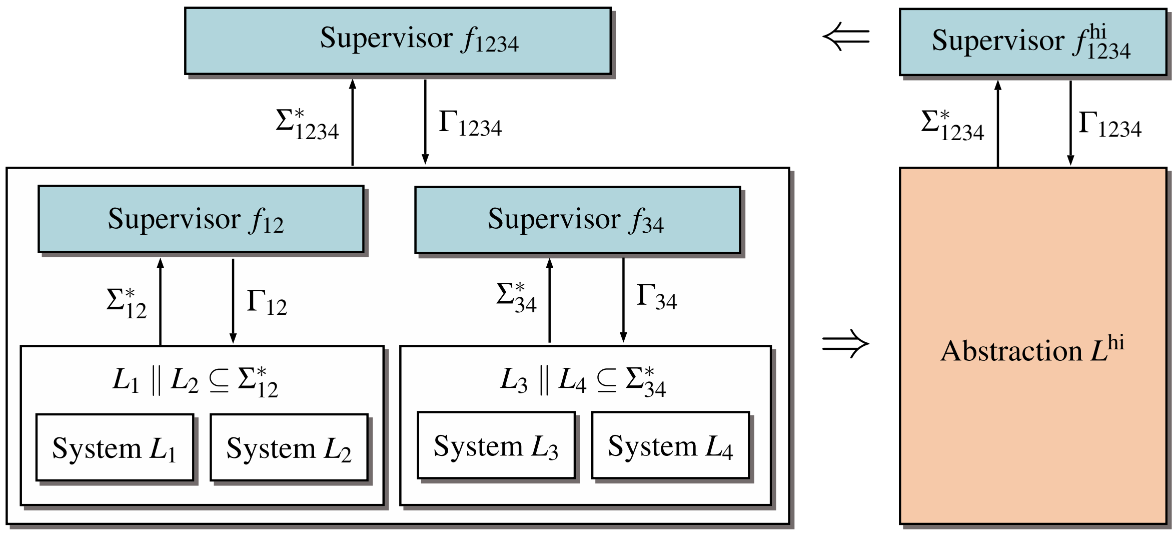


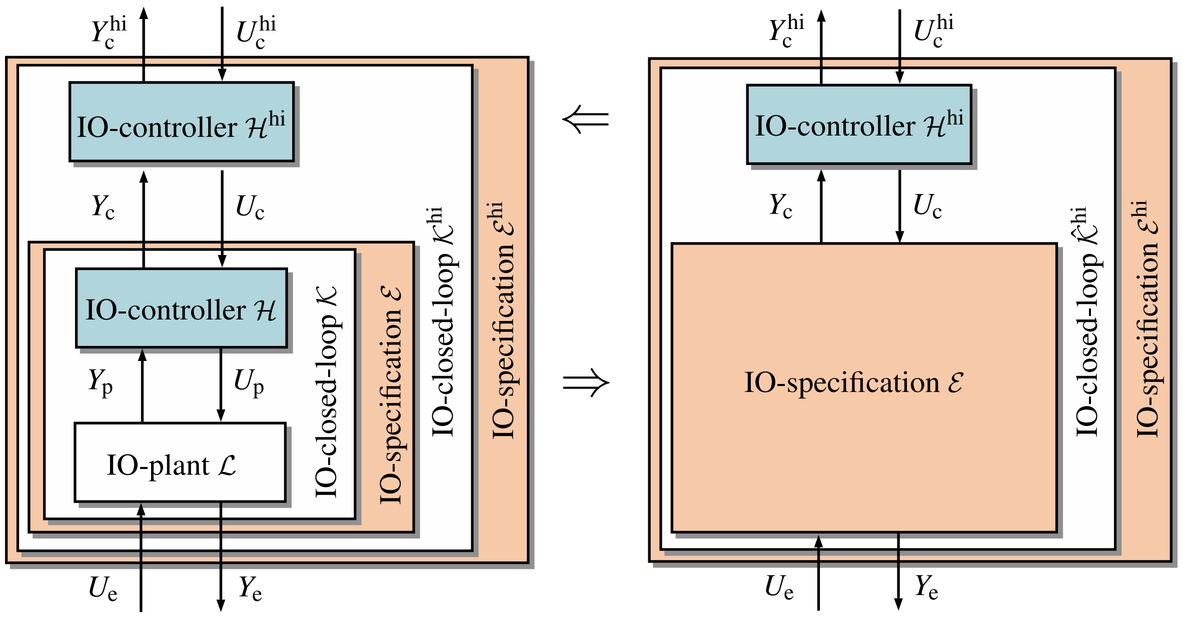
Figure 4.5: Abstraction-based and modular controller design

First, the modular supervisor f_{12} and f_{34} are designed on the low-level. Subsequently an abstraction L^{hi} of the synchronous product of the closed-loop behaviour of the modular closed-loops is computed. Depending on the abstraction, it is also possible to first compute the abstraction of each modular closed-loop and then build the synchronous product of the abstracted closed-loops, reducing thereby the computational effort. In particular, the natural observer is an effective and consistent abstraction that might be applied here; see (Schmidt et al., 2008; Feng and Wonham, 2008). The resulting abstracted plant model is used for the design of a high-level supervisor f^{hi}_{1234} that can be transferred easily to a supervisor f_{1234} that coordinates the modular supervisor on the low-level.

It would also be possible to derive low-level realizations for each sub-system; see (Schmidt et al., 2008). Moreover, abstractions can be used just to reduce the complexity of the test for non-blocking of modular supervisor; see (Pena et al., 2006). Computational efficiency of the combination of abstraction-based and modular controller design has been shown in a laboratory set-up of a three-story elevator; see (Kaess, 2014).

4.2 Abstraction-based control problem for input/output systems

Coming back to the control problem for sequential behaviours, the basic idea of abstraction-based hierarchical control shall be transferred to the presented framework of input/output systems; see also (Baier and Moor, 2015). As illustrated in Figure 4.6 to the left, it is given an IO-control problem $(\Sigma, \mathcal{L}, \mathcal{E})$ together with its solution $\mathcal{H}$ and its IO-closed-loop behaviour $\mathcal{K} = \mathcal{L}_\Sigma \cap \mathcal{H}_\Sigma$.



Y_c^{hi}
U_c^{hi}
IO-controller $\mathcal{H}^{hi}$
Y_c
U_c
IO-controller $\mathcal{H}$
Y_p
U_p
IO-plant $\mathcal{L}$
IO-closed-loop $\mathcal{K}$
IO-specification $\mathcal{E}$
IO-closed-loop $\mathcal{K}^{hi}$
IO-specification $\mathcal{E}^{hi}$
U_e
Y_e
$\Leftarrow$
$\Rightarrow$
Y_c^{hi}
U_c^{hi}
IO-controller $\mathcal{H}^{hi}$
Y_c
U_c
IO-specification $\mathcal{E}$
IO-closed-loop $\hat{\mathcal{K}}^{hi}$
IO-specification $\mathcal{E}^{hi}$
U_e
Y_e

Figure 4.6: Abstraction-based hierarchical controller design

For a superordinate control level, the *external closed-loop behaviour* $\mathcal{L}^{hi} := p_{ce}^{\omega}\mathcal{K}$ shall be used as IO-plant on the next higher control level. Thus, given a high-level specification $\mathcal{E}^{hi}$, a high-level control problem $(\Sigma^{hi}, \mathcal{L}^{hi}, \mathcal{E}^{hi})$ is received for the design of the superordinate controller $\mathcal{H}^{hi}$ and, thereby, a hierarchical control architecture can be obtained. However, in order to reduce computational effort, the ideas of the abstraction-based controller design for $*$-languages shall be transferred and an abstraction of the low-level closed-loop behaviour shall be used. By property H1 of the IO-controller $\mathcal{H}$, i.e. $\mathcal{K} \subseteq \mathcal{E}$, follows that the external closed-loop behaviour is a subset of the external specification behaviour $\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{E}$, i.e. $\mathcal{L}^{\mathrm{hi}} = \mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{K} \subseteq \mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{E}$. Hence, in order to solve $(\Sigma^{\mathrm{hi}}, \mathcal{L}^{\mathrm{hi}}, \mathcal{E}^{\mathrm{hi}})$, it is proposed to use the specification $\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{E}$ as an abstraction of the external closed-loop behaviour $\mathcal{L}^{\mathrm{hi}}$; see Figure 4.6 to the right.

There are several reasons why a computational improvement is expected from this procedure. First, the use of the projection to the alphabet Σ_{ce} hides detailed information about the direct interaction between low-level plant and low-level controller, information that is not necessary for the design of the high-level controller due to structural properties of the IO-control problem. Further, in contrast to the actual closed-loop $\mathcal{K}$, specification $\mathcal{E}$ does not express how the control objective is achieved and, hence, is often less complex. In addition, it has been shown that the specification $\mathcal{E}$ can often be formulated just on the alphabet Σ_{ce}; see e.g. (Baier and Moor, 2015). Note that, in addition, the abstraction $\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{E}$ is already given by the low-level control problem. Therefore, further computational effort in order to receive appropriate abstractions for the high-level is avoided.

The main challenge in the area of abstraction-based design for $*$-languages is to guarantee that hierarchical control consistency, as well as non-blocking is given in the closed-loop system. In the proposed approach for input/output systems, similar questions have to be considered. The first question results from the fact that the external IO-closed-loop behaviour shall be reused as IO-plant for the high-level control problem. Hence, it has to be analysed, whether the IO-plant properties P1 and P2 are retained under closed-loop composition and, thus, also satisfied by $\mathcal{L}^{\mathrm{hi}}$. The second question results from the intended usage of the specification as abstraction. As in the case of abstraction-based controller design for $*$-languages, it has to be shown that a solution of $(\Sigma^{\mathrm{hi}}, \mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{E}, \mathcal{E}^{\mathrm{hi}})$ also solves the actual problem $(\Sigma^{\mathrm{hi}}, \mathcal{L}^{\mathrm{hi}}, \mathcal{E}^{\mathrm{hi}})$. In particular, it has to be analysed, whether the actual closed-loop system satisfies the specification, property H1, and whether the controller is ω-admissible w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathcal{L}^{\mathrm{hi}})$, property H2.

As it turns out, an affirmative answer for the first question can be derived easily from the fact that the IO-closed-loop satisfies K1 and K5; see Proposition 3.2.2. However, the structural requirements P1 and P2 are not sufficient yet to provide an affirmative answer for the second question. Although solutions $\mathcal{H}^{\mathrm{hi}}$ of $(\Sigma^{\mathrm{hi}}, \mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{E}, \mathcal{E}^{\mathrm{hi}})$ are readily observed to also satisfy property H1 for the actual control problem $(\Sigma^{\mathrm{hi}}, \mathcal{L}^{\mathrm{hi}}, \mathcal{E}^{\mathrm{hi}})$, it is necessary to impose an additional condition on the IO-plant $\mathcal{L}$ in order to guarantee property H2.

4.2.1 Properties of the non-anticipating IO-plant

The additional condition imposed on the IO-plant has been studied first in (Moor et al., 2011) in the context of abstraction-based controller design for not necessarily topologically closed ω-languages. The definition relies on a system description with alternating inputs U and outputs Y, comparable to the input/output structure given in the IO-control-problem. The condition introduced therein is based on the definition of ω-controllability under complete event observation that has been introduced in Definition 3.1.4. Observe that locally-free inputs, as imposed by P2, guarantee completeness of an IO-closed-loop, even in an abstraction-based controller design. However, in (Moor et al., 2011) it has been demonstrated that they do not imply a non-blocking closed-loop for an abstraction-based controller design. In Figure 4.7, this fact is demonstrated by a small example. A simple IO-plant is given, with the alphabets $U_\mathrm{p} := \{\mathtt{a},\mathtt{b}\}$ and $Y_\mathrm{p} := \{\mathtt{A},\mathtt{B}\}$. Property P1 is easily verified. Regarding locally-free inputs, observe that the events a and b are both active in the states S1 and S3. Note further that the given IO-plant is not topologically closed. Figure 4.7 b) shows a possible abstraction $\mathcal{L}^{\mathrm{hi}} \supseteq \mathcal{L}$. In contrast to the original plant, the abstraction is topologically closed. Figure 4.7 c) shows an admissible solution to an abstraction-based control problem $(\Sigma^{\mathrm{hi}}, \mathcal{L}^{\mathrm{hi}}, \mathcal{E}^{\mathrm{hi}})$. It is obvious that the controller $\mathcal{H}^{\mathrm{hi}}$, although it is a solution for the abstraction-based control problem, is conflicting w.r.t. $\mathcal{L}$. In particular, the infinite-time closed-loop behaviour is empty. While P2 requires the plant to accept any input locally, an additional structural plant property is necessary to require that the plant is always in the position to choose its outputs such that it satisfies its own liveness properties. Hence, the plant may never anticipate the input in its infinite future. In (Moor et al., 2011) a variation of Willems' notion of non-anticipating input/output system is developed as a sufficient structural plant property for a non-conflicting

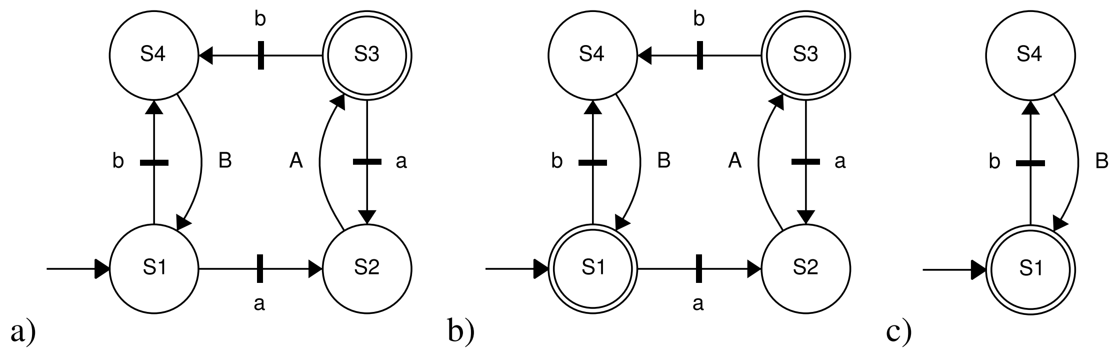


Figure 4.7: a) IO-plant $\mathcal{L}$ b) Abstraction $\mathcal{L}^{\mathrm{hi}}$ c) Abstraction-based controller $\mathcal{H}^{\mathrm{hi}}$

closed-loop. Based on those considerations, the additional requirement P3 is imposed on $\mathcal{L}$ to denote it as a *non-anticipating IO-plant*.

Definition 4.2.1 Non-anticipating IO-plant

Given an alphabet $\Sigma_{\mathrm{pe}} = \Sigma_{\mathrm{p}} \dot\cup \Sigma_{\mathrm{e}}$, the discrete event system $\mathcal{L} \subseteq \Sigma_{\mathrm{pe}}^{\omega}$ is a *non-anticipating IO-plant*, if

P1 $\mathcal{L} \subseteq ((Y_{\mathrm{p}}U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^{\omega} \subseteq \Sigma_{\mathrm{pe}}^{\omega}$,

P2 $\mathrm{pre}\,\mathcal{L}$ possesses locally free inputs U_{p} and U_{e}, and

P3 $\mathcal{L}$ is ω-controllable w.r.t. $(U_{\mathrm{p}} \dot\cup U_{\mathrm{e}}, \mathrm{clo}\,\mathcal{L})$. □

Technically, property P3 is a controllability condition, where the inputs events (the events, the plant is not able to restrict) are considered the uncontrollable events. The algorithm given in (Thistle and Wonham, 1992) can be applied to test for P3.

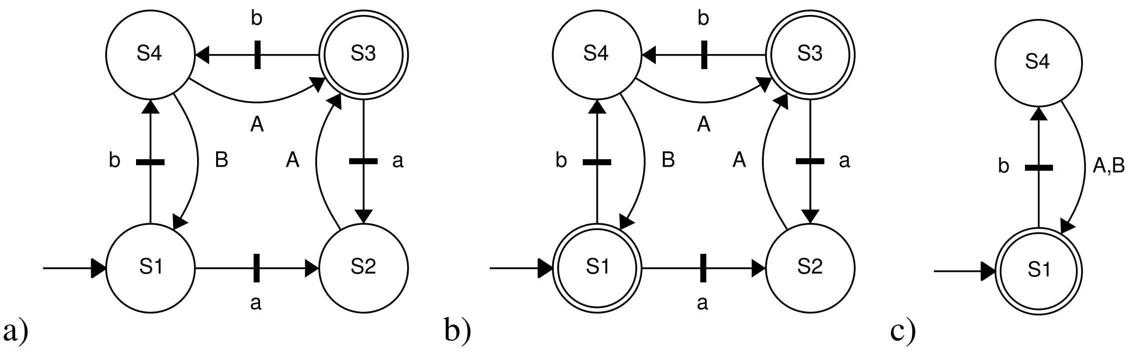


Figure 4.8: a) non-anticipating IO-plant $\mathcal{L}$ b) Abstraction $\mathcal{L}^{\mathrm{hi}}$ c) Controller $\mathcal{H}^{\mathrm{hi}}$

To demonstrate property P3, consider again, with a slight variation, the simple example in Figure 4.8 a). The only difference here to the example in Figure 4.7 is transition `A` from state `S4` to state `S3`. However, by this additional transition, the plant is always in the position to reach eventually its marked state and satisfy its liveness property, independently from the control inputs. Hence, the controller $\mathcal{H}^{\mathrm{hi}}$ in Figure 4.8 c) is non-conflicting w.r.t. $\mathcal{L}$, although its design is based on the abstraction $\mathcal{L}^{\mathrm{hi}}$.

Note that the non-anticipating property propagates from $\mathcal{L}$ to the full IO-plant behaviour $\mathcal{L}_{\Sigma}$, introduced in Definition 3.2.5.

Lemma 4.2.1

If $\mathcal{L}$ is a non-anticipating IO-plant, then $\mathcal{L}_{\Sigma}$ is ω-controllable w.r.t. $(\Sigma_{\mathrm{c}} \dot\cup U_{\mathrm{p}} \dot\cup U_{\mathrm{e}}, \mathrm{clo}\,\mathcal{L}_{\Sigma})$.

Proof. Note that $\mathcal{L}_\Sigma = (\mathrm{p}_{\mathrm{pe}}^{-\omega}(\mathcal{L} \cup \mathrm{pre}\,\mathcal{L})) \cap (\mathrm{clo}(\,(Y_{\mathrm{p}}(Y_{\mathrm{c}}U_{\mathrm{c}})^* U_{\mathrm{p}})^*\,(Y_{\mathrm{e}}U_{\mathrm{e}})^*\,)^\omega\,)$, implies $\mathrm{pre}\,\mathcal{L}_\Sigma \subseteq \mathrm{pre}(\mathrm{p}_{\mathrm{pe}}^{-\omega}(\mathcal{L} \cup \mathrm{pre}\,\mathcal{L})) = \mathrm{p}_{\mathrm{pe}}^{-1}\,\mathrm{pre}\,\mathcal{L}$. Pick an arbitrary string $s \in \mathrm{pre}\,\mathcal{L}_\Sigma$, let $r := \mathrm{p}_{\mathrm{pe}}s$, and observe that $r \in \mathrm{pre}\,\mathcal{L}$. Since $\mathcal{L}$ is non-anticipating, choose $\mathcal{W}_r \subseteq \mathcal{L}$, such that $r \in \mathrm{pre}\,\mathcal{W}_r$, and $\mathrm{pre}\,\mathcal{W}_r$ is controllable w.r.t. $(U_{\mathrm{p}} \dot\cup U_{\mathrm{e}},\, \mathrm{pre}\,\mathcal{L})$, and $\mathcal{W}_r$ is relatively closed w.r.t. $\mathrm{clo}\,\mathcal{L}$. Recall that relative closedness w.r.t. a closed language implies closedness. In particular, $\mathcal{W}_r$ is closed. To establish the non-anticipating property of $\mathcal{L}_\Sigma$, consider the candidate $\mathcal{V}_s := (\mathrm{p}_{\mathrm{pe}}^{-\omega}(\mathcal{W}_r \cup \mathrm{pre}\,\mathcal{W}_r)) \cap (\mathrm{clo}((Y_{\mathrm{p}}(Y_{\mathrm{c}}U_{\mathrm{c}})^*U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega)$. Clearly, $\mathcal{V}_s \subseteq \mathcal{L}_\Sigma$ and $\mathrm{pre}\,\mathcal{V}_s \subseteq \mathrm{pre}\,\mathrm{p}_{\mathrm{pe}}^{-\omega}(\mathcal{W}_r \cup \mathrm{pre}\,\mathcal{W}_r) = \mathrm{p}_{\mathrm{pe}}^{-1}\,\mathrm{pre}\,\mathcal{W}_r$. Further, it is that $s \in \mathrm{pre}\,\mathcal{V}_s$, since $\mathrm{p}_{\mathrm{pe}}s = r \in \mathrm{pre}\,\mathcal{W}_r$ and $s \in \mathrm{pre}((Y_{\mathrm{p}}(Y_{\mathrm{c}}U_{\mathrm{c}})^*U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega$. To show controllability, pick an arbitrary string $\hat{s} \in \mathrm{pre}\,\mathcal{V}_s$ and $\sigma \in \Sigma_{\mathrm{c}} \dot\cup U_{\mathrm{p}} \dot\cup U_{\mathrm{e}}$ such that $\hat{s}\sigma \in \mathrm{pre}\,\mathcal{L}_\Sigma$. In particular, $\mathrm{p}_{\mathrm{pe}}\hat{s} \in \mathrm{p}_{\mathrm{pe}}\mathrm{p}_{\mathrm{pe}}^{-1}\,\mathrm{pre}\,\mathcal{W}_r = \mathrm{pre}\,\mathcal{W}_r$ and $\mathrm{p}_{\mathrm{pe}}(\hat{s}\sigma) \in \mathrm{p}_{\mathrm{pe}}\,\mathrm{pre}\,\mathcal{L}_\Sigma \subseteq \mathrm{pre}\,\mathcal{L}$. Controllability of $\mathrm{pre}\,\mathcal{W}_r$ w.r.t. $\mathrm{pre}\,\mathcal{L}$ implies that $\mathrm{p}_{\mathrm{pe}}(\hat{s}\sigma) \in \mathrm{pre}\,\mathcal{W}_r$. In addition, there exists $u \in \Sigma_{\mathrm{pe}}^\omega$, such that $\mathrm{p}_{\mathrm{pe}}(\hat{s}\sigma)u \in \mathcal{W}_r$. Choose $w \in \Sigma^\omega$ such that $\hat{s}\sigma w \in \mathrm{clo}((Y_{\mathrm{p}}(Y_{\mathrm{c}}U_{\mathrm{c}})^*U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega$ and $\mathrm{p}_{\mathrm{pe}}^\omega(\hat{s}\sigma w) = \mathrm{p}_{\mathrm{pe}}(\hat{s}\sigma)u$. Note that $\hat{s}\sigma w \in \mathcal{V}_s$ and, hence, $\hat{s}\sigma \in \mathrm{pre}\,\mathcal{V}_s$. Finally, relative closedness of $\mathcal{V}_s$ w.r.t. $\mathrm{clo}\,\mathcal{L}_{\mathrm{IO},\bar{\Sigma}}$ has to be established. Since $\mathrm{clo}\,\mathcal{L}_{\mathrm{IO},\bar{\Sigma}}$ is a closed superset of $\mathcal{V}_s$, relative closedness of $\mathcal{V}_s$ and closedness of $\mathcal{V}_s$ are equivalent. This is obtained by

$$\begin{aligned}
\mathcal{V}_s &= (\mathrm{p}_{\mathrm{pe}}^{-\omega}(\mathcal{W}_r \cup \mathrm{pre}\,\mathcal{W}_r)) \cap (\mathrm{clo}((Y_{\mathrm{p}}(Y_{\mathrm{c}}U_{\mathrm{c}})^*U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega) \\
&= (\mathrm{p}_{\mathrm{pe}}^{-\omega}\,\mathrm{clo}\,\mathcal{W}_r \cup \mathrm{p}_{\mathrm{pe}}^{-\omega}\,\mathrm{pre}\,\mathcal{W}_r) \cap (\mathrm{clo}((Y_{\mathrm{p}}(Y_{\mathrm{c}}U_{\mathrm{c}})^*U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega) \\
&= (\mathrm{clo}\,\mathrm{p}_{\mathrm{pe}}^{-\omega}\mathcal{W}_r) \cap (\mathrm{clo}((Y_{\mathrm{p}}(Y_{\mathrm{c}}U_{\mathrm{c}})^*U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega).
\end{aligned}$$

The intersection of two topologically closed languages is topologically closed. □

By the following proposition, the non-anticipation property is also preserved in the full IO-closed-loop behaviour and an additional closed-loop property is obtained.

Proposition 4.2.2

If $\mathcal{H}$ is a solution to the control problem $(\Sigma, \mathcal{L}, \mathcal{E})$, and if $\mathcal{L}$ is a non-anticipating IO-plant, then the IO-closed-loop behaviour satisfies K1–K6:

K1 $\mathcal{K}$ enforces the specification $\mathcal{E}$, i.e., $\mathcal{K} \subseteq \mathcal{E}$,

K2 $\mathcal{K}$ is ω-admissible w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathcal{L}_\Sigma)$,

K3 $\mathcal{K}$ is ω-normal w.r.t. $(\Sigma_{\mathrm{o}}, \mathcal{L}_\Sigma)$,

K4 $\mathrm{pre}\,\mathcal{K}$ is normal w.r.t. $(\Sigma_{\mathrm{o}}, \mathrm{pre}\,\mathcal{L}_\Sigma)$,

K5 $\mathrm{pre}\,\mathcal{K}$ possesses locally free inputs U_{c} and U_{e}.

K6 $\mathcal{K}$ is ω-controllable w.r.t. $(U_{\mathrm{c}} \dot\cup U_{\mathrm{e}},\, \mathrm{clo}\,\mathcal{K})$.

Proof. K1–K5 have been readily verified in Proposition 3.2.2. Regarding K6, the claim is proven by construction of a suitable $\mathcal{V}_s \subseteq \mathcal{K}$ for an arbitrarily chosen $s \in \operatorname{pre}\mathcal{K}$. Note that, by Lemma 4.2.1, $\mathcal{L}_\Sigma$ is a non-anticipating IO-plant. Thus, choose $\tilde{\mathcal{V}}_s \subseteq \mathcal{L}_\Sigma$, such that $s \in \operatorname{pre}\tilde{\mathcal{V}}_s$, $\operatorname{pre}\tilde{\mathcal{V}}_s$ is controllable w.r.t. $(\Sigma_c \dot\cup U_p \dot\cup U_e, \operatorname{pre}\mathcal{L}_\Sigma)$, and $\tilde{\mathcal{V}}_s$ is relatively closed w.r.t. $\operatorname{clo}\mathcal{L}_\Sigma$. In particular, $\tilde{\mathcal{V}}_s$ is closed. By Proposition 3.2.2, $\mathcal{K}$ satisfies K1–K5. Referring to K2, choose $\mathcal{W}_s \subseteq \mathcal{K}$ with $s \in \operatorname{pre}\mathcal{W}_s$, and $\operatorname{pre}\mathcal{W}_s$ is controllable w.r.t. $(\Sigma_{uc}, \operatorname{pre}\mathcal{L}_\Sigma)$, and $\mathcal{W}_s$ is relatively closed w.r.t. $\mathcal{L}_\Sigma$. To establish ω-controllability of $\mathcal{K}$ w.r.t. $\operatorname{clo}\mathcal{K}$, consider the candidate $\mathcal{V}_s := \tilde{\mathcal{V}}_s \cap \mathcal{W}_s$. Clearly, $\mathcal{V}_s \subseteq \mathcal{K}$. Furthermore, $\mathcal{V}_s = \tilde{\mathcal{V}}_s \cap \mathcal{W}_s = \tilde{\mathcal{V}}_s \cap (\operatorname{clo}\mathcal{W}_s) \cap \mathcal{L}_\Sigma = \tilde{\mathcal{V}}_s \cap (\operatorname{clo}\mathcal{W}_s) = (\operatorname{clo}\tilde{\mathcal{V}}_s) \cap (\operatorname{clo}\mathcal{W}_s) \supseteq \operatorname{clo}\mathcal{V}_s$, i.e., $\mathcal{V}_s$ is closed and, thus, relatively closed w.r.t. any superset. To show controllability of $\operatorname{pre}\mathcal{V}_s$ w.r.t. $\operatorname{pre}\mathcal{K}$, pick $r \in \operatorname{pre}(\tilde{\mathcal{V}}_s \cap \mathcal{W}_s) \subseteq (\operatorname{pre}\tilde{\mathcal{V}}_s) \cap (\operatorname{pre}\mathcal{W}_s)$ and $\sigma \in U_c \dot\cup U_e$ such that $r\sigma \in \operatorname{pre}\mathcal{K} \subseteq \operatorname{pre}\mathcal{L}_\Sigma$. By controllability of $\operatorname{pre}\tilde{\mathcal{V}}_s$ and $\operatorname{pre}\mathcal{W}_s$, it follows that $r\sigma \in (\operatorname{pre}\tilde{\mathcal{V}}_s) \cap (\operatorname{pre}\mathcal{W}_s)$. To establish $r\sigma \in \operatorname{pre}(\tilde{\mathcal{V}}_s \cap \mathcal{W}_s)$, observe that each event in Σ is uncontrollable for either $\operatorname{pre}\tilde{\mathcal{V}}_s$ or $\operatorname{pre}\mathcal{W}_s$. Thus, starting with $r_0 = r\sigma$, an unbounded sequence $(r_n) \subseteq (\operatorname{pre}\tilde{\mathcal{V}}_s) \cap (\operatorname{pre}\mathcal{W}_s)$, with limit $w := \lim(r_n) \in (\operatorname{clo}\tilde{\mathcal{V}}_s) \cap (\operatorname{clo}\mathcal{W}_s)$, can be constructed. Since $\tilde{\mathcal{V}}_s$ is closed, it is $w \in \tilde{\mathcal{V}}_s \subseteq \mathcal{L}_\Sigma$. By relative closedness of $\mathcal{W}_s$ w.r.t. $\mathcal{L}_\Sigma$, it follows that $w \in \mathcal{W}_s$. Hence, $r\sigma \in \operatorname{pre}(\tilde{\mathcal{V}}_s \cap \mathcal{W}_s)$. □

4.2.2 Propagation of plant properties

Coming back to the first of the two questions, asked in the beginning of the present Subsection, it is now possible to show that the plant properties P1–P3 are retained under closed-loop composition. In particular, the external closed-loop behaviour that results from the presented controller design is again a non-anticipating IO-plant.

Theorem 4.2.3 (Baier and Moor, 2012)

For a non-anticipating IO-plant $\mathcal{L}$ and an IO-specification $\mathcal{E}$, consider a solution $\mathcal{H}$ of the control problem $(\Sigma, \mathcal{L}, \mathcal{E})$. Then the external closed-loop $\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{K}$, with $\mathcal{K} = \mathcal{L}_\Sigma \cap \mathcal{H}_\Sigma$, is a non-anticipating IO-plant, too.

Proof. Regarding the event ordering P1, refer to K1 and E1 to obtain $\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{K} \subseteq ((Y_c U_c)^*(Y_e U_e)^*)^{\omega}$. Regarding locally free inputs P2, recall from K5 that $\mathcal{K}$ has locally free inputs U_c and U_e, that are preserved under projection to Σ_{ce}. It is left to verify non-anticipation P3. Pick $s \in \operatorname{pre}\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{K}$. Then, there exists $t \in \operatorname{pre}\mathcal{K}$ such that $\mathrm{p}_{\mathrm{ce}}t = s$. According to K6, it is possible to choose $\mathcal{W}_t \subseteq \mathcal{K}$ such that $t \in \operatorname{pre}\mathcal{W}_t$, and $\operatorname{pre}\mathcal{W}_t$ is controllable w.r.t. $(U_c \dot\cup U_e, \operatorname{pre}\mathcal{K})$, and $\mathcal{W}_t$ is closed. As a candidate to establish P3, let $\mathcal{V}_s := \mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{W}_t$. Note that $\mathcal{V}_s \subseteq \mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{K}$. Further, $s = \mathrm{p}_{\mathrm{ce}}t \in \mathrm{p}_{\mathrm{ce}}\operatorname{pre}\mathcal{W}_t = \operatorname{pre}\mathcal{V}_s$. To verify controllability of $\operatorname{pre}\mathcal{V}_s$, consider an arbitrary $\hat{s} \in \operatorname{pre}\mathcal{V}_s$ and $\sigma \in U_c \dot\cup U_e$ such that $\hat{s}\sigma \in \operatorname{pre}\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{K}$. Then, there exists $\hat{t} \in \operatorname{pre}\mathcal{K}$ such that $\mathrm{p}_{\mathrm{ce}}\hat{t} = \hat{s}$ and $\hat{t} \in \operatorname{pre}\mathcal{W}_t$. Furthermore, $\hat{t}\sigma \in \operatorname{pre}\mathcal{K}$, since $\hat{s}\sigma = \mathrm{p}_{\mathrm{ce}}(\hat{t}\sigma) \in \mathrm{p}_{\mathrm{ce}}\operatorname{pre}\mathcal{K}$. Controllability of $\operatorname{pre}\mathcal{W}_t$

implies that $\hat{t}\sigma \in \operatorname{pre}\mathcal{W}_t$ and $\mathrm{p}_{\mathrm{ce}}(\hat{t}\sigma) \in \mathrm{p}_{\mathrm{ce}}\operatorname{pre}\mathcal{W}_t = \operatorname{pre}\mathcal{V}_s$. Consequently, the candidate $\operatorname{pre}\mathcal{V}_s$ is controllable w.r.t. $(U_{\mathrm{c}} \dot\cup U_{\mathrm{e}}, \operatorname{pre}\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{K})$. To verify closedness of $\mathcal{V}_s$, observe that $\mathcal{V}_s = \mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{W}_t = (\mathrm{p}^{\omega}_{\mathrm{ce}}\operatorname{clo}\mathcal{W}_t) \cap \Sigma^{\omega}_{\mathrm{ce}} = \operatorname{clo}\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{W}_t$. □

As a consequence of this result, the external closed-loop behaviour $\mathcal{L}^{\mathrm{hi}}$ satisfies the properties requested from the IO-plant on the next higher controller design level.

4.2.3 Abstraction-based controller design

Although the external closed-loop behaviour $\mathcal{L}^{\mathrm{hi}} = \mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{K}$ could be used for the next design step, for computational reasons, it is preferable to use the abstraction $\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{E}$. However, from the abstraction-based control problem for $*$-languages, it is known that the major challenge is to guarantee a non-conflicting behaviour. Hence, following the argumentation in (Moor et al., 2011), first it is analysed, whether the behaviour of the full IO-plant $\mathcal{L}_\Sigma$ can be represented as a union of languages with particular properties that support the argumentation of the abstraction-based controller design; see also (Baier and Moor, 2012).

Lemma 4.2.4

For a non-anticipating IO-plant $\mathcal{L}$, the full IO-plant behaviour can be represented as a union $\mathcal{L}_\Sigma = \cup_{a\in A}\mathcal{L}_a$, where for all $a \in A$

(i) $\mathcal{L}_a$ has locally free inputs U_{c}, $U_{\mathrm{p}} \dot\cup Y_{\mathrm{c}}$, and U_{e}.

(ii) $\mathcal{L}_a$ is closed.

Proof. Technically, P3 together with (Baier and Moor, 2012), Proposition 9, implies that $\mathcal{L}_\Sigma$ itself is the supremal ω-controllable sub-language of $\mathcal{L}_\Sigma$. Thus, by (Moor et al., 2011), Proposition 12, $\mathcal{L}_\Sigma$ can be represented as a union $\mathcal{L}_\Sigma = \cup_{a\in A}\mathcal{L}_a$, where, for each, $a \in A$, $\operatorname{pre}\mathcal{L}_a$ is controllable w.r.t. $(\Sigma_{\mathrm{c}} \dot\cup U_{\mathrm{p}} \dot\cup U_{\mathrm{e}}, \operatorname{pre}\mathcal{L}_\Sigma)$ and $\mathcal{L}_a$ is closed. To establish (i), pick $s \in \Sigma^*$, $\mu, \mu' \in U_{\mathrm{c}}$, with $s\mu \in \operatorname{pre}\mathcal{L}_a$. The locally free input of $\operatorname{pre}\mathcal{L}_\Sigma$ implies $s\mu' \in \operatorname{pre}\mathcal{L}_\Sigma$, and controllability of $\operatorname{pre}\mathcal{L}_a$ w.r.t. $\operatorname{pre}\mathcal{L}_\Sigma$ implies, that $s\mu' \in \operatorname{pre}\mathcal{L}_a$. Locally free inputs $U_{\mathrm{p}} \dot\cup Y_{\mathrm{c}}$, and U_{e} are verified likewise. □

Next, it is analysed, whether it is possible to implement a non-conflicting closed-loop behaviour w.r.t. the original IO-plant based on a solution that has been designed on the basis of an abstraction.

Lemma 4.2.5

Given a control problem $(\Sigma, \mathcal{L}, \mathcal{E})$ with a non-anticipating IO-plant $\mathcal{L}$, let $\mathcal{L}' \subseteq \Sigma^\omega$ denote a plant abstraction, i.e., $\mathcal{L} \subseteq \mathcal{L}'$. Consider any solution $\mathcal{H}$ of the control problem $(\Sigma, \mathcal{L}', \mathcal{E})$. If $\mathcal{V}' \subseteq \mathcal{L}'_\Sigma \cap \mathcal{H}_\Sigma$, and if $\operatorname{pre} \mathcal{V}'$ is controllable w.r.t. $(\Sigma_{\text{uc}}, \mathcal{L}'_\Sigma)$, and if $\mathcal{V}'$ is relatively closed w.r.t. $\mathcal{L}'_\Sigma$, then $\mathcal{L}_\Sigma$ and $\mathcal{V}'$ are non-conflicting.

Proof. For $\mathcal{V}' \subseteq \mathcal{L}'_\Sigma \cap \mathcal{H}_\Sigma$ it has to be shown that, if $\operatorname{pre} \mathcal{V}'$ is controllable w.r.t. $(\Sigma_{\text{uc}}, \mathcal{L}'_\Sigma)$, and if $\mathcal{V}'$ is relatively closed w.r.t. $\mathcal{L}'_\Sigma$, then $\mathcal{L}_\Sigma$ and $\mathcal{V}'$ are non-conflicting, i.e., that $\operatorname{pre}(\mathcal{L}_\Sigma \cap \operatorname{pre} \mathcal{V}') = (\operatorname{pre} \mathcal{L}_\Sigma) \cap (\operatorname{pre} \mathcal{V}')$. Pick an arbitrary string $s \in (\operatorname{pre} \mathcal{L}_\Sigma) \cap (\operatorname{pre} \mathcal{V}')$. Referring to Lemma 4.2.4, represent $\mathcal{L}_\Sigma$ as $\mathcal{L}_\Sigma = \cup_{a \in A} \mathcal{L}_a$ with $\mathcal{L}_a$ satisfying conditions (i) and (ii). In particular, there exists $a \in A$ with $s \in \operatorname{pre} \mathcal{L}_a \subseteq \mathcal{L}_\Sigma \subseteq \mathcal{L}'_\Sigma$. To extend $s \in (\operatorname{pre} \mathcal{L}_a) \cap (\operatorname{pre} \mathcal{V}')$ by one event, pick σ such that $s\sigma \in \operatorname{pre} \mathcal{L}_a$. If $\sigma \in \Sigma_{\text{uc}}$, then controllability of $\operatorname{pre} \mathcal{V}'$ implies $s\sigma \in \operatorname{pre} \mathcal{V}'$, and thus $s\sigma \in (\operatorname{pre} \mathcal{L}_a) \cap (\operatorname{pre} \mathcal{V}')$. If, on the other hand, $\sigma \in U_\text{p} \dot{\cup} Y_\text{c}$, Lemma 4.2.4, condition (i), implies that $s(U_\text{p} \dot{\cup} Y_\text{c}) \subseteq \operatorname{pre} \mathcal{L}_a$. Referring to the event ordering in the definition of $\mathcal{L}_\Sigma$, decompose $s = r\nu$ with $\nu \in U_\text{c} \dot{\cup} Y_\text{p}$. Again by the definition of $\mathcal{L}_\Sigma$, now using $r\nu \in \operatorname{pre} \mathcal{V}' \subseteq \operatorname{pre} \mathcal{L}'_\Sigma$, the existence of $\sigma \in U_\text{p} \dot{\cup} Y_\text{c}$ is obtained such that $s\sigma \in \operatorname{pre} \mathcal{V}'$ and, thus, conclude with $s\sigma \in (\operatorname{pre} \mathcal{L}_a) \cap (\operatorname{pre} \mathcal{V}')$. Repeatedly extending s, a strictly monotone sequence $(s_n) \subseteq (\operatorname{pre} \mathcal{L}_a) \cap (\operatorname{pre} \mathcal{V}')$ is constructed, with limit $w := \lim(s_n) \in (\operatorname{clo} \mathcal{L}_a) \cap (\operatorname{clo} \mathcal{V}')$ and $s = s_0 < w$. Since $\mathcal{L}_a$ is closed, $w \in \mathcal{L}_a$ is obtained to observe $w \in \mathcal{L}_a \cap (\operatorname{clo} \mathcal{V}') \subseteq \mathcal{L}_\Sigma \cap \mathcal{L}'_\Sigma \cap (\operatorname{clo} \mathcal{V}') = \mathcal{L}_\Sigma \cap \mathcal{V}'$. In particular, $s \in \operatorname{pre}(\mathcal{L}_\Sigma \cap \mathcal{V}')$. □

Any controller candidate $\mathcal{V}'$, found by using the abstraction $\mathcal{L}'$ as plant, will result in a non-conflicting closed loop composed from the original plant $\mathcal{L}$ and the candidate $\mathcal{V}'$. Hence, the theorem regarding abstraction-based controller design is adapted from (Moor et al., 2011) to the closed-loop configuration with external signals; see Figure 4.6.

Theorem 4.2.6 (Baier and Moor, 2012)

Given a control problem $(\Sigma, \mathcal{L}, \mathcal{E})$ with a non-anticipating IO-plant $\mathcal{L}$, let $\mathcal{L}' \subseteq \Sigma^\omega$ denote a plant abstraction, i.e., $\mathcal{L} \subseteq \mathcal{L}'$. Then, any solution $\mathcal{H}$ of $(\Sigma, \mathcal{L}', \mathcal{E})$ also solves $(\Sigma, \mathcal{L}, \mathcal{E})$.

Proof. Note that $\mathcal{H}$ trivially enforces the specification H1, since $\mathcal{L}_\Sigma \cap \mathcal{H}_\Sigma \subseteq \mathcal{L}'_\Sigma \cap \mathcal{H}_\Sigma \subseteq \mathcal{E}$. It is left to verify admissibility H2. Pick an arbitrary $s \in (\operatorname{pre} \mathcal{L}_\Sigma) \cap (\operatorname{pre} \mathcal{H}_\Sigma)$. Since $\mathcal{H}$ is a solution to $(\Sigma, \mathcal{L}', \mathcal{E})$, choose $\mathcal{V}'_s \subseteq \mathcal{L}'_\Sigma \cap \mathcal{H}_\Sigma$ such that $s \in \operatorname{pre} \mathcal{V}'_s$, and $\operatorname{pre} \mathcal{V}'_s$ is controllable w.r.t. $(\Sigma_{\text{uc}}, \operatorname{pre} \mathcal{L}'_\Sigma)$, and $\operatorname{pre} \mathcal{V}'_s$ is prefix-normal w.r.t. $(\Sigma_\text{o}, \operatorname{pre} \mathcal{L}'_\Sigma)$, and $\mathcal{V}'_s$ is relatively closed w.r.t. $\mathcal{L}'_\Sigma$. The candidate $\mathcal{V}_s := \mathcal{V}'_s \cap \mathcal{L}_\Sigma$ is chosen. Observe that $\mathcal{V}_s \subseteq \mathcal{L}'_\Sigma \cap \mathcal{H}_\Sigma \cap \mathcal{L}_\Sigma = \mathcal{H}_\Sigma \cap \mathcal{L}_\Sigma$ and $s \in (\operatorname{pre} \mathcal{L}_\Sigma) \cap (\operatorname{pre} \mathcal{H}_\Sigma) \cap (\operatorname{pre} \mathcal{V}'_s)$. Lemma 4.2.5 implies that $s \in \operatorname{pre}(\mathcal{L}_\Sigma \cap \mathcal{V}'_s) = \operatorname{pre} \mathcal{V}_s$. Regarding controllability, pick any $s\nu \in \operatorname{pre} \mathcal{L}_\Sigma$ with $s \in \operatorname{pre} \mathcal{V}_s$ and $\nu \in \Sigma_{\text{uc}}$. By controllability of $\operatorname{pre} \mathcal{V}'_s$ w.r.t. $\operatorname{pre} \mathcal{L}'_\Sigma$ and $\mathcal{L}_\Sigma \subseteq \mathcal{L}'_\Sigma$, it can be deduced that $sv \in \operatorname{pre}\mathcal{V}_s'$. Again by Lemma 4.2.5 follows that $sv \in \operatorname{pre}(\mathcal{L}_\Sigma \cap \mathcal{V}_s')$. Hence, $\operatorname{pre}\mathcal{V}_s$ is controllable w.r.t. $\operatorname{pre}\mathcal{L}_\Sigma$. Regarding prefix-normality, first observe that $(\operatorname{pre}\mathcal{L}_\Sigma) \cap (\mathrm{p}_{\mathrm{cp}}^{-1}\mathrm{p}_{\mathrm{cp}} \operatorname{pre}\mathcal{V}_s) \subseteq (\operatorname{pre}\mathcal{L}_\Sigma) \cap (\mathrm{p}_{\mathrm{cp}}^{-1}\mathrm{p}_{\mathrm{cp}} \operatorname{pre}\mathcal{V}_s') = (\operatorname{pre}\mathcal{L}_\Sigma) \cap (\operatorname{pre}\mathcal{L}_\Sigma') \cap (\mathrm{p}_{\mathrm{cp}}^{-1}\mathrm{p}_{\mathrm{cp}} \operatorname{pre}\mathcal{V}_s') = (\operatorname{pre}\mathcal{L}_\Sigma) \cap (\operatorname{pre}\mathcal{V}_s') = \operatorname{pre}(\mathcal{L}_\Sigma \cap \mathcal{V}_s') = \operatorname{pre}\mathcal{V}_s$, where the last equality is by Lemma 4.2.5. Together with $\operatorname{pre}V_s \subseteq \operatorname{pre}\mathcal{L}_\Sigma$, this constitutes prefix-normality of $\operatorname{pre}\mathcal{V}_s$ w.r.t. $(\Sigma_o, \mathcal{L}_\Sigma)$. Regarding relative closedness, observe that $(\operatorname{clo}\mathcal{V}_s) \cap \mathcal{L}_\Sigma \subseteq (\operatorname{clo}\mathcal{V}_s') \cap \mathcal{L}_\Sigma' \cap \mathcal{L}_\Sigma = \mathcal{V}_s' \cap \mathcal{L}_\Sigma = \mathcal{V}_s$, concluding the proof of H2. □

Finally, an affirmative answer to both questions can be provided. Propagation of plant properties by the IO-closed-loop behaviour is given and an abstraction-based controller design is possible due to the additional plant property P3. Hence, Theorem 4.2.3 and Theorem 4.2.6 formally justify the hierarchical controller design in Figure 4.6.

4.3 Hierarchical and modular control problem for input/output systems

In addition to the hierarchical abstraction-based controller design, the idea of modular controller design shall be transferred to the framework of input/output systems; see Figure 4.9.

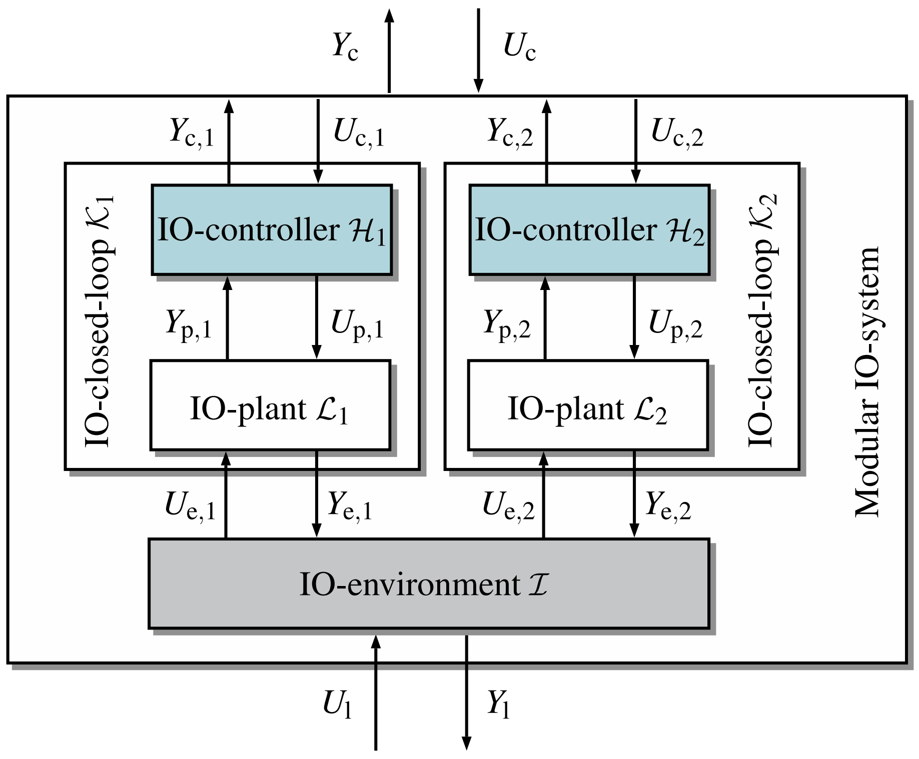


Figure 4.9: Modular control problem

In accordance with the modular control approach for $*$-languages, the given system consists of several independent non-anticipating IO-plants and for each IO-plant a local closed loop is designed. By definition the given IO-plants do not share any events, i.e. the alphabets of the closed loops are disjoint. Thereby, the composition of both closed loops is non-blocking, by definition. However, physical dependencies between both systems exist and can be considered by means of an additional model called *IO-environment*. Basically, the IO-environment interacts with the IO-plants in a similar fashion as the IO-controller. Hence, in order to guarantee non-conflicting behaviour, particular properties have to be imposed on this model. The behaviour of the composed system is described by means of a particular construction, introduced as *modular IO-system*. Following the ideas of modular and hierarchical control, the external behaviour of the composed system shall be reused as IO-plant for the high-level control problem. Thus, it has to be analysed whether the IO-plant properties P1 - P3, that are requested in the basic control problem, are retained under *modular IO-system* composition. In particular, it has to be shown that they are satisfied by the external behaviour of the IO-system, which shall be used for the controller design on the high-level.

In the following it is explained step-by-step how to compose IO-plants. From Theorem 4.2.3 it is known that the external closed-loop behaviour satisfies again the plant properties P1 - P3. Hence, instead of considering the local IO-closed-loops, they are replaced by IO-plants to simplify further argumentations. Properties of the IO-environment are introduced and the composition of IO-plants and the IO-environment to a modular IO-system is defined. Finally, it is verified that the IO-plant properties are retained under the proposed system composition. Thus, the resulting external behaviour is again an IO-plant, and can be subject to subsequent controller design.

4.3.1 Properties of the IO-system composition

Figure 4.10 shows the basic composition that is discussed in this Subsection. Several IO-plants are composed by a shuffle construction. The IO-environment serves as a model to further restrict the resulting behaviour regarding physical dependencies of the given IO-plants.

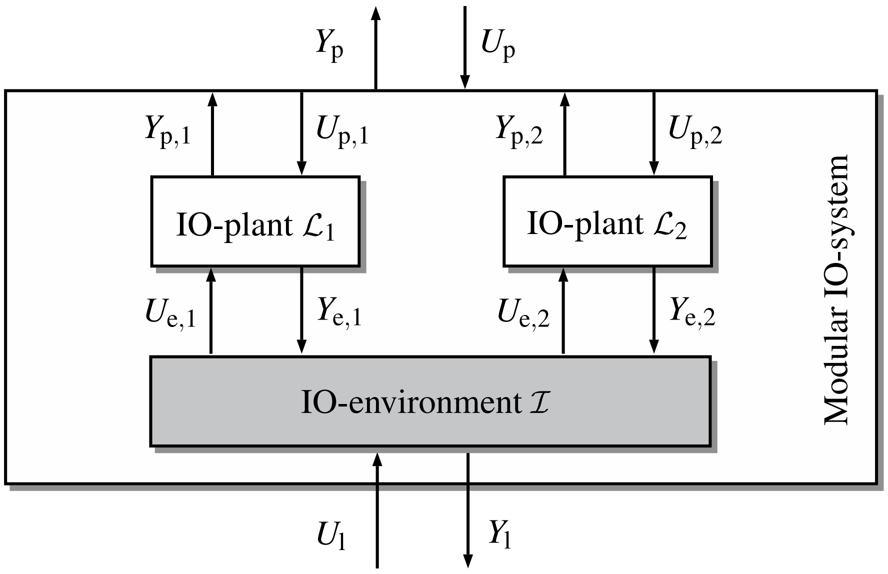


Figure 4.10: Modular IO-System

In the first step, just the IO-plants are composed and the IO-environment is let aside.

Definition 4.3.1 IO-plant ω-composition

Given a number $m \in \mathbb{N}$ of IO-plants $\mathcal{L}_n \subseteq \Sigma_n^\omega$, with $\Sigma_n = U_{p,n} \dot\cup Y_{p,n} \dot\cup U_{e,n} \dot\cup Y_{e,n}$, $n = 1,\ldots,m$. The *IO-plant ω-composition* is defined by

$$\mathcal{L}_{\|} := \mathcal{L}_1 \parallel \cdots \parallel \mathcal{L}_m := \{ w \in ((Y_p U_p)^*(Y_e U_e)^*)^\omega \mid p_n^\omega w \in \mathcal{L}_n \text{ for } n = 1,\ldots,m \},$$

with the composed plant alphabet $\Sigma_{pe} := \Sigma_1 \dot\cup \cdots \dot\cup \Sigma_m$, and the input and output alphabets $U_p := \dot\bigcup_{n=1}^m U_{p,n}$, $Y_p := \dot\bigcup_{n=1}^m Y_{p,n}$, $U_e := \dot\bigcup_{n=1}^m U_{e,n}$, $Y_e := \dot\bigcup_{n=1}^m Y_{e,n}$. □

Since all alphabets are assumed to be disjoint, the above IO-plant ω-composition is a simple shuffle product of ω-languages. However, in order to maintain the structural IO-plant requirements regarding input/output alternation in the composed system, it is necessary to restrict the shuffle accordingly and demand the composed system to be a subset of $((Y_p U_p)^*(Y_e U_e)^*)^\omega$. This restriction is possible, since an arbitrary output of any plant is always followed by an input selected by the operating controller. As a consequence, it is just demanded that the controller, which operates a composed system, replies instantaneously to any output event by an input event directed to the respective plant component. However, locally free inputs are obviously not given, since the union of the respective inputs of all plant components is considered as inputs of the composed system. In order to guarantee locally free inputs with respect to the composed system, the union is taken with an artificial *error behaviour* that accounts for missing input events.

Definition 4.3.2 IO-shuffle

Given a number $m \in \mathbb{N}$ of IO-plants $\mathcal{L}_n \subseteq \Sigma_n^\omega$, with $\Sigma_n = U_{p,n} \dot\cup Y_{p,n} \dot\cup U_{e,n} \dot\cup Y_{e,n}$, $n = 1, \dots, m$. Then, the *IO-shuffle* is defined by

$$\mathcal{L}_{IO} := \mathcal{L}_1 \,\|_{IO} \cdots \|_{IO}\, \mathcal{L}_m := \mathcal{L}_\| \cup \mathcal{L}_{err} \subseteq \Sigma_{pe}^\omega ,$$

with the IO-plant ω-composition $\mathcal{L}_\|$ and the *error behaviour*

$$\mathcal{L}_{err} := \bigcup_{x\in\{e,p\}} \bigcup_{\substack{k=1,\\ k\neq n}}^{m} \bigcup_{n=1}^{m} ((\Sigma_{pe}^* Y_{x,n}) \cap (\operatorname{pre}\mathcal{L}_\|)) U_{x,k} ((Y_p U_p)^* (Y_e U_e)^*)^\omega$$

For easier reference, replace $\Cup := \bigcup_{x\in\{e,p\}} \bigcup_{k=1,k\neq n}^{m} \bigcup_{n=1}^{m}$. □

Note that any relevant specification will implicitly prevent the controller from issuing miss-directed input events in order to avoid the error behaviour in the closed-loop configuration. Before the composition is continued and the *IO-environment* is added, it is analysed whether the IO-shuffle preserves the IO-plant properties P1–P3.

Proposition 4.3.1

Given $m \in \mathbb{N}$ non-anticipating IO-plants $\mathcal{L}_n \subseteq \Sigma_n^\omega$, $n = 1, \dots, m$, then the IO-shuffle $\mathcal{L}_{IO} := \mathcal{L}_1 \,\|_{IO} \cdots \|_{IO}\, \mathcal{L}_n$ is a non-anticipating IO-plant, too.

Proof. Regarding the event ordering P1, it is given by definition that $\mathcal{L}_\| \subseteq [(Y_p U_p)^* (Y_e U_e)^*]^\omega$. Referring to the definition of $\mathcal{L}_{err}$, this implies that $\mathcal{L}_{err} \subseteq [(Y_p U_p)^* (Y_e U_e)^*]^\omega$, and, hence, $\mathcal{L}_{IO} \subseteq [(Y_p U_p)^* (Y_e U_e)^*]^\omega$. Regarding locally free inputs, property P2, the attention is focused on the input alphabet U_p, pick an arbitrary $s\mu \in \operatorname{pre}\mathcal{L}_{IO}$, with $\mu \in U_p$, and an alternative input symbol $\mu' \in U_p$. By P1, decompose $s = t\nu$ with $\nu \in Y_{p,n}$ for some n. If $t\nu \in \mathcal{L}_{err}$, it follows that $t\nu\mu' \in \operatorname{pre}\mathcal{L}_{err} \subseteq \operatorname{pre}\mathcal{L}_{IO}$ by the definition of $\mathcal{L}_{err}$, and, hence, $t\nu\mu' \in \mathcal{L}_{IO}$. Else, $t\nu \notin \mathcal{L}_{err}$ and, consequently, $t\nu \in \operatorname{pre}\mathcal{L}_\|$ by the definition of $\mathcal{L}_{IO}$. Here, two more cases can be distinguished. First, if $\mu' \in U_{p,n}$, the locally free input $U_{p,n}$ of $\operatorname{pre}\mathcal{L}_n$ implies that $p_n(s)\nu\mu' \in \operatorname{pre}\mathcal{L}_n$ and, thus $t\nu\mu' \in \operatorname{pre}\mathcal{L}_\| \subseteq \operatorname{pre}\mathcal{L}_{IO}$. In the the second case it is $\mu' \in U_{p,k}$ for some $k \neq n$, and again obtain $t\nu\mu' \in \operatorname{pre}\mathcal{L}_{err} \subseteq \operatorname{pre}\mathcal{L}_{IO}$. This establishes that U_p is a locally free input of $\mathcal{L}_{IO}$. The free input U_e is verified likewise, concluding the proof of P2. Regarding non-anticipation, property P3, ω-controllability of $\mathcal{L}_{IO}$ w.r.t. $(U_p \dot\cup U_e, \operatorname{clo}\mathcal{L}_{IO})$ has to be verified. Pick an arbitrary string $t \in \operatorname{pre}\mathcal{L}_{IO}$ and denote by s the longest prefix of t such that $s \in \operatorname{pre}\mathcal{L}_{IO}$. In the following, it is first constructed a candidate $\mathcal{V}_s$ for s, relevant properties of $\mathcal{V}_s$ are analysed, a respective error behaviour for $\mathcal{V}_s$ is constructed and, finally, a candidate $\mathcal{V}_t$ for t is deduced that proves ω-controllability. To find a candidate for s, denote $r_n := p_n s$ for $n = 1, \dots, m$. Choose $w \in \mathcal{L}_\|$, $s < w$, to observe $p_n^\omega w \in \mathcal{L}_n$ and, hence, $r_n \in \operatorname{pre}\mathcal{L}_n$, for all $n = 1,\dots,m$. Since each $\mathcal{L}_n$ is non-anticipating, it is possible to choose $\mathcal{V}_{r,n} \subseteq \mathcal{L}_n$ with $r_n \in \operatorname{pre}\mathcal{V}_{r,n}$ and $\operatorname{pre}\mathcal{V}_{r,n}$ satisfies A1 and A2. In particular, $\mathcal{V}_{r,n}$ is closed. For the string s, define the candidate $\mathcal{V}_s := \left(\bigcap_{n=1}^{m} \operatorname{clo} \mathrm{p}_{\mathrm{n}}^{-\omega}\mathcal{V}_{r,n}\right) \cap \mathcal{L}_{\mathrm{seq}}$ with $\mathcal{L}_{\mathrm{seq}} := (\Sigma_1\Sigma_1 + \ldots + \Sigma_m\Sigma_m)^\omega \cap \Sigma^{|s|}(\varepsilon+\Sigma)(\Sigma_1\Sigma_1\cdots\Sigma_m\Sigma_m)^\omega$. Note that, as a finite intersection of closed languages, $\mathcal{V}_s$ itself is closed. To show that $\mathcal{V}_s \subseteq \mathcal{L}_{\mathrm{IO}}$, pick $w \in \mathcal{V}_s$ and n arbitrarily. By $w \in \operatorname{clo}\mathrm{p}_n^{-\omega}\mathcal{V}_{r,n}$, it is $\operatorname{pre} w \subseteq \operatorname{pre}\mathrm{p}_n^{-\omega}\mathcal{V}_{r,n}$ and, hence, $\mathrm{p}_n \operatorname{pre} w \subseteq \operatorname{pre}\mathcal{V}_{r,n}$. Referring to the definition of $\mathcal{L}_{\mathrm{seq}}$, $\mathrm{p}_n \operatorname{pre} w$ is unbounded. This implies that $\mathrm{p}_n^\omega w = \lim \mathrm{p}_n \operatorname{pre} w \subseteq \lim \operatorname{pre}\mathcal{V}_{r,n} = \mathcal{V}_{r,n}$, i.e., $\mathrm{p}_n^\omega w \in \mathcal{V}_{r,n}$. By the arbitrary choice of w and n, it follows that $\mathcal{V}_s \subseteq \mathcal{L}_{\|} \subseteq \mathcal{L}_{\mathrm{IO}}$. Based on $\mathcal{V}_s$ an error behaviour is constructed $\mathcal{V}_{s,Err} := \uplus\left((\Sigma_{\mathrm{pe}}^* Y_{\mathrm{x},n}) \cap (\operatorname{pre}\mathcal{V}_s)\right) U_{\mathrm{x},k}((Y_{\mathrm{p}}U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega$ The candidate for t is based on $\mathcal{V}_{s,Err}$. Define $\mathcal{V}_t := \operatorname{clo}\mathcal{V}_{s,Err}$. To show that $t \in \operatorname{pre}\mathcal{V}_t$, is is analysed whether $s \in \operatorname{pre}\mathcal{V}_s$. Therefore, an arbitrary $\hat{s} \in \operatorname{pre}\mathcal{L}_{\mathrm{seq}}$ is selected. It is claimed that $\hat{r}_n := \mathrm{p}_{\mathrm{n}}\hat{s} \in \operatorname{pre}\mathcal{V}_{r,n}$ for all n implies $\hat{s} \in \operatorname{pre}\mathcal{V}_s$. From $\hat{r}_n \in \operatorname{pre}\mathcal{V}_{r,n}$, choose $\hat{u}_n \in \Sigma_n^\omega$ such that $\hat{r}_n\hat{u}_n \in \mathcal{V}_{r,n}$. The alphabets Σ_1 to Σ_m are disjoint, and it is possible to choose $\hat{u}$ in the shuffle $\mathrm{p}_1^{-\omega}\hat{u}_1 \cap \cdots \cap \mathrm{p}_{\mathrm{m}}^{-\omega}\hat{u}_m$ such that $\mathrm{p}_{\mathrm{n}}^\omega(\hat{s}\hat{u}) = \hat{r}_n\hat{u}_n \in \mathcal{V}_{r,n}$ and $\hat{s}\hat{u} \in \mathcal{L}_{\mathrm{seq}}$. Thus, it follows indeed that $\hat{s} \in \operatorname{pre}\mathcal{V}_s$. As an immediate consequence, $s \in \operatorname{pre}\mathcal{V}_s$. By definition $t/s \in \operatorname{pre}(\uplus U_{\mathrm{x},k}((Y_{\mathrm{p}}U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega) \subseteq \mathcal{V}_{s,Err}/s$. Hence, $t \in \mathcal{V}_t$. To show that $\mathcal{V}_t \subseteq \mathcal{L}_{\mathrm{IO}}$, observe that $\mathcal{V}_t$ can be divided into $\mathcal{V}_t = \mathcal{V}_{t,1} \cup \mathcal{V}_{t,2}$, with $\mathcal{V}_{t,1} := \uplus(\operatorname{clo}((\Sigma_{\mathrm{pe}}^* Y_{\mathrm{x},n}) \cap (\operatorname{pre}\mathcal{V}_s)))$ and $\mathcal{V}_{t,2} := \uplus(((\Sigma_{\mathrm{pe}}^* Y_{\mathrm{x},n}) \cap (\operatorname{pre}\mathcal{V}_s)) \operatorname{clo} U_{\mathrm{x},k}((Y_{\mathrm{p}}U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega)$. Obviously $\mathcal{V}_{t,1} \subseteq \lim \operatorname{pre}\mathcal{V}_s = \mathcal{V}_s \subseteq \mathcal{L}_{\|} \subseteq \mathcal{L}_{\mathrm{IO}}$. As a consequence and by definition, $\mathcal{V}_{t,2} = \uplus\left((\Sigma_{\mathrm{pe}}^* Y_{\mathrm{x},n}) \cap (\operatorname{pre}\mathcal{V}_s)\right) U_{\mathrm{x},k}((Y_{\mathrm{p}}U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*)^\omega \subseteq \mathcal{L}_{\mathrm{err}}$. Thus $\mathcal{V}_t \subseteq \mathcal{L}_{\mathrm{IO}}$. Regarding controllability of $\operatorname{pre}\mathcal{V}_t$ w.r.t. $(U_{\mathrm{p}} \dot\cup U_{\mathrm{e}}, \operatorname{pre}\mathcal{L}_{\mathrm{IO}})$, pick an arbitrary string $\hat{s} \in \operatorname{pre}\mathcal{V}_t$ and $\sigma \in \Sigma_{\mathrm{uc}}$, such that $\hat{s}\sigma \in \operatorname{pre}\mathcal{L}_{\mathrm{IO}}$. Two different cases can be differed. First, $\hat{s}\sigma \in \operatorname{pre}\mathcal{L}_{\|}$. Hence, $\hat{s} \in \operatorname{pre}\mathcal{V}_s$. Denote j the index of the corresponding component, i.e. $\sigma \in \Sigma_j$. Controllability of $\operatorname{pre}\mathcal{V}_{r,j}$ w.r.t. $\operatorname{pre}\mathcal{L}_j$ implies $(\mathrm{p}_{\mathrm{j}}\hat{s})\sigma \in \operatorname{pre}\mathcal{V}_{r,j}$. Thus, it is $\mathrm{p}_{\mathrm{n}}(\hat{s}\sigma) \in \operatorname{pre}\mathcal{V}_{r,n}$ for all n and $\hat{s}\sigma \in \operatorname{pre}\mathcal{L}_{\mathrm{seq}}$. This implies $\hat{s}\sigma \in \operatorname{pre}\mathcal{V}_s \subseteq \operatorname{pre}\mathcal{V}_t$. Second, $\hat{s}\sigma \in \operatorname{pre}\mathcal{L}_{\mathrm{err}}$. Since $\operatorname{pre}\mathcal{L}_{\mathrm{err}}/\hat{s} = \operatorname{pre}\mathcal{V}_{s,Err}/\hat{s}$, $\hat{s}\sigma \in \operatorname{pre}\mathcal{V}_{s,Err} \subseteq \operatorname{pre}\mathcal{V}_t$. Obviously, the candidate $\mathcal{V}_t$ is closed, hence also relatively closed w.r.t. $\operatorname{clo}\mathcal{L}_{\mathrm{IO}}$. □

The composition is continued by adding the *IO-environment*; see Figure 4.11.

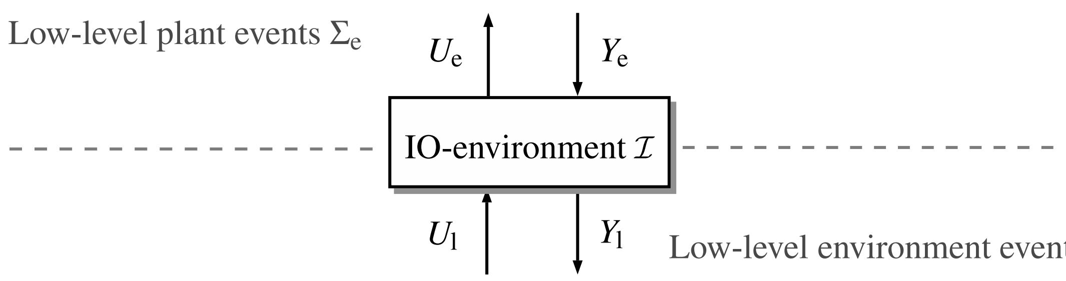

Low-level plant events Σ_{e}
U_{e}
Y_{e}
IO-environment $\mathcal{I}$
U_{l}
Y_{l}
Low-level environment events Σ_{l}

Figure 4.11: IO-environment

It provides interaction interfaces for low-level plant events Σ_{e}, and *low-level environment events* $\Sigma_{\mathrm{l}} := Y_{\mathrm{l}} \dot{\cup} U_{\mathrm{l}}$ that propagate interfaces that result available after the composition with several IO-plants. In general, the *IO-environment* models dependencies between the individual plant components, like already demonstrated in Figure 4.10. By modelling the dependencies in separate models, modular controller can be designed locally without causing conflicting behaviour in the composed system. Further, it facilitates the redesign of the layout of the given composed system. Nevertheless, in order to avoid conflicting behaviour of the IO-environment with the respective IO-plants, it is necessary to require the following specific properties.

Definition 4.3.3 IO-environment

Given an alphabet $\Sigma_{\mathrm{el}} := \Sigma_{\mathrm{e}} \dot{\cup} \Sigma_{\mathrm{l}}$, with $\Sigma_{\mathrm{l}} = Y_{\mathrm{l}} \dot{\cup} U_{\mathrm{l}}$, the discrete event system $\mathcal{I} \subseteq \Sigma_{\mathrm{el}}^{\omega}$ is an *IO-environment*, if

I1 $\mathcal{I} \subseteq ((Y_{\mathrm{e}}U_{\mathrm{e}})^*(Y_{\mathrm{e}}Y_{\mathrm{l}}U_{\mathrm{l}}U_{\mathrm{e}})^*)^{\omega}$.

I2 $\mathrm{pre}\mathcal{I}$ possesses locally free inputs U_{l} and Y_{e}.

I3 $\mathcal{I}$ is topologically closed. □

Regarding I1, certain similarities between the event ordering of the IO-environment and the IO-controller introduced in Section 3.2 can be noticed. Like the high-level control events, the low-level environment events represent an interface that results available in the propagated part of the composed system used for the next composition level. Properties I2 and I3 are logical consequences in order to guarantee non-conflicting behaviour and locally free inputs of the composed system.

In order to compose the IO-shuffle $\mathcal{L}_{\mathrm{IO}}$ with an IO-environment, both languages are raised to the *modular IO-system alphabet* $\bar{\Sigma} := \Sigma_{\mathrm{p}} \dot{\cup} \Sigma_{\mathrm{e}} \dot{\cup} \Sigma_{\mathrm{l}}$, to consider the *full IO-shuffle behaviour*.

Definition 4.3.4 Full IO-shuffle behaviour

Given an IO-shuffle $\mathcal{L}_{\mathrm{IO}} \subseteq \Sigma_{\mathrm{pe}}^{\omega}$. The *full IO-shuffle behaviour* is given by

$$\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} := \left(\mathrm{p}_{\mathrm{pe}}^{-\omega}(\mathcal{L}_{\mathrm{IO}} \cup \mathrm{pre}\,\mathcal{L}_{\mathrm{IO}})\right) \cap \left(\mathrm{clo}((Y_{\mathrm{p}}U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*(Y_{\mathrm{e}}Y_{\mathrm{l}}U_{\mathrm{l}}U_{\mathrm{e}})^*)^{\omega}\right) \quad \square$$

Similar to Proposition 4.2.1, and the inverse projection of the IO-plant to the overall alphabet, IO-plant property P3 is propagated again in the inverse projected IO-shuffle.

Proposition 4.3.2 (Baier and Moor, 2015)

For non-anticipating IO-plant components $\mathcal{L}_n \subseteq \Sigma_n^{\omega}$, $n = 1, \ldots, m$, the full IO-shuffle $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \subseteq \bar{\Sigma}^{\omega}$ is ω-controllable w.r.t. $(U_{\mathrm{p}} \dot{\cup} U_{\mathrm{l}}, \mathrm{clo}\,\mathcal{L}_{\mathrm{IO},\bar{\Sigma}})$. □

In addition, the *full IO-environment behaviour* is also raised to the overall alphabet by a similar construction.

Definition 4.3.5 Full IO-enviroment behaviour

Given an IO-environment $\mathcal{I} \subseteq \Sigma_{\mathrm{el}}^{\omega}$. The *full IO-environment behaviour* is given by

$$\mathcal{I}_{\bar{\Sigma}} := \ (\mathrm{p}_{\mathrm{el}}^{-\omega}(\mathcal{I} \cup \mathrm{pre}\mathcal{I})) \ \cap \ (\mathrm{clo}((Y_{\mathrm{p}}U_{\mathrm{p}})^{*}(Y_{\mathrm{e}}U_{\mathrm{e}})^{*}(Y_{\mathrm{e}}Y_{\mathrm{l}}U_{\mathrm{l}}U_{\mathrm{e}})^{*})^{\omega}) \qquad \square$$

Note that the full IO-environment is topologically closed due to property I3.

The final composition procedure of full IO-environment and full IO-shuffle behaviour is analogue to the closed-loop construction in Section 3.2. The composed system is received by a simple intersection of both behaviours.

Definition 4.3.6 Modular IO-system

Given an IO-shuffle $\mathcal{L}_{\mathrm{IO}} \subseteq \Sigma_{\mathrm{pe}}^{\omega}$ and an IO-environment $\mathcal{I} \subseteq \Sigma_{\mathrm{el}}^{\omega}$. The *modular IO-system* is given by $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}} \subseteq \bar{\Sigma}^{\omega}$. $\square$

In order to analyse relevant properties of the resulting behaviour of the modular IO-system the focus lies on verifying non-conflicting behaviour of $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}}$ and $\mathcal{I}_{\bar{\Sigma}}$ and the propagation of IO-plant property P3.

Proposition 4.3.3

For non-anticipating IO-plant components $\mathcal{L}_n \subseteq \Sigma_n^{\omega}$, $n = 1,\dots,m$ and an IO-environment $\mathcal{I} \subseteq \Sigma_{\mathrm{el}}^{\omega}$, consider the full behaviours $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \subseteq \bar{\Sigma}^{\omega}$ and $\mathcal{I}_{\bar{\Sigma}} \subseteq \bar{\Sigma}^{\omega}$, respectively. Then, $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}}$ and $\mathcal{I}_{\bar{\Sigma}}$ are non-conflicting. Moreover, the modular IO-system $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}}$ is ω-controllable w.r.t. $(U_{\mathrm{p}} \dot{\cup} U_{\mathrm{l}}, \mathrm{clo}(\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}}))$.

Proof. The claim is verified by the same line of thought as in the proof of Proposition 4.2.2. Given non-anticipating IO-plant components, consider the full IO-shuffle and the full environment behaviour.

$$\begin{aligned}
\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} &= \ (\mathrm{p}_{\mathrm{pe}}^{-\omega}(\mathcal{L}_{\mathrm{IO}} \cup \mathrm{pre}\,\mathcal{L}_{\mathrm{IO}})) \ \cap \ (\mathrm{clo}((Y_{\mathrm{p}}U_{\mathrm{p}})^{*}(Y_{\mathrm{e}}U_{\mathrm{e}})^{*}(Y_{\mathrm{e}}Y_{\mathrm{l}}U_{\mathrm{l}}U_{\mathrm{e}})^{*})^{\omega}), \\
\mathcal{I}_{\bar{\Sigma}} &= \ (\mathrm{p}_{\mathrm{el}}^{-\omega}(\mathcal{I} \cup \mathrm{pre}\mathcal{I})) \ \cap \ (\mathrm{clo}((Y_{\mathrm{p}}U_{\mathrm{p}})^{*}(Y_{\mathrm{e}}U_{\mathrm{e}})^{*}(Y_{\mathrm{e}}Y_{\mathrm{l}}U_{\mathrm{l}}U_{\mathrm{e}})^{*})^{\omega}),
\end{aligned}$$

respectively. It has to be shown that $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}}$ and $\mathcal{I}_{\bar{\Sigma}}$ are non-conflicting and that $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}}$ is ω-controllable w.r.t. $(U_{\mathrm{p}} \dot{\cup} U_{\mathrm{l}}, \mathrm{clo}(\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}}))$. Beginning with ω-controllability, a suitable candidate $\mathcal{V}_s \subseteq \mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}}$ for an arbitrarily chosen $s \in \mathrm{pre}(\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}})$ is constructed. Referring to Proposition 4.3.2, there exists $\mathcal{W}_s \subseteq \mathcal{L}_{\mathrm{IO},\bar{\Sigma}}$, such that $s \in \mathrm{pre}\,\mathcal{W}_s$, $\mathrm{pre}\,\mathcal{W}_s$ is controllable w.r.t. $(\Sigma_{\mathrm{l}} \dot{\cup} U_{\mathrm{p}} \dot{\cup} U_{\mathrm{e}}, \mathrm{pre}\,\mathcal{L}_{\mathrm{IO},\bar{\Sigma}})$, and $\mathcal{W}_s$ is relatively closed w.r.t. $\mathrm{clo}\,\mathcal{L}_{\mathrm{IO},\bar{\Sigma}}$. In particular, $\mathcal{W}_s$ is closed. To establish ω-controllability of $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}}$ w.r.t.

$\mathrm{clo}(\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}})$, consider the candidate $\mathcal{V}_s := \mathcal{W}_s \cap \mathcal{I}_{\bar{\Sigma}}$. Clearly, $\mathcal{V}_s \subseteq \mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}}$. Furthermore, $\mathcal{V}_s = \mathcal{W}_s \cap \mathcal{I}_{\bar{\Sigma}} = (\mathrm{clo}\,\mathcal{W}_s) \cap (\mathrm{clo}\,\mathcal{I}_{\bar{\Sigma}}) \supseteq \mathrm{clo}\,\mathcal{V}_s$, i.e., $\mathcal{V}_s$ is closed and, thus, relatively closed w.r.t. any superset. To show controllability of $\mathrm{pre}\,\mathcal{V}_s$ w.r.t. $\mathrm{pre}(\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}})$, pick $r \in \mathrm{pre}(\mathcal{W}_s \cap \mathcal{I}_{\bar{\Sigma}})$ and $\sigma \in U_\mathrm{l} \dot{\cup} U_\mathrm{p}$ such that $r\sigma \in \mathrm{pre}\,\mathrm{clo}(\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}}) \subseteq (\mathrm{pre}\,\mathcal{L}_{\mathrm{IO},\bar{\Sigma}}) \cap (\mathrm{pre}\,\mathcal{I}_{\bar{\Sigma}})$. By controllability of $\mathrm{pre}\,\mathcal{W}_s$, it follows that $r\sigma \in (\mathrm{pre}\,\mathcal{W}_s)$. The locally free input U_l of $\mathcal{I}$ and the inverse projection in the definition of $\mathcal{I}_{\bar{\Sigma}}$ imply $r\sigma \in (\mathrm{pre}\,\mathcal{I}_{\bar{\Sigma}})$ for either case $\sigma \in U_\mathrm{l}$ or $\sigma \in \dot{\cup} U_\mathrm{p}$, respectively. So far, it is $r\sigma \in \mathrm{pre}(\mathcal{W}_s) \cap (\mathrm{pre}\,\mathcal{I}_{\bar{\Sigma}})$. To establish $r\sigma \in \mathrm{pre}(\mathcal{W}_s \cap \mathcal{I}_{\bar{\Sigma}})$, observe that each event in Σ is either uncontrollable for $\mathrm{pre}\,\mathcal{W}_s$ or a locally free input for $\mathrm{pre}\,\mathcal{I}_{\bar{\Sigma}}$. Thus, starting with $r_0 = r\sigma$, an unbounded sequence $(r_n) \subseteq (\mathrm{pre}\,\mathcal{W}_s) \cap (\mathrm{pre}\,\mathcal{I}_{\bar{\Sigma}})$ with limit $w := \lim(r_n) \in (\mathrm{clo}\,\mathcal{W}_s) \cap (\mathrm{clo}\,\mathcal{I}_{\bar{\Sigma}}) = \mathcal{W}_s \cap \mathcal{I}_{\bar{\Sigma}}$ can be constructed. Hence, $r\sigma \in \mathrm{pre}(\mathcal{W}_s \cap \mathcal{I}_{\bar{\Sigma}})$. This concludes the proof of ω-admissibility. To show that the languages are non-conflicting, pick an arbitrary $s \in (\mathrm{pre}\,\mathcal{L}_{\mathrm{IO},\bar{\Sigma}}) \cap (\mathrm{pre}\,\mathcal{I}_{\bar{\Sigma}})$. In the argument, it is referred to the same candidate $\mathcal{W}_s$ as used in the first part of this proof. In particular, it is $s \in (\mathrm{pre}\,\mathcal{W}_s) \cap (\mathrm{pre}\,\mathcal{I}_{\bar{\Sigma}})$ and, as above, it is possible to start with $s_0 = s$ to construct an unbounded sequence $(s_n) \subseteq (\mathrm{pre}\,\mathcal{W}_s) \cap (\mathrm{pre}\,\mathcal{I}_{\bar{\Sigma}})$ by successively appending events that are either uncontrollable for $\mathrm{pre}\,\mathcal{W}_s$ or a locally free input for $\mathrm{pre}\,\mathcal{I}_{\bar{\Sigma}}$. Consequently, it follows that $w := \lim(s_n) \in (\mathrm{clo}\,\mathcal{W}_s) \cap (\mathrm{clo}\,\mathcal{I}_{\bar{\Sigma}}) = \mathcal{W}_s \cap \mathcal{I}_{\bar{\Sigma}} \subseteq \mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}}$ and thus $s \in \mathrm{pre}(\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}})$. □

Hence, non-conflicting behaviour of $\mathcal{L}_{\mathrm{IO},\bar{\Sigma}}$ and $\mathcal{I}_{\bar{\Sigma}}$ and the propagation of IO-plant property P3 are satisfied and can be further used to verify the propagation of the IO-plant properties P1 - P3 in the *external modular IO-system behaviour*, which shall be used as IO-plant on the next hierarchical level.

4.3.2 Propagation of plant properties

For computational reasons, hiding further information that is not necessary for the controller design of the next higher level is of interest. In particular, the focus lies on hiding events from Σ_e, to hide IO-plant interfaces that are already connected to other IO-plants in the modular IO-system. The *external IO-system behaviour* is denoted by $\mathcal{L}_{\mathrm{pl}} := \mathrm{p}^{\omega}_{\mathrm{pl}}(\mathcal{L}_{\mathrm{IO},\bar{\Sigma}} \cap \mathcal{I}_{\bar{\Sigma}})$; see also Figure 4.10. It is readily verified that $\mathcal{L}_{\mathrm{pl}}$ satisfies the IO-plant properties P1 - P3.

Theorem 4.3.4

Given a modular IO-system, consisting of the plant components $\mathcal{L}_n \subseteq \Sigma_n^{\omega}$, $n = 1, \ldots, m$, an IO-environment $\mathcal{I} \subseteq \Sigma_{\mathrm{el}}^{\omega}$, and the external IO-system behaviour $\mathcal{L}_{\mathrm{pl}}$. If $\mathcal{L}_n \subseteq \Sigma_n^{\omega}$, $n = 1, \ldots, m$ are non-anticipating IO-plants, then so is $\mathcal{L}_{\mathrm{pl}}$.

Proof. Regarding the event ordering P1, observe that the modular IO-system satisfies $\mathcal{L}_{\mathrm{pl}} \subseteq \mathrm{p}_{\mathrm{pl}}^{\omega}((Y_{\mathrm{p}}U_{\mathrm{p}})^*(Y_{\mathrm{e}}U_{\mathrm{e}})^*(Y_{\mathrm{e}}Y_{\mathrm{l}}U_{\mathrm{l}}U_{\mathrm{e}})^*)^{\omega} = ((Y_{\mathrm{p}}U_{\mathrm{p}})^*(Y_{\mathrm{l}}U_{\mathrm{l}})^*)^{\omega}$. Regarding locally free inputs P2, it is first shown that $\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}}$ possesses locally free inputs U_{p} and U_{l}. Pick $s, s' \in \mathrm{pre}(\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}})$ and $\mu, \mu' \in U_{\mathrm{p}}$, as well as $\nu, \nu' \in U_{\mathrm{l}}$ such that $s\mu, s'\nu \in \mathrm{pre}(\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}})$. By the locally free inputs U_{p} and U_{l} of $\mathrm{pre}\,\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}}$ and $\mathrm{pre}\,\mathcal{I}_{\tilde{\Sigma}}$ $s\mu', s'\nu' \in (\mathrm{pre}\,\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}}) \cap (\mathrm{pre}\,\mathcal{I}_{\tilde{\Sigma}})$ is obtained, and, referring to non-conflictingness from Proposition 4.3.3, $s\mu', s'\nu' \in \mathrm{pre}(\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathrm{pre}\,\mathcal{I}_{\tilde{\Sigma}})$. Thus, $\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}}$ indeed possesses locally free inputs U_{p} and U_{l}, which are retained under projection to Σ_{pl}. It is left to verify non-anticipation P3. Pick any $s \in \mathrm{pre}\,\mathcal{L}_{\mathrm{pl}} = \mathrm{pre}\,\mathrm{p}_{\mathrm{pl}}^{\omega}(\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}})$. Then, there exists $t \in \mathrm{pre}(\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}})$ such that $\mathrm{p}_{\mathrm{pl}}t = s$. Recall from Proposition 4.3.3 that $\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}}$ is ω-admissible w.r.t. $(U_{\mathrm{p}} \dot{\cup} U_{\mathrm{l}}, \mathrm{clo}(\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}}))$. Thus, $\mathcal{W}_t \subseteq \mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}}$ can be chosen such that $t \in \mathrm{pre}\,\mathcal{W}_t$, and $\mathrm{pre}\,\mathcal{W}_t$ is controllable w.r.t. $(U_{\mathrm{p}} \dot{\cup} U_{\mathrm{l}}, \mathrm{pre}\,\mathrm{clo}(\mathcal{L}_{\mathrm{IO},\tilde{\Sigma}} \cap \mathcal{I}_{\tilde{\Sigma}}))$, and $\mathcal{W}_t$ is closed. As a candidate to establish P3, let $\mathcal{V}_s := \mathrm{p}_{\mathrm{pl}}^{\omega}\mathcal{W}_t \subseteq \mathrm{p}_{\mathrm{pl}}^{\omega}\mathcal{L}_{\mathrm{pl}}$ and observe $s = \mathrm{p}_{\mathrm{pl}}t \in \mathrm{p}_{\mathrm{pl}}\,\mathrm{pre}\,\mathcal{W}_t = \mathrm{pre}\,\mathcal{V}_s$. Controllability and closedness of $\mathcal{V}_s$ follow as in the proof of Theorem 4.2.3, and it follows that $\mathcal{L}_{\mathrm{pl}}$ is ω-admissible w.r.t. $(U_{\mathrm{p}} \dot{\cup} U_{\mathrm{l}}, \mathrm{clo}\,\mathcal{L}_{\mathrm{pl}})$. □

This concludes the procedure regarding modular controller design. It has been proven that the IO-plant properties P1 - P3 are retained under the proposed modular IO-system composition. A direct consequence is the possibility to design controller in a modular fashion, to compose modular controlled IO-plants together with the respective IO-environment to a modular IO-system, as given in Figure 4.9, and to use the resulting external IO-system behaviour as new IO-plant for the subsequent controller design. Theorems 4.2.3 and 4.3.4 together verify that it is possible to apply a hierarchical and modular controller design for large-scale systems modelled by ω-languages.

4.4 Example - Application on transport systems

The proposed approach is demonstrated in the context of the controller design for a large-scale transport system; see Figure 4.12.

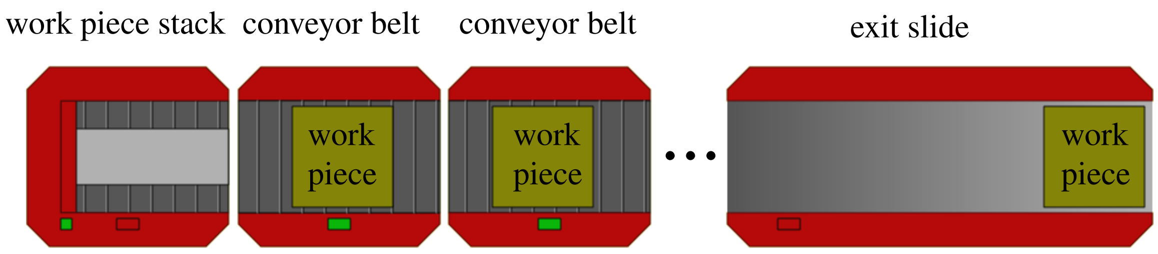


Figure 4.12: Transport system example

It consists of several conveyor belts, as described in Section 3.2, which are arranged next to each other. A work piece stack delivers work pieces from the left and an exit slide receives work pieces from the left. The purpose of this transport system, is the transport of work pieces from left to right. However, to keep it simple, the focus of this example lies on the controller design of the conveyor belt chain, considering an increasing number of conveyor belts and an increasing number of work pieces in the transport system. The example demonstrates the use of IO-plant models for the conveyor belts. Further, IO-environment models are introduced. In addition, the use of IO-specifications as abstractions, and the efficient hierarchical abstraction-based and modular controller design are discussed. Again, Büchi automata are applied for the representation of the respective ω-languages and the software library (libFAUDES, 2015) is used to compute respective solutions.

4.4.1 Low-level controller design

On the low-level, a number of conveyor belts is given. The set-up of the conveyor belts is similar to the one described already in Section 3.2, including respective events and their properties.

Remark 4.4.1

Owing to the limitation of the libFAUDES to provide algorithms only for Büchi automata, it is necessary to simplify the models from Section 3.2. In the hierarchical context, projections of ω-languages have to be computed in order to reduce systems to their external behaviour, for example in the context of the modular IO-system. However, the computation of projections leads, in general, to non-deterministic automata. After determinisation, as for example in (Mukund, 1996), the result is not necessarily a Büchi automaton, but an automaton with a stronger acceptance condition, for example a Rabin automaton. Proceeding with algorithms of the libFAUDES is not possible and another software library would be necessary. However, to the author's best knowledge a software library that includes as well Rabin automata, as algorithms for supervisor synthesis is not available. As a work around, the example has been elaborated such that only ω-closed behaviours are involved in projection constructions or behaviours where it is possible to reconstruct eventuality properties after the projection. □

Figure 4.13 shows the physical ad hoc model of the conveyor belt. The only difference to the model in Figure 3.14 is the missing immediate event `enter` after a work piece has exited the belt. It is assumed that the `receipt` event is always generated before another work piece enters the belt. This fact is guaranteed by the additional requirement to stop a work piece at the present conveyor belt sensor as long as the next conveyor belt is still occupied.

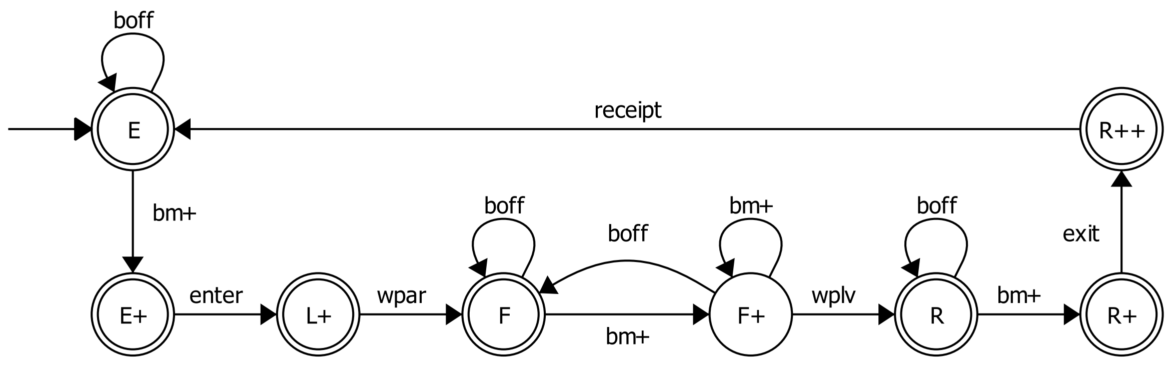


Figure 4.13: Physical ad hoc model of the conveyor belt

As demonstrated in Section 3.2, the physical model can be transformed into an IO-plant; see Figure 4.14.

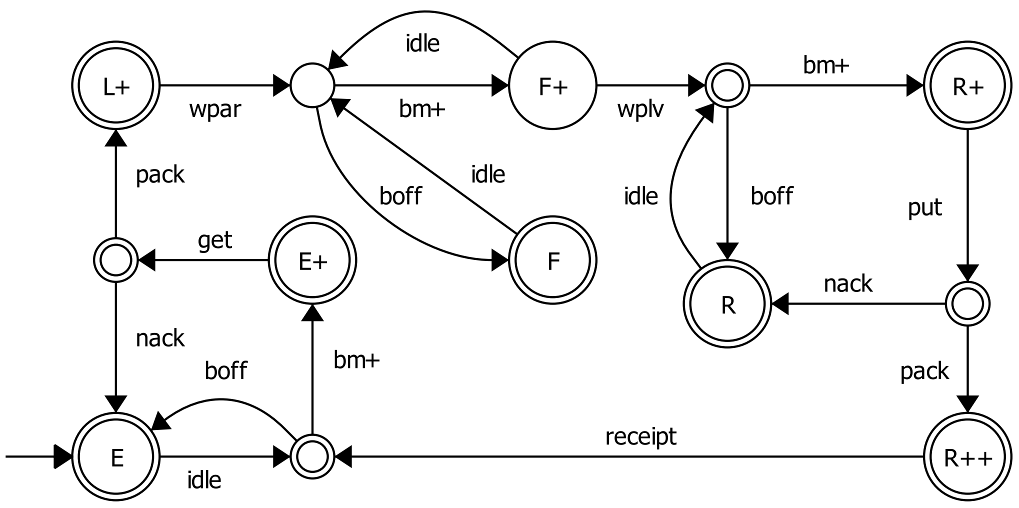


Figure 4.14: Conveyor belt, IO-plant model

In Table 4.1, the IO-plant events are categorized into respective alphabets. IO-plant property P1, the event ordering, and IO-plant property P2, locally free inputs, are readily verified as before in Section 3.2. To verify IO-plant property P3, the property of non-anticipation, the supremal ω-controllable sub-language of the conveyor belt behaviour w.r.t. its topological closure is computed. Thereby, P3 is verified and the model in Figure 4.14 is an IO-plant.

$Y_\mathrm{e} = \{\mathtt{get}, \mathtt{put}\}$	get/put a work piece from/to the environment
$U_\mathrm{e} = \{\mathtt{pack}, \mathtt{nack}\}$	acknowledgement of recent `get`/`put`
$U_\mathrm{p} = \{\mathtt{bm+}, \mathtt{boff}\}$	plant actuator to operate belt motor
$Y_\mathrm{p} = \{\mathtt{idle}, \mathtt{wpar}, \mathtt{wplv}, \mathtt{receipt}\}$	plant sensors with dummy `idle` if nothing else is to report

Table 4.1: List of events of the conveyor belt IO-plant model

The control objective is basically the same as in Section 3.2, with the difference that more than one conveyor belt is involved. Hence, to the operator output events `full` and `empty` another output event `wpa` is added to indicate whether a work piece is available, while it is still possible to receive (`wpr`) or to deliver (`wpd`) another one. In the specification, the intended semantics of the introduced events is defined by relating them to the environment events; see Figure 4.15 and Table 4.2.

$Y_\mathrm{c} = \{\mathtt{empty}, \mathtt{full}, \mathtt{wpa}\}$	operator feedback to indicate presence of a work piece
$U_\mathrm{c} = \{\mathtt{wait}, \mathtt{wpd}, \mathtt{wpr}\}$	operator event to wait, or to receive/deliver a work piece
$Y_\mathrm{e} = \{\mathtt{get}, \mathtt{put}\}$	get/put a work piece from/to the environment
$U_\mathrm{e} = \{\mathtt{pack}, \mathtt{nack}\}$	acknowledgement of recent `get`/`put`

Table 4.2: List of events of the IO-specification

The specification automaton exclusively refers to the alphabet Σ_ce and it is left to the synthesis procedure to figure out how to drive the plant by interleaving events from Σ_p. Technically, the depicted automaton realises the projection $\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{E}$ of the formal specification $\mathcal{E}$. In particular, there is no need to compute $\mathrm{p}^{\omega}_{\mathrm{ce}}\mathcal{E}$ from $\mathcal{E}$, avoiding a potentially exponential growth in the state count.

Although, the specification automaton exclusively refers to the alphabet Σ_ce, it is still possible to further add requirements regarding the internal plant events, constructing a subset of the original specification. Here, two additional requirements are considered, which are represented in Figure 4.16.

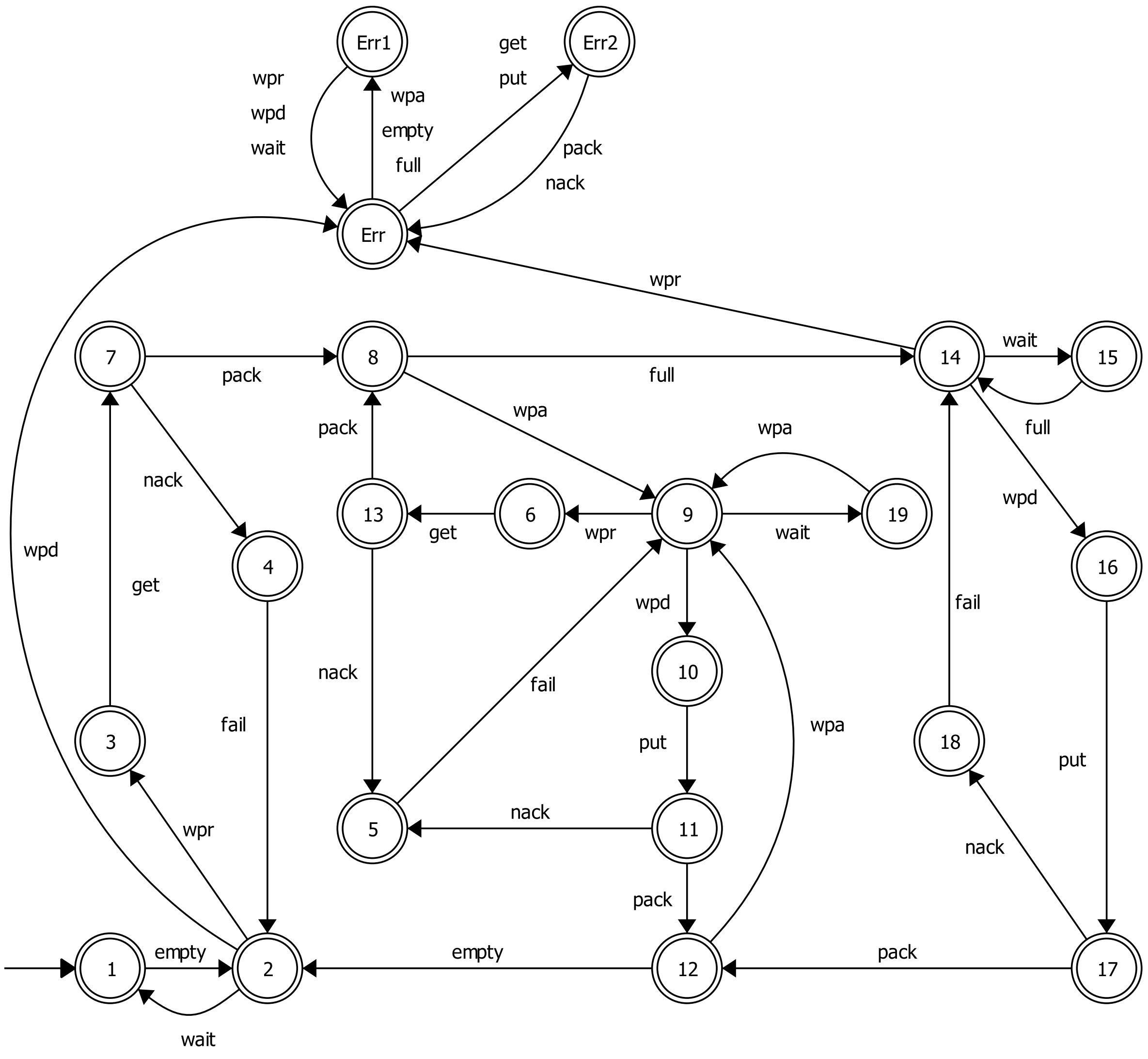
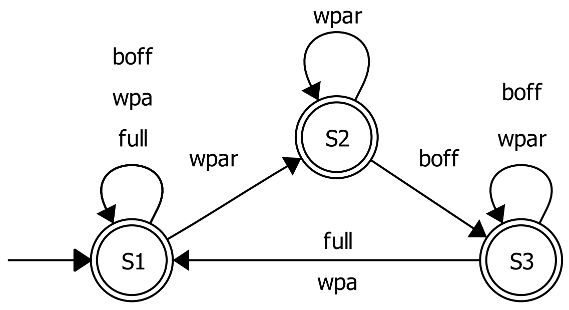


Figure 4.15: Specification for arbitrary number of conveyor belts

As mentioned before, the specifications request the conveyor belt to stop after the arrival of a work piece (`wpar`) and before sending a feedback to the operator. Further it is required to stop in the case that the operator has sent a `wait` command. Finally, the specification properties E1 and E2 are verified as in Section 3.2.



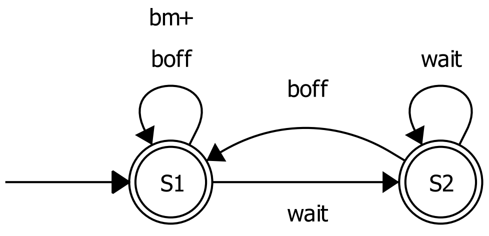


Figure 4.16: Additional specifications regarding plant events

The solution of the given IO-control-problem, yields to a non-empty closed loop $\mathcal{K}$ that satisfies closed loop properties K1–K3 and is further relatively closed w.r.t. the plant behaviour. Thus, a controller that satisfies properties H1 and H2 is obtained by projection of clo$\mathcal{K}$ to Σ_{cp}. The resulting controller amounts to 31 states. The conveyor belts in the transport system are assumed to be identical. Hence it is possible to use the synthesised controller for each belt as low-level controller. In order to formally end up with disjoint alphabets, the convention to prefix each event with an identifier of the respective component is used; e.g., `cb1_boff` to turn off the motor of the first conveyor belt `cb1`, counting from the left to the right.

4.4.2 Hierarchical and modular controller design

The control objective on the high level is the coordination of a group of conveyor belts to behave in the same way as a single conveyor belt. Hence, the specification on each level is equal to the specification on the low-level, given in Figure 4.15.

In Figure 4.17, the hierarchical control architecture that is build-up is shown for a chain of four conveyor belts. For each single conveyor belt a low-level controller ($\mathcal{H}_1 - \mathcal{H}_4$) is provided. Aside from the event labelling, they are identical. Next, the two leftmost conveyor belts are coordinated by the controller $\mathcal{H}_{12}$ one level above. Step by step, another conveyor belt is added from left to right by adding another controller for the component considered before and the additional locally controlled conveyor belt. As IO-plant behaviour of each component, the specification of the closed loop one level below is used, applying thereby an abstraction-based controller design. Since the behaviour of the IO-plants and the IO-specification are equal in each level, controller design has to be performed just once and the resulting controller can be applied for each level, just by relabelling respective events. Hence, the controller $\mathcal{H}_{12}, \mathcal{H}_{1-3}$ and $\mathcal{H}_{1-4}$ are structurally equal, aside from the particular event relabelling.

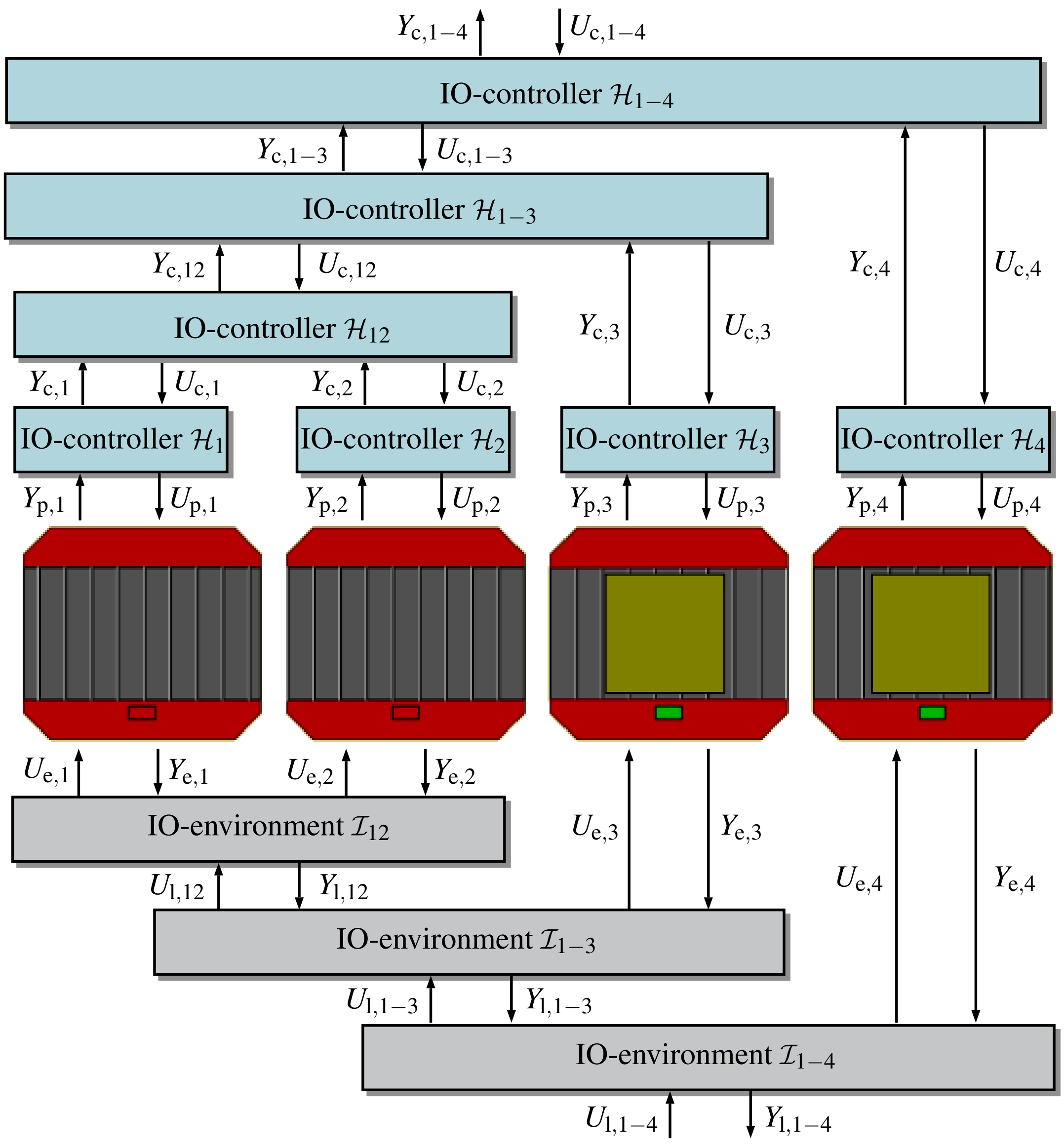


Figure 4.17: Hierarchical control architectures for the conveyor belts cb1, cb2, cb3 and cb4

In order to combine two locally controlled components to a modular IO-system, for example the two left most conveyor belts cb1 and cb2, an IO-environment is used, as described in Section 4.3. The respective automaton is given in Figure 4.18 and the respective alphabets in Table 4.3. Owing to the physical arrangement of the conveyor belts, the departure of a work piece from belt cb1 corresponds to the arrival of the work piece at belt cb2. This correspondence is modelled in the IO-environment by a direct sequence of cb1_put–cb1_pack–cb2_get–cb2_pack.

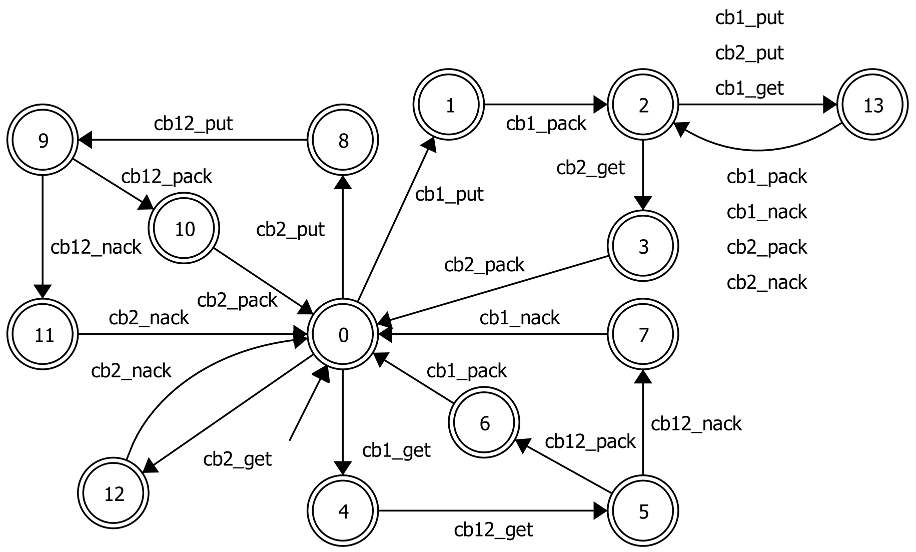


Figure 4.18: Environment for the two conveyor belts `cb1` and `cb2`

$Y_l = \{$`cb12_get`, `cb12_put`$\}$	get/put a work piece from/to the environment
$U_l = \{$`cb12_pack`, `cb12_nack`$\}$	acknowledgement of recent `cb12_get`/`cb12_put`
$Y_{e,1} = \{$`cb1_get`, `cb1_put`$\}$	get/put a work piece from/to the environment
$U_{e,1} = \{$`cb1_pack`, `cb1_nack`$\}$	acknowledgement of recent `cb1_get`/`cb1_put`
$Y_{e,2} = \{$`cb2_get`, `cb2_put`$\}$	get/put a work piece from/to the environment
$U_{e,2} = \{$`cb2_pack`, `cb2_nack`$\}$	acknowledgement of recent `cb2_get`/`cb2_put`

Table 4.3: List of events of the IO-environment

For the purpose of further compositions at subsequent stages of the hierarchical design, the proposed environment also introduces events to represent work pieces entering or leaving the group of the two conveyor belts. These additional events are prefixed by `cb12_`. The IO-environment properties I1, the event ordering, and I2, locally free inputs, are verified by the same inspection as P1 and P2. IO-environment property I3, the topological closedness, is an immediate consequence of the marking of all states.

In the next step, the closed-loop behaviour of the two belts `cb1` and `cb2` under low-level control can be composed with the environment to an IO-system as discussed in Section 4.3. Referring to Theorem 4.2.3, the individual closed loops satisfy the IO-plant properties P1–P3, and, by Theorem 4.3.4, so does the composed system including the environment. In particular, the design of a high-level controller that coordinates the two conveyor belts w.r.t. their effect on the environment can be based on an abstraction. The latter can be constructed by replacing the low-level closed-loop behaviours with their respective specifications prior to the composition with the environment. The resulting automaton counts 679 states.

Remark 4.4.2

In order to receive the external behaviour of the IO-composition, the projection has to be applied. In general, this operation results into a non-deterministic Büchi automaton and a determinisation procedure has to be applied to receive a deterministic structure. However, as mentioned in Remark 4.4.1, the resulting automaton is in general not a Büchi automaton and cannot be converted into a Büchi automaton. In order to be able to compute the whole example using the libFAUDES, ω-closed specifications have been developed, in order to have the possibility to reproduce liveness properties in the composed system after the projection. It is proposed to build the composition on basis of the closed behaviours, to apply the projection and to intersect afterwards with a behaviour that considers the alternation of the involved plants on the infinite-time axis.

Given the modular IO-system, and the specification in Figure 4.15, a controller is synthesised with 2984 states. Thus, for the control of the two conveyor belts, two low-level controllers with 31 states each and one high-level controller with 2984 states are used. For an implementation of the controllers, there is no need to apply a parallel composition to them. Thus, the overall state count that is relevant for application is $2 \times 31 + (2-1) \times 2984$. Extending this approach to an arbitrary number of conveyor belts n with n hierarchical levels, the overall state count is $n \times 31 + (n-1) \times 2984$; see Table 4.4.

number of conv. belts	1	2	3	4	5	6	7	8
hierarchical design	31	3046	6061	9076	12091	15106	18121	21136
monolithic design	16	147	1454	13892	132162	1376371	–	–

Table 4.4: Controller state count for monolithic and hierarchical controller design

For the controller state count of the hierarchical controller design based on input/output systems, it is observed a favourable linear growth in the number of conveyor belts. Hence, the proposed approach compares well with a monolithic controller design with an expected state count exponential in the number of conveyor belts.

To actually compare the hierarchical approach to a monolithic control problem, the plant model in Figure 4.13 is used. The correspondence between the events `exit` and `enter` of two neighbouring conveyor belts is modelled by an additional automaton that prohibits any other plant event during the work piece exchange. The overall model is obtained by parallel composition and a monolithic controller is computed. For the controller design, the specification model in Figure 4.19 is used and adapted to the respective number of considered conveyor belts. In Figure 4.19 it is given for two conveyor belts.

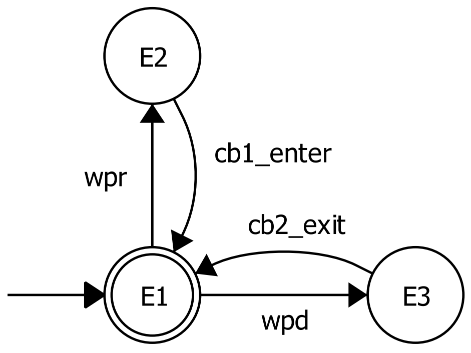


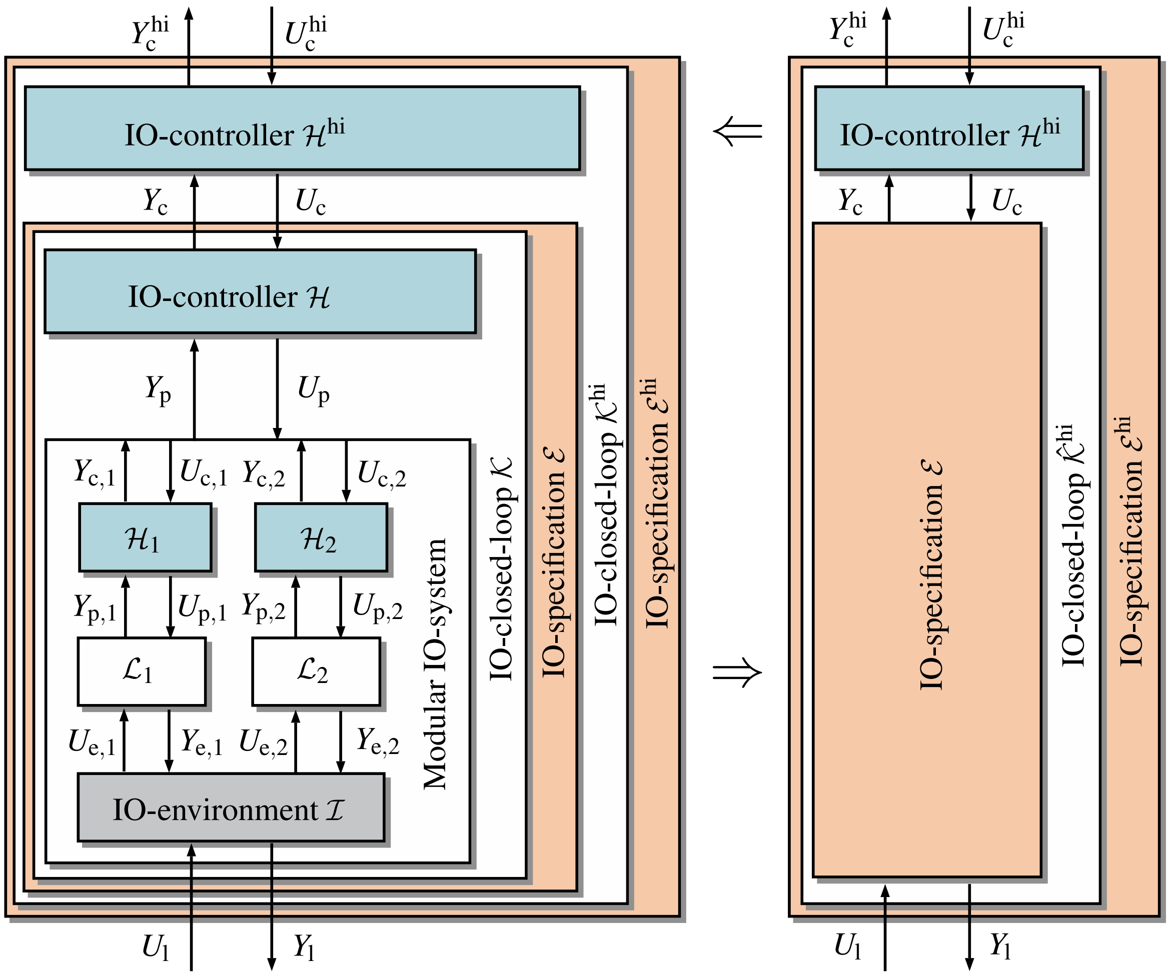
Figure 4.19: Specification for monolithic control problem for two belts `cb1` and `cb2`

The respective state counts for up to six conveyor belts are given in Table 4.4. The computation of the results for more than six conveyor belts already exceeded the memory capacities of a standard computer with a user memory of 16 GB. Nevertheless, it is readily observed that for more than three conveyor belts, the hierarchical design already provides a significantly smaller controller state count.

All the results have been validated and the expected closed loop behaviour have been confirmed in simulation experiments. The animated simulator FlexFact (libFAUDES, 2015) has been used to simulate the continuous-time physical behaviour, including digital signals for the purpose of controller interconnection provided via a network interface. The hierarchy of controllers was interpreted by the discrete-event simulator that is also provided by the software library libFAUDES. Here, events have been defined as edges on the digital signals accessible via the network interface of the plant simulation.

4.5 Conclusion - Hierarchical and modular controller design

In this Chapter, a general introduction into the idea behind hierarchical and modular control has been given and different procedures for $*$-language control problems have been discussed. Basic ideas of these procedures have been adopted to develop an approach for the design of abstraction-based hierarchical and modular control architectures for ω-languages given in form of input/output systems. It is obvious from the transport system example, that there exist various possibilities to build up the control architecture. Figure 4.20 demonstrates just one possible procedure.



Y_c^{hi}
U_c^{hi}
IO-controller $\mathcal{H}^{hi}$
Y_c
U_c
IO-controller $\mathcal{H}$
Y_p
U_p
$Y_{c,1}$
$U_{c,1}$
$Y_{c,2}$
$U_{c,2}$
$\mathcal{H}_1$
$\mathcal{H}_2$
$Y_{p,1}$
$U_{p,1}$
$Y_{p,2}$
$U_{p,2}$
$\mathcal{L}_1$
$\mathcal{L}_2$
$U_{e,1}$
$Y_{e,1}$
$U_{e,2}$
$Y_{e,2}$
IO-environment $\mathcal{I}$
Modular IO-system
IO-closed-loop $\mathcal{K}$
IO-specification $\mathcal{E}$
IO-closed-loop $\mathcal{K}^{hi}$
IO-specification $\mathcal{E}^{hi}$
U_l
Y_l
$\Leftarrow$
$\Rightarrow$
Y_c^{hi}
U_c^{hi}
IO-controller $\mathcal{H}^{hi}$
Y_c
U_c
IO-specification $\mathcal{E}$
IO-closed-loop $\tilde{\mathcal{K}}^{hi}$
IO-specification $\mathcal{E}^{hi}$
U_l
Y_l

Figure 4.20: Proposal for the design of a hierarchical and modular control architecture

Here, the control hierarchy is structured into three control levels. In a first step, two local modular controller are designed for two IO-plants. Next, the respective specifications are used as abstractions of the local closed loops. They are composed together with an IO-environment to a modular IO-system. Its external behaviour is used for the controller design on level two. For the controller design on level three, abstraction-based design is applied again and the specification of the closed-loop in level two is used as abstraction.

A similar set-up has been applied in the example in Subsection 4.4. The results therein justify the approach to alternate in an arbitrary way controller synthesis, closed-loop composition, abstraction and component composition. Compared to a monolithic approach, it has been demonstrated that the procedure is clearly more efficient, since a linear growth of the computational effort in the number of components is achieved, in contrast to the exponential growth in the monolithic context.

5 Summary

Since Ramadge and Wonham introduced for the first time their perspective of supervisory control of discrete event systems in (Ramadge and Wonham, 1987b), several approaches have been developed to handle its application in large-scale systems; see e.g. (Schmidt et al., 2008; Feng and Wonham, 2008; Leduc et al., 2005; Perk et al., 2008). The approaches differ regarding the particular modelling framework, required structural conditions or the type of model abstraction that is applied. Nevertheless, they share the fact that they are limited to modelling system behaviour with $*$-languages on the finite-time horizon. However, in the context of supervisory control of ω-languages, describing behaviour on the infinite-time axis, hierarchical and modular control approaches that enable an efficient controller design are still missing. Motivated thereby, the contribution of this thesis is an approach to efficiently design hierarchical and modular control architectures for ω-languages. To the author's best knowledge, it is the first approach, that deals with modelling discrete event systems by ω-languages that includes also general liveness properties, as well as with dividing the control task into sub-tasks in order to build an abstraction-based hierarchical and modular control architecture.

In Chapter 2 the particular modelling framework using discrete event systems and formal languages is introduced. A supervisory control problem for finite-time behaviours under partial event observation is set-up and a solution procedure is proposed. This particular control problem is further extended to include the property of completeness in the closed-loop system, in order to lead over to the control problem on the infinite-time horizon.

In Chapter 3 the control problem for ω-languages under complete and under partial observation is presented. There exists a strong connection to the control problem for finite-time behaviours, which is utilized to describe solutions to the control problem for ω-languages. Following the considerations in (Thistle and Wonham, 1992), an algorithm is proposed for the computation of solutions to the control problem under complete observation, described by the propositional μ-calculus. Based on this algorithm, a heuristic algorithm for the solution of the control problem under partial event observation is described. Both algorithm are implemented in the software library (libFAUDES, 2015). In the second part of this chapter, the basic control problem for the proposed hierarchical control architecture is presented. It is based on the description by input/output systems derived from the Behavioural Systems Theory according to (Willems, 1991). The input/output framework has already been used for hierarchical controller design in (Perk et al., 2008), involving only topologically closed behaviours. In this thesis it is further elaborated towards the inclusion of more general liveness properties, enabling further application areas.

In Chapter 4 basic concepts of abstraction-based hierarchical and modular control are illustrated and applied to the control problem in the input/output framework. Particular plant properties are introduced that are necessary for the application of abstraction-based design. Further, step-by-step a composition procedure is proposed for locally controlled systems, that shall be coordinated by high-level controller. It is shown, that the properties of the given framework enable a systematic construction of a hierarchical and modular control architecture. As in the case of hierarchical and modular approaches for $*$-languages, the respective computational effort can be reduced by the avoidance of large monolithic models and dividing the control task into sub-tasks enforced by sub-controllers. In contrast to most of the approaches for $*$-languages and in accordance with (Perk et al., 2008), abstraction procedures involving further computations are not required to build up the hierarchical control architecture. The abstractions are already given by the specification of the underlying control problems. As a consequence, the individual control levels can be synthesised independently from each other, in arbitrary order and the computational effort can be reduced. Finally, a transport system example has been used to illustrate step-by-step how to build up such a control architecture for large-scale systems. The hierarchical controller design shows a linear growth in the number of system components. In contrast, it has been shown that a similar monolithic controller design requires highly more computational effort due to an exponential growth of the system states in the number of components. Hence, for a transport systems example, computational efficiency of the proposed approach has been proven.

The design of appropriate models is an important topic that is still open to future research and challenging for all synthesis approaches in this area, in particular for practical applications. Plant models are based on the physical system behaviour. Hence, it is promising to elaborate a systematic way to extract them from the physical phenomenon. Thereby, required plant properties like the IO-plant properties P1 - P3 are expected to support a systematic design of adequate models. Once given an appropriate representation of the uncontrolled behaviour, it can be reused easily for various control problems. The development of model libraries further simplifies the application. In order to design appropriate specification models it seems to be promising to further analyse the connection of the Supervisory Control Theory to the area of Temporal Logics. This area has been extensively used for the verification of system behaviour, in particular in the area of program verification. However, model discrepancies have been a limiting factor up to now, to further analyse similarities in both areas. However, with the presented approach, limitations could be resolved since ω-automata represent a common basis of both. Supervisory Control Theory could benefit from the experience of a well established formalism to define required system behaviour. Nevertheless, in order to apply the approach to industrial examples it is necessary to further elaborate appropriate software for the computation of solutions. Available software libraries are still restricted. The example presented in this thesis focused on Büchi-automata, which can be handled by the software tool libFAUDES. However, the presented approach is not limited to Büchi automata and can be applied to all kind of ω-automata. A respective software library, e.g. for Rabin automata, would enable further prospects regarding the application of the presented procedure.

Appendix

A Technical details

This section collects a number of technical details to support the core arguments.

Lemma A.1

Consider $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{o}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and $\mathcal{L}$, $\mathcal{K} \subseteq \Sigma^{\omega}$. If $\mathcal{K}$ is ω-controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathcal{L})$, then $\mathrm{pre}\,\mathcal{K}$ is controllable w.r.t. $(\Sigma_{\mathrm{uc}}, \mathrm{pre}\,\mathcal{L})$, and $\mathcal{L}$ and $\mathcal{K}$ are non-conflicting.

Proof. Regarding controllability, pick any $s \in \mathrm{pre}(\mathcal{K} \cap \mathcal{L})$ and $\sigma \in \Sigma_{\mathrm{uc}}$, such that $s\sigma \in \mathrm{pre}\,\mathcal{L}$. In particular, there can be chosen a $\mathcal{V}_s \subseteq \mathcal{L} \cap \mathcal{K}$, with $s \in \mathrm{pre}\,\mathcal{V}_s$, satisfying C1 and C2. Controllability C2 implies $s\sigma \in \mathrm{pre}\,\mathcal{V}_s \subseteq \mathrm{pre}(\mathcal{L} \cap \mathcal{K}) \subseteq \mathrm{pre}\,\mathcal{K}$. This concludes the proof of controllability for $\mathrm{pre}\,\mathcal{K}$. For non-conflictingness, pick an arbitrary $s \in (\mathrm{pre}\,\mathcal{K}) \cap (\mathrm{pre}\,\mathcal{L})$ and choose again $\mathcal{V}_s \subseteq \mathcal{L} \cap \mathcal{K}$, with $s \in \mathrm{pre}\,\mathcal{V}_s$, satisfying C1 and C2. Since $s \in \mathrm{pre}\,\mathcal{V}_s \subseteq \mathrm{pre}(\mathcal{K} \cap \mathcal{L})$, it follows that $s \in \mathrm{pre}(\mathcal{K} \cap \mathcal{L})$. This concludes the proof of non-conflictingness. □

Lemma A.2

Let $\Sigma_{\mathrm{uc}} \subseteq \Sigma$, $\Sigma_{\mathrm{o}} \subseteq \Sigma$, $\Sigma_{\mathrm{c}} \subseteq \Sigma_{\mathrm{o}}$, and consider $\mathcal{L} \subseteq \Sigma^{\omega}$ and $\mathcal{K} \subseteq \Sigma^{\omega}$. If $\mathcal{K}$ is ω-admissible w.r.t. $(\Sigma_{\mathrm{uc}}, \Sigma_{\mathrm{o}}, \mathcal{L})$, then $(\mathrm{pre}\,\mathcal{L}) \cap (\mathrm{pre}\,\mathcal{K})$ is prefix-normal w.r.t. $(\Sigma_{\mathrm{o}}, \mathrm{pre}\,\mathcal{L})$.

Proof. Pick an arbitrary string $s \in (\mathrm{p}_{\mathrm{o}}^{-1}\mathrm{p}_{\mathrm{o}}((\mathrm{pre}\,\mathcal{L}) \cap (\mathrm{pre}\,\mathcal{K}))) \cap \mathrm{pre}\,\mathcal{L}$. Then there exists $s' \in (\mathrm{pre}\,\mathcal{L}) \cap (\mathrm{pre}\,\mathcal{K})$ such that $\mathrm{p}_{\mathrm{o}}s' = \mathrm{p}_{\mathrm{o}}s$, and choose $\mathcal{V}_{s'} \subseteq \mathcal{L} \cap \mathcal{K}$, $s' \in \mathrm{pre}\,\mathcal{V}_{s'}$, satisfying C1 - C3. Here, prefix-normality C3 implies $s \in \mathrm{pre}\,\mathcal{V}_{s'}$. Together with $\mathrm{pre}\,\mathcal{V}_{s'} \subseteq \mathrm{pre}(\mathcal{L} \cap \mathcal{K}) \subseteq (\mathrm{pre}\,\mathcal{L}) \cap (\mathrm{pre}\,\mathcal{K})$, follows $s \in (\mathrm{pre}\,\mathcal{L}) \cap (\mathrm{pre}\,\mathcal{K})$. □

B Table of symbols

Important symbols used throughout this paper are summarized in the following tables.

General symbols

Symbol	Description	Page
Σ	alphabet	10
Σ_{c}	controllable events	21
Σ_{o}	observable events	21
$\Sigma_{\mathrm{uc}} = \Sigma - \Sigma_{\mathrm{c}}$	uncontrollable events	21
$\Sigma_{\mathrm{uo}} = \Sigma - \Sigma_{\mathrm{o}}$	unobservable events	21
$r,s,t \subseteq \Sigma^*$	finite-length strings	10
$u,v,w \subseteq \Sigma^\omega$	infinite-length strings	12
$L,H,K,E \subseteq \Sigma^*$	finite-time system behaviour, $*$-languages	10
$\mathcal{L},\mathcal{H},\mathcal{K},\mathcal{E},\mathcal{V} \subseteq \Sigma^\omega$	infinite-time system behaviour, ω-languages	12
f	supervisor	21
Γ	control pattern	21
$(\mathrm{pre}\,L)_f,(\mathrm{pre}\,\mathcal{L})_f$	local closed-loop behaviour	23
L_f	finite-time closed-loop behaviour	23
$\mathcal{L}_f$	infinite-time closed-loop behaviour	31

Operators

Symbol	Description	Page
pre	prefix closure operator for $*$-languages/ω-languages	10/12
$/$	quotient operator	13
elig	active event set operator	13
lim	limit operator for $*$-languages	13
clo	topological closure operator for ω-languages	13
sup	supremum operator	25
p_-	projection from Σ^* to Σ_-^*	11
p_-^{-1}	set-valued inverse projection from Σ_-^* to Σ^*	11
p_-^{ω}	projection from Σ^ω to $\Sigma_-^\omega \cup \Sigma_-^*$	14
$\mathrm{p}_-^{-\omega}$	set-valued inverse projection from $\Sigma_-^\omega \cup \Sigma_-^*$ to Σ^ω	14
cpco	controllability-prefix under complete observation	41

Symbols in the context of input/output systems

Symbol	Description	Page
U	input	57
Y	output	57
Σ_p	internal plant alphabet	60
Σ_e	low-level plant alphabet	60
Σ_c	high-level control alphabet	64
Σ_l	low-level environment alphabet	90
$\Sigma_\mathrm{pe} = \Sigma_\mathrm{p} \dot\cup \Sigma_\mathrm{e}$	plant alphabet	60
$\Sigma_\mathrm{cp} = \Sigma_\mathrm{c} \dot\cup \Sigma_\mathrm{p}$	controller alphabet	59
$\Sigma_\mathrm{el} = \Sigma_\mathrm{e} \dot\cup \Sigma_\mathrm{l}$	environment alphabet	90
$\Sigma := \Sigma_\mathrm{p} \dot\cup \Sigma_\mathrm{e} \dot\cup \Sigma_\mathrm{c}$	overall alphabet	64
$\bar\Sigma := \Sigma_\mathrm{p} \dot\cup \Sigma_\mathrm{e} \dot\cup \Sigma_\mathrm{l}$	modular IO-system alphabet	90
$\mathcal{L}$	IO-plant over alphabet Σ_pe	61
$\mathcal{E}$	IO-specification over alphabet Σ	64
$\mathcal{H}$	IO-controller over alphabet Σ_cp	66
$\mathcal{K}$	IO-closed-loop behaviour over alphabet Σ	67
$\mathcal{I}$	IO-environment over alphabet Σ_el	90
$\mathcal{L}_\Sigma$	full plant behaviour over alphabet Σ	66
$\mathcal{H}_\Sigma$	full controller behaviour over alphabet Σ	66
$\mathcal{L}_\Vert$	IO-plant ω-composition	87
$\mathcal{L}_\mathrm{err}$	error behaviour for IO-plant ω-composition	88
$\mathcal{L}_\mathrm{IO}$	IO-shuffle for IO-plants $\mathcal{L}_n$, $n = 1, ..., m$	88
$\mathcal{L}_{\mathrm{IO},\bar\Sigma}$	full IO-shuffle behaviour over alphabet $\bar\Sigma$	90
$\mathcal{I}_{\bar\Sigma}$	full IO-environment behaviour over alphabet $\bar\Sigma$	91

Bibliography

B. Alpern and F. Schneider, “Defining liveness,” *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, 1985.

C. Baier and M. Kwiatkowska, “On topological hierarchies of temporal properties,” *Fundamenta Informaticae*, vol. 41, p. 259, 2000.

C. Baier and T. Moor, “A hierarchical control architecture for sequential behaviours,” *Workshop on Discrete Event Systems 2012*, pp. 259–264, 2012.

——, “A hierarchical and modular control architecture for sequential behaviours,” *Discrete Event Dynamic Systems*, vol. 25, no. 1-2, pp. 95–124, 2015.

F. L. Baldissera and J. E. R. Cury, “Application of supervisory control theory to guide cellular dynamics,” *Workshop on Discrete Event Systems 2012*, pp. 384–389, 2012.

J. R. Büchi, “Symposium on decision problems: On a decision method in restricted second order arithmetic,” vol. 44, pp. 1 – 11, 1966.

C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.

H. Cho and S. I. Marcus, “On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation,” *Mathematics of Control, Signals and Systems*, vol. 2, no. 1, pp. 47–69, 1989.

——, “Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations.” *Mathematical Systems Theory*, vol. 22, no. 3, pp. 177–211, 1989.

R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya, “Modelling and control of discrete event systems,” *25th IEEE Conference on Decision and Control, 1986*, pp. 604–607, 1986.

——, “Supervisory control of discrete-event processes with partial observations,” *IEEE Transactions on Automatic Control*, vol. 33, no. 3, pp. 249–260, 1988.

E. M. Clarke, E. A. Emerson, and A. P. Sistla, “Automatic verification of finite-state concurrent systems using temporal logic.” *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.

A. E. C. da Cunha, J. E. R. Cury, and B. H. Krogh, “An assume-guarantee reasoning for hierarchical coordination of discrete event systems,” *Workshop on Discrete Event Systems 2006*, pp. 75–80, 2002.

M. H. de Queiroz and J. E. R. Cury, "Modular control of composed systems," *Proceedings of the American Control Conference 2000*, vol. 6, pp. 4051–4055, 2000.

——, "Modular supervisory control of large scale discrete event systems," *Workshop on Discrete Event Systems 2000*, pp. 103–110, 2000.

R. Ehlers, S. Lafortune, S. Tripakis, and M. Vardi, "Bridging the gap between supervisory control and reactive synthesis: Case of full observation and centralized control," *Workshop on Discrete Event Systems 2014*, pp. 222–227, 2014.

E. A. Emerson and E. M. Clarke, "Characterizing correctness properties of parallel programs using fixpoints," in *Automata, Languages and Programming*. Springer Berlin Heidelberg, 1980, vol. 85, pp. 169–181.

E. A. Emerson and C.-L. Lei, "Efficient Model Checking in Fragments of the Propositional Mu-Calculus," *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS)*, pp. 267–278, 1986.

L. Feng and W. Wonham, "Computationally efficient supervisor design: abstraction and modularity," *8th International Workshop on Discrete Event Systems, 2006*, pp. 3–8, 2006.

L. Feng and W. M. Wonham, "On the computation of natural observers in discrete-event systems," *Discrete Event Dynamic Systems*, vol. 20, no. 1, pp. 63–102, 2010.

L. Feng and W. Wonham, "Supervisory control architecture for discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 53, no. 6, pp. 1449–1461, 2008.

P. Gohari and W. M. Wonham, "On the complexity of supervisory control design in the RW framework," *Systems, Man, and Cybernetics, Part B: IEEE Transactions on Cybernetics*, vol. 30, no. 5, pp. 643–652, 2000.

J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, 1979.

G. Jirásková and T. Masopust, "On properties and state complexity of deterministic state-partition automata," in *Proceedings of the 7th IFIP TC 1/WG 202 International Conference on Theoretical Computer Science*, ser. TCS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 164–178.

A. Käß, "Modellbildung und Reglerentwurf eines dreigeschossigen Fahrstuhls," Bachelorarbeit, Lehrstuhl für Regelungstechnik, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2014.

R. Kumar and V. K. Garg, *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, 1995.

R. Kumar, V. Garg, and S. I. Marcus, "On supervisory control of sequential behaviors," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1978 –1985, 1992.

O. Kupferman and M. Y. Vardi, “Synthesis with incomplete informatio,” in *Advances in Temporal Logic*. Kluwer Academic Publishers, 2000, pp. 109–127.

S. Lafortune, “Modeling and analysis of transaction execution in database systems,” *Automatic Control, IEEE Transactions on*, vol. 33, no. 5, pp. 439–447, 1988.

R. J. Leduc, M. Lawford, and W. M. Wonham, “Hierarchical interface-based supervisory control - part ii: Parallel case,” *IEEE Trans. on Automatic Control*, vol. 50, no. 9, pp. 1336–1348, 2005.

libFAUDES, “Software library for discrete-event systems.” Version of January 2015, http://www.rt.eei.uni-erlangen.de/FGdes/faudes, 2015.

F. Lin and W. M. Wonham, “On observability of discrete-event systems,” *Information Sciences*, vol. 44, pp. 173–198, 1988.

F. Lin and W. Wonham, “Decentralized supervisory control of discrete-event systems,” *Information Sciences*, vol. 44, no. 3, pp. 199 – 224, 1988.

C. Ma and W. Wonham, “Nonblocking supervisory control of state tree structures,” *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 782–793, 2006.

Z. Manna and A. Pnueli, “A hierarchy of temporal properties,” *Proc. 9th ACM Symposium on Principles of Distributed Computing*, pp. 377–408, 1990.

R. McNaughton, “Testing and generating infinite sequences by a finite automaton,” *Information and Control*, vol. 9, no. 5, pp. 521 – 530, 1966.

T. Moor, “Natural projections for the synthesis of non-conflicting supervisory controllers,” *Workshop on Discrete Event Systems 2014*, pp. 300–305, 2014.

T. Moor and J. Raisch, “Supervisory control of hybrid systems within a behavioural framework,” *Systems and Control Letters*, vol. 38, no. 3, pp. 157–166, 1999.

T. Moor, J. M. Davoren, and J. Raisch, “Modular supervisory control of a class of hybrid systems in a behavioural framework,” *Proc. Euproean Control Conference (ECC2001)*, pp. 870–875, 2001.

T. Moor, J. Raisch, and J. M. Davoren, “Admissibility criteria for a hierarchical design of hybrid control systems,” *Proc. IFAC Conference on the Analysis and Design of Hybrid Systems (ADHS’03’)*, pp. 389–394, 2003.

T. Moor, K. Schmidt, and S. Perk, “Applied supervisory control for a flexible manufactoring system,” *Workshop on Discrete Event Systems 2010*, pp. 263–268, 2010.

T. Moor, K. Schmidt, and T. Wittmann, “Abstraction-based control for not necessarily closed behaviours,” *Proc. 18th IFAC World Congress*, pp. 6988–6993, 2011.

T. Moor, C. Baier, T.-S. Yoo, F. Lin, and S. Lafortune, “On the computation of supremal sublanguages relevant to supervisory control,” *Workshop on Discrete Event Systems 2012*, pp. 175–180, 2012.

T. Moor, C. Baier, and T. Wittmann, “Consistent abstractions for the purpose of supervisory control,” *IEEE 52nd Annual Conference on Decision and Control (CDC)*, pp. 7291–7296, 2013.

M. Mukund, “Finite-state automata on infinite inputs,” *Internal Report TCS-96-2, SPIC Mathematical Institute*, 1996.

——, “Linear-time temporal logic and büchi automata,” *Tutorial talk, Winter School on Logic and Computer Science, Indian Statistical Institute, Calcutta*, 1997.

P. Pena, J. Cury, and S. Lafortune, “Testing modularity of local supervisors: An approach based on abstractions,” *8th International Workshop on Discrete Event Systems 2006*, pp. 107–112, 2006.

S. Perk, T. Moor, and K. Schmidt, “Hierarchical discrete event systems with inputs and outputs,” *Workshop on Discrete Event Systems 2006*, pp. 427–432, 2006.

——, “Controller synthesis for an i/o-based hierarchical system architecture,” *Workshop on Discrete Event Systems 2008*, pp. 474–479, 2008.

A. Pnueli and R. Rosner, “On the synthesis of a reactive module,” *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 179–190, 1989.

P. J. Ramadge, “Some tractable supervisory control problems for discrete-event systems modeled by büchi automata,” *IEEE Transactions on Automatic Control*, vol. 34, no. 1, pp. 10–19, 1989.

P. J. Ramadge and W. M. Wonham, “Modular feedback logic for discrete event systems,” *SIAM Journal on Control and Optimization*, vol. 25, no. 5, pp. 1202–1218, 1987.

——, “Supervisory control of a class of discrete event processes.” *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

——, “The control of discrete event systems,” *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

K. Rohloff and S. Lafortune, “On the computational complexity of the verification of modular discrete-event systems,” in *Proc. 41 st IEEE Conference on Decision and Control, Las Vegas*, 2002.

K. Schmidt and C. Breindl, “Maximally permissive hierarchical control of decentralized discrete event systems,” *IEEE Transactions on Automatic Control*, vol. 56, no. 4, pp. 723–737, 2011.

K. Schmidt, E. Schmidt, and J. Zaddach, “A shared-medium communication architecture for distributed discrete event systems,” *Mediterranean Conference on Control Automation, 2007.*, pp. 1–6, 2007.

K. Schmidt, T. Moor, and S. Perk, “Nonblocking hierarchical control of decentralized discrete event systems,” *IEEE Trans. on Automatic Control*, vol. 53, no. 10, pp. 2252–2265, 2008.

A. Tarski, “A lattice-theoretical fixpoint theorem and its applications.” *Pacific Journal of Mathematics*, vol. 5, no. 2, pp. 285–309, 1955.

J. G. Thistle, “Supervisory control of discrete event systems,” *Mathematical and Computer Modelling*, vol. 23, no. 11/12, pp. 25 – 53, 1996.

J. G. Thistle and H. M. Lamouchi, “Effective control synthesis for partially observed discrete-event systems,” *SIAM J. Control and Optimization*, vol. 48, no. 3, pp. 1858–1887, 2009.

J. G. Thistle and W. M. Wonham, “Control of ω-automata, church’s problem, and the emptiness problem for tree ω-automata,” *Computer Science Logic*, vol. 626, pp. 367–381, 1992.

——, “Supervision of infinite behavior of discrete-event systems,” *SIAM J. Control and Optimization*, vol. 32, no. 4, pp. 1098–1113, 1994.

——, “Control of infinite behavior of finite automata,” *SIAM J. Control and Optimization*, vol. 32, no. 4, pp. 1075–1097, 1994.

J. Thistle, “On control of systems modelled as deterministic rabin automata,” *Discrete Event Dynamic Systems*, vol. 5, no. 4, pp. 357–381, 1995.

W. Thomas, “Automata on infinite objects,” *Handbook of theoretical computer science (vol. B), The MIT Press, Cambridge, MA*, pp. 133–191, 1990.

J. C. Willems, “Paradigms and puzzles in the theory of dynamical systems,” *IEEE TAC*, vol. 36, no. 3, pp. 259–294, 1991.

K. C. Wong, “On the complexity of projections of discrete-event systems,” *IEEE Workshop on Discrete Event Systems*, pp. 201–208, 1998.

K. C. Wong and W. M. Wonham, “Hierarchical control of discrete-event systems,” *Discrete Event Dynamic Systems: Theory and Applications*, vol. 6, no. 3, pp. 241–273, 1996.

W. M. Wonham and P. Ramadge, “On the supremal controllable sublanguage of a given language,” *The 23rd IEEE Conference on Decision and Control, 1984*, vol. 23, pp. 1073–1080, 1984.

W. Wonham and P. Ramadge, “Modular supervisory control of discrete-event systems,” *Mathematics of Control, Signals and Systems*, vol. 1, no. 1, pp. 13–30, 1988.

T.-S. Yoo, S. Lafortune, and F. Lin, “A uniform approach for computing supremal sublanguages arising in supervisory control theory,” *Technical Report, Department of EECS, University of Michigan*, 2002.

H. Zhong and W. M. Wonham, “On the consistency of hierarchical supervision in discrete-event systems,” *IEEE Transactions on Automatic Control*, vol. 35, no. 10, pp. 1125–1134, 1990.

R. Ziller and J. Cury, “On the supremal Lm-closed and L-controllable sublanguages of a given language,” *Analysis and Optimization of Systems – Discrete Event Systems*, vol. 199, pp. 80–85, 1994.

——, “On the supremal L-controllable sublanguage of a non-prefix-closed language,” *Anais do 10. Congresso Brasileiro de Automática e 6. Congresso Latino-Americano de Controle Automático*, vol. 2, pp. 260–265, 1994.

R. Ziller and K. Schneider, “Combining supervisor synthesis and model checking,” *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 2, pp. 331–362, 2005.

Eigene Publikationen

C. Baier and T. Moor, ”A hierarchical control architecture for sequential behaviours,”*Workshop on Discrete Event Systems 2012*, pp. 259–264, 2012.

——, ”A hierarchical and modular control architecture for sequential behaviours,”*Discrete Event Dynamic Systems,*, vol. 25, no. 1-2, pp. 95–124, 2015.

T. Moor, C. Baier, T.-S. Yoo, F. Lin, and S. Lafortune, ”On the computation of supremal sublanguages relevant to supervisory control,”*Workshop on Discrete Event Systems 2012,* pp. 175–180, 2012.

T. Moor, C. Baier, and T. Wittmann, ”Consistent abstractions for the purpose of supervisory control,”*Conference on Decision and Control (CDC) in IEEE 52nd Annual*, Dec 2013, pp. 7291–7296.

Betreute studentische Arbeiten

A. Käß, ”Modellbildung und Reglerentwurf eines dreigeschossigen Fahrstuhls,” Bachelorarbeit, Lehrstuhl für Regelungstechnik, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2014.

In the Supervisory Control Theory according to P.J. Ramadge und W.M. Wonham, the behaviour of a dynamical system is described by sequences of discrete events. Typical applications are production and transportation systems that are operated by programmable logic controllers. A challenge in this area is the controller design for systems consisting of a large number of components and requiring the coordination of various synchronous processes. The size of a model that describes their synchronous behaviour grows exponentially in the number of components. Hence, monolithic supervisor design using the Supervisory Control Theory in the context of large-scale systems requires a prohibitive amount of computational effort. In order to avoid large monolithic models, several hierarchical and modular procedures have been presented that structure the monolithic control problem vertically and horizontally into manageable synthesis sub-tasks. However, existing approaches are limited to modelling behaviours on a finite-time horizon, using $*$-languages. Whereas synthesis procedures of the monolithic controller design also include the controller design for behaviours that evolve on an infinite-time axis, so called ω-languages, offering thereby a more general field of application.

In this thesis, a hierarchical and modular control approach for ω-languages is developed that is based on the use of abstractions. Therefore, known methods regarding the monolithic controller design for ω-languages are adopted to the hierarchical and modular control context. In addition, dynamical systems are modelled as input/output systems, following the Behavioural Systems Theory of J.C. Willems. For a practical perspective, algorithms to solve the presented hierarchical control problem on the basis of finite-state automata are elaborated and an example from the area of transportation systems is discussed. It demonstrates that the presented approach enables efficient controller design for large-scale systems modelled by ω-languages and reduces the respective computational effort.

ISBN 978-3-96147-028-0

9 783961 470280

FAU UNIVERSITY PRESS 2017