Hierarchical Control of Discrete Event Systems with Inputs and Outputs

Der Technischen Fakultät der

Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR-INGENIEUR

vorgelegt von

Sebastian Perk

Erlangen 2010

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

Tag der Einreichung:	03. 07. 2009
Tag der Promotion:	05. 02. 2010
Dekan:	Prof. Dr.-Ing. Reinhard German
Berichterstatter:	Prof. Dr.-Ing. Th. Moor
	Prof. Dr.-Ing. J. Raisch

Vorwort

Mein erster Dank gebührt dem Betreuer meiner Promotion Herrn Prof. Dr.-Ing. Thomas Moor. Seine Ideen zu den wesentlichen Aspekten des Promotionthemas und die engagierte Betreuung der Promotion von Beginn an legten ein festes Fundament für meine Arbeit. Viel zusätzlicher Freiraum und das somit entgegengebrachte Vertrauen schufen ein optimales Umfeld, um auf dieses Fundament aufzubauen.
Herrn Prof. Dr.-Ing Jörg Raisch danke ich herzlich für die Übernahme des Korreferates und, zusammen mit den Herren Prof. Dr.-Ing. Bernhard Schmauß und Prof. Dr. Volker Strehl, für die bereitwillige Mitwirkung im Prüfungskollegium.
Ich danke außerdem Herrn Prof. Dr.-Ing. Günter Roppenecker, Vorstand des Lehrstuhls für Regelungstechnik der Universität Erlangen-Nürnberg. Sein beständiges Interesse an meiner Arbeit und der regelmäßige Zuspruch gaben mir das Selbstvertrauen, das für eine Promotion die notwendige Vorraussetzung ist.
Allen Kollegen des Lehrstuhls möchte ich danken, denn auf ihren Rat wie auch Aufmunterung konnte ich mich stets verlassen.
Ein ganz besonderer Dank gilt meinem Kollegen und Freund Dr.-Ing. Klaus Schmidt für die nie versiegende Quelle an fachlichen Informationen und Ideen, den Rat in allen Lebenslagen und allem voran für die Freundschaft.
Ich danke meiner Familie für den Rückhalt und für die Liebe und Unterstützung, die ich in sich wunderbar ergänzenden Facetten erfahren durfte.
Der wichtigste Dank gilt von Herzen meiner Frau Claudia, denn ihre beständige Liebe, Unterstützung und Aufmunterung waren eine wesentliche Vorraussetzung, dass diese Arbeit zum erfolgreichen Abschluss kam.

Nürnberg, Februar 2010 Sebastian Perk

Abstract

In the late 1980's, the Supervisory Control Theory (SCT) was proposed by P.J. Ramadge and W.M. Wonham [RW87b], that provides a systematic method for automated synthesis of discrete event controllers with guaranteed safety and liveness properties, but fails for systems of praxis-relevant size due to extensive computational complexity. Since then, research has been aimed at design methods that are based on the SCT, but scale better with the system size by structural exploitation of the problem and thus introduce access to practical applications.

In this thesis, we propose an I/O-based framework for the design of hierarchical controllers for discrete event systems that addresses both safety and liveness properties. Technically, we build on J.C. Willems' behavioural systems theory [Wil91], that describes fundamental properties of dynamic systems like the characteristics of inputs and outputs based on their behaviours. The generality of the behavioural approach allows us to transfer this notion of inputs and outputs to discrete event systems. As a consequence, results from previous work on abstraction-based control of hybrid systems in [MR99, MRD03] could be elaborated for discrete-event dynamics.

The I/O-based approach is applied to systems that consist of a number of interacting components (local subsystems) and builds a hierarchy of superposed controllers and subordinate environment models on the component models. The I/O-based description of the component models is independent of their surroundings (i.e. neighbour components or controller) to obtain reusability within different configurations. Each model of the local subsystems provides two I/O ports, one to interact with a controller and the other one to interact with the environment of the component. At first, local controllers are designed for each component according to local specifications that are independent from the component's environment. While the design method is based on the Supervisory Control Theory, safety and liveness of the closed loop are a consequence of the I/O properties of component model and controller.
On the next layer of the hierarchy, groups of several components each are formed, and their interaction is described by a dynamic environment model. The synthesis of a superposed controller for each group is not performed on the detailed description of the locally controlled component models, but on their abstractions in form of the local specifications, which effectively limits the computational complexity. The abstraction-based controllers are proven to correctly control also the original system. By alternation of the abstraction step, the grouping of components via en-

vironment models and the design of superposed controllers, an overall system is developed that scales well with the number of system components. The expected reduction of complexity became evident also by application to the conceptional example of a transport-unit chain that comes along with this thesis.

Zusammenfassung

Hierarchische Steuerung von ereignisdiskreten Systemen mit Ein- und Ausgängen

In den späten 1980'er Jahren wurde mit der Supervisory Control Theory (SCT) nach P.J. Ramadge und W.M. Wonham [RW87b] ein systematisches Verfahren bereitgestellt, welches die automatische Synthese sicherer und lebendiger Steuerungen für ereignisdiskrete Systeme ermöglicht, sich aber aufgrund des erheblichen Rechenaufwands nicht für Systeme praxisrelevanter Größe eignet. Seither wird nach Entwurfsverfahren geforscht, die auf der SCT aufbauen, jedoch durch Nutzung der Prozessstruktur besser mit der Systemgröße skalieren und somit den Zugang zur praktischen Anwendung ermöglichen, siehe [PMS07b] für einen deutschsprachigen Überblick.
In dieser Arbeit wird ein Ein-Ausgangs-(E/A-) basierter Ansatz zum Entwurf hierarchischer Steuerungen für ereignisdiskrete Systeme vorgestellt, der sowohl Sicherheits- als auch Lebendigkeitseigenschaften berücksichtigt. Dieser ist angelehnt an die so genannte Behavioural Systems Theory nach J.C. Willems [Wil91], welche grundlegende Eigenschaften dynamischer Systeme, wie die Charakteristik von Ein- und Ausgängen, verhaltensorientiert und so allgemeingültig beschreibt, dass sie auch zur E/A-basierten Beschreibung von ereignisdiskreten Systemen herangezogen werden konnte. Auf dieser Grundlage lassen sich Ergebnisse zum abstraktionsbasierten Reglerentwurf für hybride Systeme aus [MR99, MRD03] auf ereignisdiskrete Systeme übertragen.

Beim E/A-basierten Ansatz wird auf einen aus Komponenten bestehenden Prozess eine Hierarchie überlagerter Steuerungen und unterlagerter Umgebungsmodelle aufgebaut. Die E/A-basierte Modellierung der einzelnen Prozesskomponenten erfolgt zunächst umgebungsunabhängig (d.h. unabhängig von Nachbarkomponenten oder der Steuerung). Dies bewirkt ihre Wiederverwendbarkeit innerhalb unterschiedlicher Anordnungen. Jedes E/A-basierte Modell der lokalen Teilsysteme verfügt über zwei E/A-Ports, einer zum Anschluss einer Steuerung, der andere zum Anschluss eines dynamischen Umgebungsmodells, welches jeweils die Interaktion einer Gruppe von Teilsystemen untereinander und mit der restlichen Umgebung beschreibt. Gemäß lokaler von der Umgebung unabhängiger Spezifikationen werden für die einzelnen Prozesskomponenten zunächst lokale Steuerungen entworfen. Die Entwurfsmethodik baut dabei auf die Supervisory Control Theory auf. Der

Nachweis von Sicherheit und Lebendigkeit des geschlossenen Regelkreises gelingt infolge der ein-/ausgangsbasierten Systembeschreibung.
Auf der nächsthöheren Stufe der Hierarchie werden jeweils mehrere Prozesskomponenten zusammengefasst und ihre Interaktion durch ein dynamisches Umgebungsmodell modelliert. Der Entwurf überlagerter Steuerungen für Gruppen von Komponenten greift nicht auf die detaillierte Beschreibung der lokal gesteuerten Prozessmodelle, sondern auf eine Abstraktion derselben auf Grundlage der lokalen Spezifikationen zurück, was den Rechenaufwand wirkungsvoll begrenzt. Die anhand der Abstraktion entworfene Steuerung steuert auch das tatsächliche System nachweislich korrekt. Durch gezieltes Abwechseln der Abstraktionsschritte, der Beschreibung der Interaktion durch Umgebungsmodelle und der Überlagerung von Steuerungen lässt sich ein Gesamtsystem entwickeln, welches mit der Anzahl der Prozesskomponenten gut skaliert. Der erwartete geringe Rechenaufwand ergab sich auch bei der Anwendung auf das konzeptionelle Beispiel einer Kette von Transporteinheiten, welches diese Arbeit begleitet.

. . . after all, salesmen continue to travel . . .

W.M. Wonham

Table of Contents

Chapter 1

Introduction

When facing a complicated technical problem, a powerful instrument of successful engineering practice is to exploit the structure of the problem until it can be cast to one or a number of simpler problems whose solution applicably exists. The *control of discrete event systems* (DES) is a complicated problem. Though it has been formally solved in the late 1980's by the supervisory control theory (SCT) of Ramadge and Wonham, which delivers controllers with guaranteed correctness and performance, the model-based approach did get only limited access to industrial deployment, which is mainly due to the affinity of DES to intractable complexity of the plant model.
In contrast, engineers manage to automatize even large-scale DES such as logistics, communication networks and manufacturing systems. By practical experience and technologies like divide and conquer strategies, measurement aggregation and last but not least the hiding of apparently less relevant plant behaviour , the original problem is turned into one with manageable complexity. However, this inevitably limited view on the original problem has a number of unpleasant consequences. As correctness of the controller software cannot be formally guaranteed, usually a large number of trial and error runs is necessary for debugging. Depending on the existence and accuracy of a simulation model, a considerable number of trials run after sales on the real plant involving safety problems and high costs. Further issues are suboptimal capacity utilization as well as low configurability and low scalability. Hence the question arises how to avoid such shortcomings when organizing the problem in an applicable manner.
Since the proposal of the SCT, considerable research effort was spent on incorporating the mentioned engineering skills in model-based discrete event controller design to reduce computational complexity without the loss of the guarantees gained by the SCT. Let us introduce an example to illustrate some of the challenges in control of DES.

Example 1.1
We consider a small manufacturing line as in Figure 1.1.

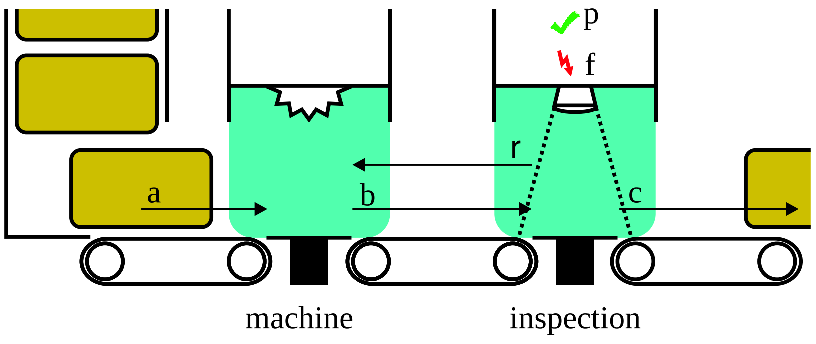
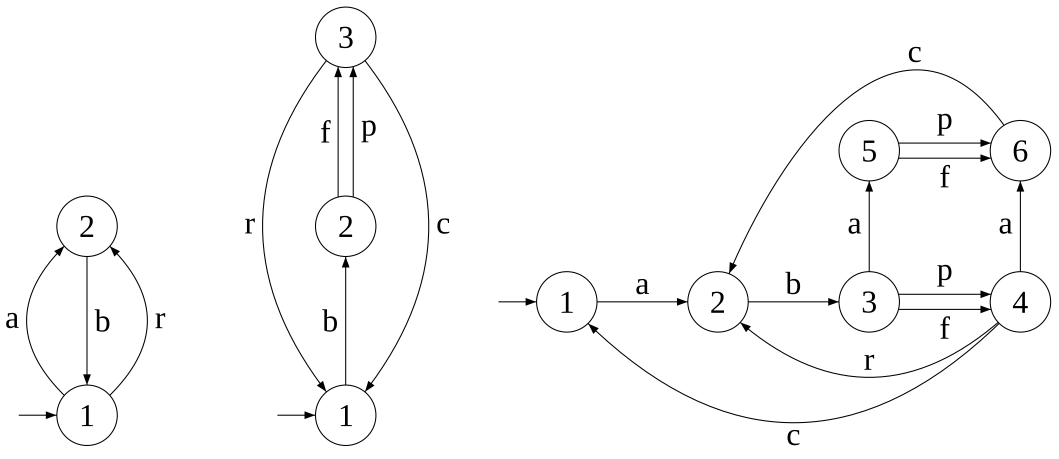


Figure 1.1: Manufacturing line

From an always filled stack feeder, raw workpieces enter a machine via a conveyor belt and are processed (a). When the process in the machine is finished, the workpiece proceeds to an inspection unit (b). The inspection results are “pass” (p) or “fail” (f). After inspection, workpieces can exit the manufacturing line (c) or return to the machine (r). The discrete event behaviour of this technical process can be represented by an automaton model of the machine and of the inspection unit, see Figure 1.2 a) and b), respectively. The occurrence of events is denoted by transitions, which are visualized as arrows labeled with the triggering event that lead from the system state (drawn as circle) before the transition to the state after the transition. The initial system state (here: states with label 1) is denoted by a sourceless arrow. The behaviour of the whole line is given by the synchronous composition of both automata, see Figure 1.2 c). The composition of all plant components is denoted *monolithic plant model*.

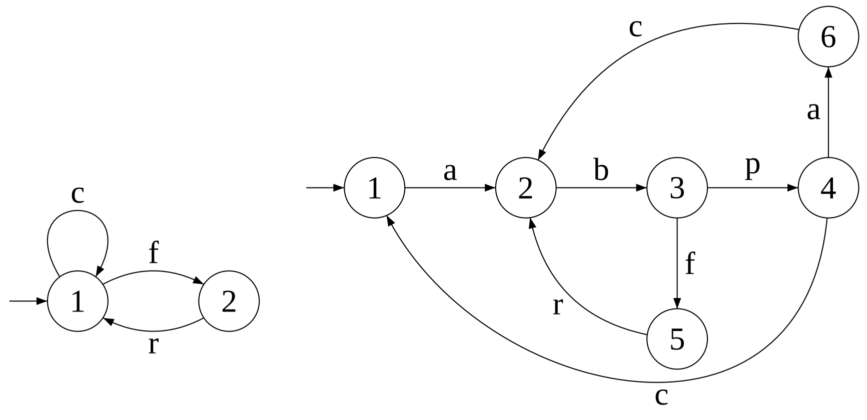


(a) machine (b) inspection (c) composed behaviour

Figure 1.2: Manufacturing line: automata models

As can be seen, the number of states of the monolithic model is the product (rather than the sum) of the state counts of the machine and the inspection model; i.e. we face the general case, where

the complexity of the whole plant (counted in number of states) is exponential in the number of plant components.
The control objective for this example shall be the reprocessing (r) of a workpiece in case it fails inspection (f). The according specification forbids the release (c) of those workpieces from the manufacturing line that failed inspection but requires reprocessing instead; it is easily formulated as an automaton, see Figure 1.3 a). For the enforcement of the specification on the plant, the uncontrollability of the events f and p has to be taken into account, as they cannot be directly disabled by a controller.
If this specification is applied directly to the plant, then the resulting behaviour blocks: consider state 6 in Figure 1.2 c) and assume it was reached from state 5 via the event f, i.e. a workpiece just failed inspection. Then, the specification forbids event c, but at the same time the required reprocessing (r) is not possible, as the machine is occupied by another workpiece. Hence, no further event is possible, and the system gets stuck in a deadlock.
The *supervisory control theory* (SCT, [RW87b]) provides an efficient algorithm to compute a minimally restrictive supervisor such that the closed-loop behaviour of the plant under supervisory control meets the given specification (*safety*) and is nonblocking (*liveness*).[1] The resulting closed-loop behaviour of the manufacturing line is shown in Figure 1.3 b). As can be seen, the supervisor avoids the aforementioned deadlock by allowing for only one workpiece in the line until the inspection result is positive: a second occurrence of the the event a is disabled until the occurrence of the event p.



(a) specification (b) closed-loop behaviour

Figure 1.3: Manufacturing line: supervisory control

Note that the closed loop-behaviour can indeed be achieved by a supervisor, as only controllable events are be disabled. This fundamental condition for successful controller design is denoted

[1]The avoidance of blockings is achieved by the marking technology which, for simplicity, is not considered here.

controllability. The model of the closed-loop behaviour serves as realization of the supervisor and can be implemented e.g. in the form of PLC code. □

As the most appealing feature of a supervisor designed according to the SCT and as a consequence of the model-based approach, the closed-loop behaviour is *guaranteed* to comply with the specification, to be nonblocking and to be minimally restricted. On the downside of the SCT, the supervisor has to be computed on the basis of the monolithic plant model, whose complexity is prohibitive in most practical applications. Example calculation: the monolithic model of a plant consisting of 10 components with 10 states each can embrace up to 10^{10} states.
The reason why, aside from the shortcomings mentioned at the outset, engineers successfully design controller software for e.g. large scale automation systems lies in the structural exploitation of the design problem: usually, the software is modular, composed of subroutines for different control tasks and different functional system components. For superposed control tasks, the view on the system is aggregated adequately by respecting only the features relevant to the task.

Since the supervisory control theory was proposed in the late 1980's, a major contingent of research in the field of discrete event systems has aimed at *structured approaches* based on the SCT that reduce the complexity of controller design and at the same time preserve properties such as guaranteed enforcement of the specification and nonblocking.
Modular approaches such as [RW87a, WR88, RW89, dQC00, QC00, GM04, MF08] are useful if the overall control objective is given as a set of specifications for individual tasks. One supervisor controlling the whole plant is designed for each specification, see Figure 1.4.

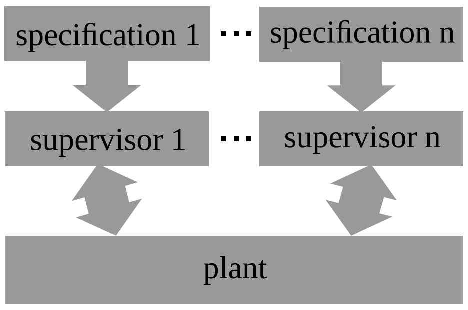


Figure 1.4: modular control architecture

While still the composed plant has to be computed, complexity reduction results from the low complexity of the individual specifications compared to their composition. The possibility of conflicts (in case the interaction of the modular supervisors causes blocking) requires the test for nonblocking closed-loop behaviour e.g. as in [FM06] or structural conditions that avoid conflicts.

In *decentralized approaches*, the overall control task is performed by a set of supervisors (Figure 1.5), each of which controls only one component of the plant ("local control") and thus is expected to feature low complexity.

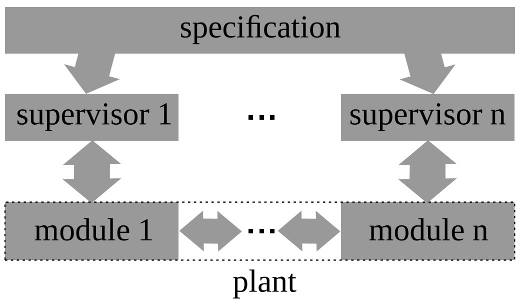
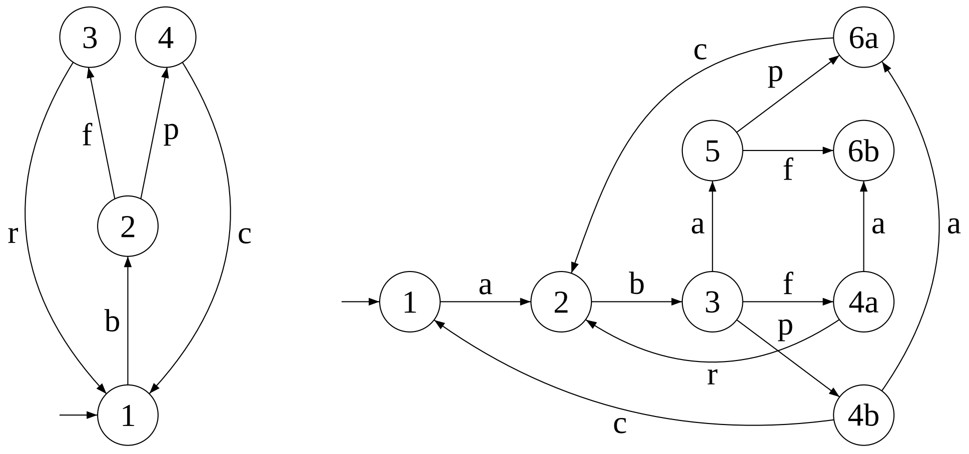


Figure 1.5: decentralized control architecture

Without further measures, the plant-wide enforcement of the specification is not guaranteed, and the behaviour of the local supervisors within each other or with other plant components may be conflicting. We reconsider the manufacturing line example to illustrate such conflict situation.

Example 1.2
We apply the specification for the manufacturing line (Figure 1.3 a)) separately to the machine and to the inspection unit. The resulting supervisor for the machine does not restrict its behaviour and may be omitted. The locally controlled behaviour of the inspection unit is shown in Figure 1.6 a).



(a) locally controlled inspection (b) closed-loop behaviour

Figure 1.6: Manufacturing line: decentralized control

As can be seen, the locally controlled inspection unit is free of deadlocks. When composed with the machine, the resulting behaviour of the overall plant (Figure 1.6 b)) meets the specification but, however, is blocking - observe the deadlock in state 6b. □

Approaches like [CDFV88, LW90, BGK+90] guarantee safety and liveness by requiring structural conditions (see also [LW91]). However, computations still require the detailed monolithic plant model. In extended decentralized approaches ([WH91, LW97, LW02, KvS04]), the composition of the plant components and hence the exponential growth of complexity is effectively avoided. Additionally, [SMG06] provides a method to exhibit modular or decentralized supervisor design on reduced system models.

The idea of information hiding has lead to *hierarchical approaches* (e.g. based on [ZW90, WW96, dCCK02, HC02]), that map the original plant to one or more superposed layers of less complex high-level-models, where the degree of abstraction is oriented towards the according specification. The complexity is reduced by designing the supervisor for the high-level model. From this supervisor, that virtually controls the abstracted model, an implementation for the original plant has to be derived, see Figure 1.7.

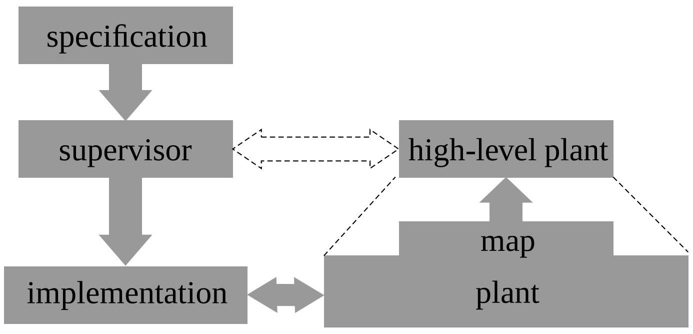


Figure 1.7: hierarchical control architecture

The monotonicity of the involved operators guarantees enforcement of the specification on the original plant by the implementation. If, moreover, the abstraction of the closed loop of implementation and original plant exactly matches the closed loop of abstracted plant and supervisor, the desirable property of *hierarchical consistency* is met. If not, then the implementation can be conservative. Furthermore, measures have to be taken such that the resulting closed-loop behaviour is nonblocking. In [Led02], low- and high-level are connected by a layer of particular interfaces, and desired properties like nonblocking are achieved by a request and answer structure of the involved components. As this structure can be met individually by the plant components, the detailed monolithic plant model never needs to be computed. As one of the first approaches, applicability has been proven by physical, industry-oriented examples, see [Led96, Wen06].

By combining the hierarchical and the decentralized method, the *hierarchical-decentralized* approach ([Sch05, SMP08]) allows for a multi-level hierarchy of supervisory control by alternation of decentralized control, hierarchical abstraction and subsystem composition. The implementation of each high-level supervisor restricts the behaviour of the subordinate supervisors, down to the layer of plant components, in a way such that the whole architecture is hierarchically consistent.

Moreover, reasonable structural conditions and reasonable conditions for the abstraction map are identified that guarantee nonblocking closed-loop behaviour. The applicability of the approach to large-scale DES has been demonstrated with a laboratory case study, see also [Per04]. Extensions cover the maximal permissiveness ([SB08]) and the distributed implementation of the supervisors over communication networks ([SSZ07]).

A method that has been extensively studied in supervisory control of hybrid systems is *abstraction based control*; see e.g. [CKN98, RO98, KASL00, MR99]. In such approaches, the original plant is replaced by an approximation that relates to the original by a simple subset relation: in the abstraction, a less detailed likewise less complex description of the system behaviour allows for more possible system trajectories compared to the original plant model, see Figure 1.8.

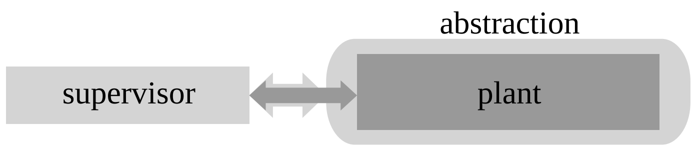
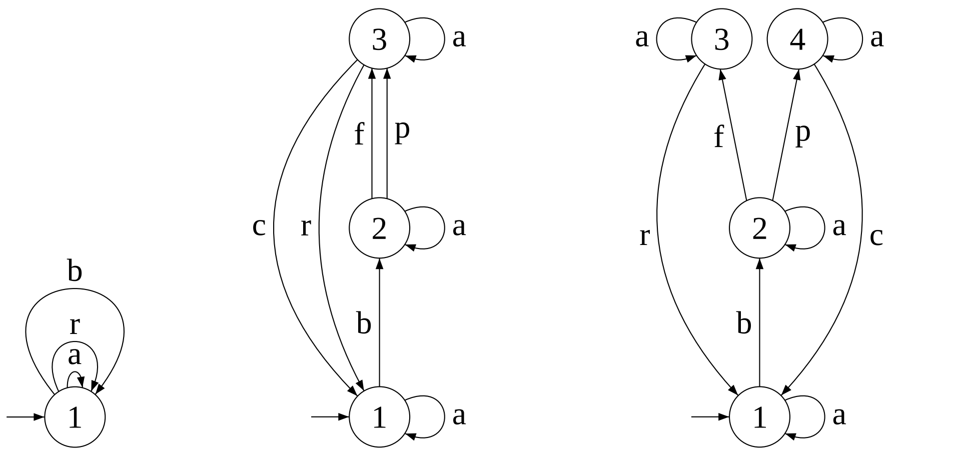


Figure 1.8: abstraction-based control

The supervisor is designed for the abstracted model and then applied to the original plant. Similar to the hierarchical approach, safety is guaranteed by monotonicity, while liveness has to be dealt with separately.

Example 1.3
We replace the original plant model of the manufacturing line by an abstraction to show how computational savings can be made. For the machine that can hold not more than one workpiece, we introduce a half as complex model that ignores the limited capacity, see Figure 1.9 a). Observe that any sequence of events in the original model (Figure 1.2 a)) is also possible in the abstraction, i.e. the abstraction meets the required subset relation. Also the composition with the inspection unit to an abstracted model of the plant (Figure 1.9 b)) is of lower complexity and a valid abstraction due to monotonicity of the composition operator. The realization of the abstraction-based supervisor is depicted in Figure 1.9 c).



(a) machine, abstracted (b) plant, abstracted (c) closed-loop behaviour, abstracted

Figure 1.9: Manufacturing line: abstraction-based control

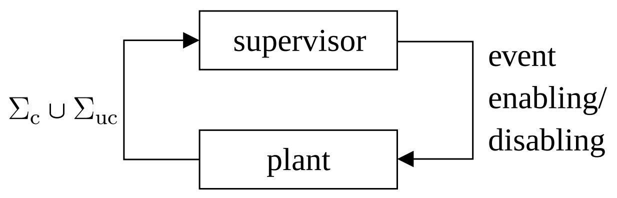
Observe that also the supervisor realization is less complex. Unfortunately, the closed loop of the abstraction-based supervisor and the original plant matches with the blocking behaviour in Figure 1.6 b) that has also been achieved by decentralized control. Hence, as noticed before, the enforcement of the specification is preserved, while liveness, in general, is not. □

Usually, the liveness of the resulting closed loop is ensured by structural conditions on the original plant and the supervisor only, such that the subset relation remains the only condition required for the abstraction. Hence, its degree can be chosen freely between arbitrary and original behaviour which can result in considerable computational savings. Naturally, too coarse abstractions lead to excessively restricted closed-loop behaviour.

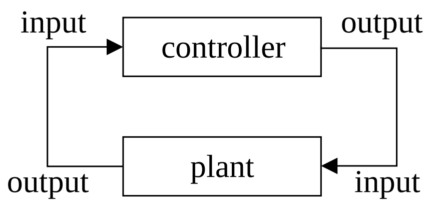
Contribution and Outline of the Thesis

For discrete event controller design, the preservation of both, controllability and liveness properties are problems of primary concern in all approaches based on the SCT. Interestingly, these problems seem to be a specialty of the class of discrete event systems. Example: in the control of systems with continuous dynamics described according to linear systems theory, the violation of comparable properties is not observed: in the closed loop of any controller and any plant, the controller never directly changes measurement signals issued the plant, but does influence the plant output only indirectly via the plant input (≅ controllability). Also, the trajectories in the closed loop never break up, as any system accepts arbitrary input signals and always, there exists an according output signal (≅ no deadlocks). Hence, basic properties are given a priori by the input-/output-based system description rather than by additional measures in controller design.
This difference between the two fields of control theory does not lie solely in the different nature of the considered dynamics, but also in the different view on it:

In the SCT, the plant model is interpreted as a system that by itself spontaneously generates controllable events Σ_c and uncontrollable events Σ_{uc}. The influence by a supervisor is passive, by enabling or disabling the the occurrence of controllable events. An input-/output based system description differs from this paradigm in that systems perform interaction by actively generating output signals and passively accepting output signals, see Figure 1.10 b). In [BHP+93, Wen06], the former and the latter model interpretations are denoted *asymmetric* and *symmetric* feedback, respectively.



(a) asymmetric feedback in the SCT



(b) symmetric feedback with I/O structure

Figure 1.10: Comparison of SCT- and I/O-feedback types [BHP+93]

Our approach develops an input-/output-based (I/O-based) description for DES aiming at a notion of inputs and outputs for DES that

(i) legitimates a direction of cause and effect as in Figure 1.10 b).

(ii) achieves controllability and basic liveness properties for the closed loop of *any* controller and *any* plant as a consequence of the I/O structure.

(iii) allows for abstraction-based controller synthesis.

(iv) enables hierarchical design of the plant model and the control system.

(v) exploits the structure of composed systems similarly to decentralized approaches.

(vi) facilitates the description of a discrete event model in separation from its surroundings via its input and output such that the model is reusable within various configurations.

In references such as [LT89, BHP+93, Bal94, KGM95, JMRT08], discrete event models are provided with different notions of inputs and outputs, each adequate to the considered problem. In our approach and in contrast to the references, the notion of inputs and outputs and relevant fundamental properties like the novel event-based notion of Y_P-liveness are derived from J.C. Willems' behavioural systems theory [Wil91], which, due to its generality, can in principle be directly adopted to DES to meet the above items (i) and (ii). This allows us to build on the core ideas of [MR99] on abstraction-based control of hybrid systems and the hierarchical extension [MRD03] (for items (iii) and (iv)), as both are stated within the behavioural systems theory. To meet items (iv) and (v), we introduce further extensions required for subsystem composition and a two-sided controller- and environment hierarchy. Here, we refer to approaches like [GM05, Led02, Ma04, Sch05] where the vertical (de)-composition introduced by a hierarchical architecture is complemented by a horizontal (de)-composition of modular or decentralized supervision.

In this contribution, we propose to alternate subsystem composition and controller synthesis resulting in a hierarchical control system that complements a hierarchical plant model; see Figure 1.11.

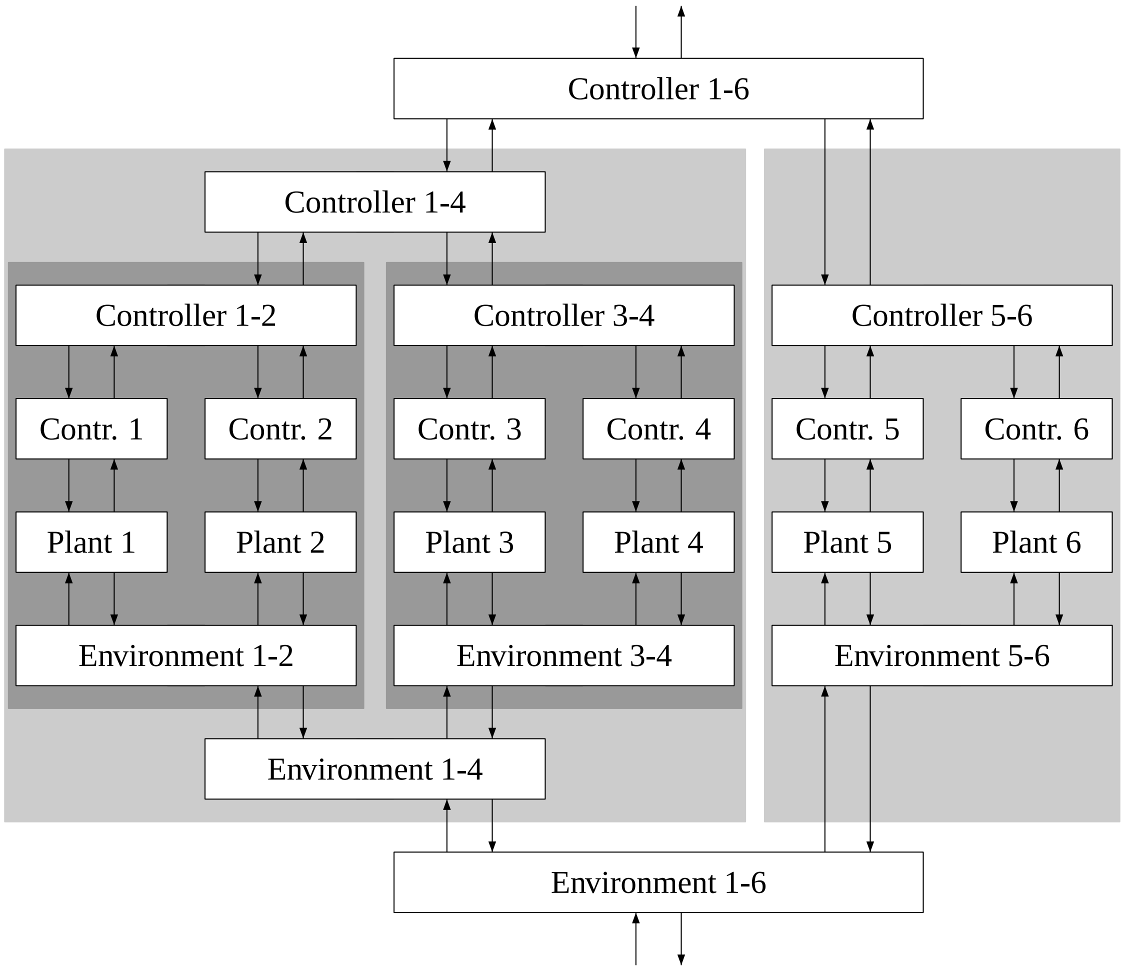


Figure 1.11: Hierarchical control system for a plant with 6 interactive components

On the innermost level of the hierarchy, all subsystems are modeled independently from their environment aiming at reusability within various configurations. For each subsystem model, local

controllers are designed to enforce local specifications that model the desired external behaviour of the closed loop. In the design step, additional assumptions on the external configuration can be taken into account by well-defined constraints. Their enforcement is passed on to the next level of superposed control.
On the next level of the hierarchy, we use the specifications of the preceding level as an abstraction of the controlled subsystems. The admissibility of this abstraction follows directly from the I/O-based system structure. We then synthesize controllers for groups of abstracted low-level control systems. The latter have been designed independently, so constraints in interconnection of groups of subsystems (e.g. shared resources) have not yet been considered. Our framework accounts for such constraints by a hierarchy of environment models that complements the hierarchy of controllers: each dynamic environment model describes the interaction within one group of locally controlled subsystems.
The complexity of the compound group models is effectively reduced by the preceding abstraction step. As a benefit of our framework, the controllability and liveness of each hierarchical layer directly result from the I/O-based system structure. Superposed controllers designed for each group based on the abstractions solve the control problem provably also for the original groups of subsystems.
The alternation of system composition, controller synthesis, abstraction and environment interconnection is continued in a bottom-up fashion until a single top-level controller is synthesized to control an abstract overall model.

The outline of the thesis is as follows. In the following chapter, we introduce the notation and terminology of the formal language framework which is used in this thesis to elaborate and to express theoretical results. The notion of an automaton is introduced to serve as graphical representation of languages.
Chapter 3 proposes and explains the I/O-based description of DES and presents the definition of the I/O plant that interacts with an operator and an environment. The desired liveness properties are presented in an I/O-based formulation and in the presence of constraints on the plant's external configuration. The conceptional example of a transport unit is introduced that goes along with the whole thesis to illustrate the formal statements.
Then, the notion of an I/O controller is defined as an operator of the plant. As the first main result, an admissibility condition for the I/O controller is identified that guarantees liveness for the closed loop. Moreover, the external view on the closed loop features I/O plant properties and thus is ready for further superposed control.
Next, the controller synthesis problem is formally defined introducing the specification as a model of the external view on the desired closed loop behaviour. Finally, an important theorem is proposed stating that the original synthesis problem is readily solved based on an abstraction of the I/O plant.
In Chapter 4, the synthesis of an admissible I/O controller is considered. The notion of Y_{C}-acyclic sublanguages is introduced featuring a unique supremal element to achieve the desired Y_{C}-liveness

property for the closed loop. As the controller has to meet the I/O-structure and cannot directly observe the plant's interaction with the environment, controller synthesis involves the computation of a complete, controllable and normal sublanguage. A controller synthesis procedure is presented and proven to deliver a solution to the synthesis problem, i.e. an admissible I/O controller.

Chapter 5 studies the design of a hierarchical control system as in Figure 1.11. The first part considers the description of a group of subsystems in a compound model that is ready for controller design. First a technical shuffle compound of the components is computed by the I/O shuffle operator. Then, the I/O environment model is formally defined that captures the concurrent behaviour of the components and their interaction with the remaining environment. As the main results of this chapter, I/O plant properties are proven for the external view on the compound of I/O shuffle and I/O environment. Moreover, constraints for the external configuration of the compound are identified that preserve the liveness of the involved subsystems, such that an admissible controller can be computed according to the previous chapters.

The second part of the chapter gives guidance to step by step develop a hierarchical control system based on the presented results. The applicability to multi-component DES is shown by the application to a chain of transport units and by evaluation of the complexity that results for this example.

Chapter 2

Formal Languages: Notation and Terminology

The concept of formal languages originates from computer sciences and has been adopted to a control theoretic context by the Supervisory Control Theory in order to mathematically describe the dynamical behaviour and the properties of discrete-event systems. Also in this framework, formal languages are used as the standard tool to express and prove formal statements, while finite automata serve as a graphical representation of languages. In the following, we provide an overview on the notation used in this text to describe discrete event systems in a language-based framework. For an elaborate introduction to discrete event systems, we refer to [Won08, CL08].

Alphabet, Kleene-closure and strings. Let Σ be a finite set of distinct symbols, called finite alphabet. The Kleene-closure Σ^* is the set of finite *strings* over Σ; i.e.

$$\Sigma^* = \{s | \exists\, n \in \mathbb{N},\ \forall\ i \leq n : \sigma_i \in \Sigma,\ s = \sigma_1\sigma_2\cdots\sigma_n\} \cup \{\epsilon\}$$

with the *empty string* $\epsilon \in \Sigma^*$. The *length* of a string $s = \sigma_1\cdots\sigma_n$ is denoted $|s| = n$, with $|s| = 0$ if $s = \epsilon$ and $|s| = k$ if $s = \sigma_1\cdots\sigma_k$ with $\sigma_i \in \Sigma$ for $i = 1..k$. A string $r = st$ with $s, t \in \Sigma^*$ is called concatenation of s and t. If for two strings $s, r \in \Sigma^*$ there exists $t \in \Sigma^*$ such that $s = rt$, we say r is a *prefix* of s and write $r \leq s$; moreover, r is called *strict* prefix of s if $s \neq r$ and write $r < s$. A prefix of s of length $n \in \mathbb{N}_0$ is denoted s^n.

Language. A language over Σ is a subset $\mathcal{L} \subseteq \Sigma^*$. Note that Σ itself and any of its subsets are languages. A language potentially can contain an infinite number of strings.

Operations on languages. Besides the ordinary set operations union, intersection, set difference and complement w.r.t. Σ^*, the following language operations are common practice, see e.g. [CL08]:

- Concatenation of $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Sigma^*$: $\mathcal{L}_1\mathcal{L}_2 := \{st \in \Sigma^* | s \in \mathcal{L}_1 \wedge t \in \mathcal{L}_2\}$

- Kleene-closure of $\mathcal{L} \subseteq \Sigma^*$: $\mathcal{L}^* := \{\epsilon\} \cup \mathcal{L} \cup \mathcal{L}\mathcal{L} \cup \mathcal{L}\mathcal{L}\mathcal{L} \cup \cdots$
- Prefix closure of $\mathcal{L} \subseteq \Sigma^*$: $\overline{\mathcal{L}} = \{r | \exists\, s \in \mathcal{L} : r \leq s\} \subseteq \Sigma^*$

A language $\mathcal{L}$ is *prefix closed* if $\mathcal{L} = \overline{\mathcal{L}}$. The prefix closure distributes over unions, i.e.

$$\overline{\mathcal{L}_1 \cup \mathcal{L}_2} = \overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}$$

Completeness.[KGM92][1] The language $\mathcal{L}$ is *complete* if

$$(\forall s \in \mathcal{L} :\ \exists\, \sigma \in \Sigma)\ [s\sigma \in \overline{\mathcal{L}}] \tag{2.1}$$

Technically, $\mathcal{L} = \{\varnothing\}$ is complete. A language $\mathcal{L}$ is complete if and only if $\overline{\mathcal{L}}$ is complete ([KGM92]).

Regular expressions and regular language. A way to represent languages over an alphabet Σ in a compact fashion is to use regular expressions, defined recursively as follows ([CL08]):

1. The empty language $\{\varnothing\}$ is denoted by the regular expression $\varnothing$, the empty string language $\{\epsilon\}$ is denoted by the regular expression ϵ, and σ is a regular expression denoting the set $\{\sigma\}$, for all $\sigma \in \Sigma$.
2. If r and s are regular expressions, then so are rs, $(r + s)$, r^*, s^*, representing the concatenation, union and Kleene-closure of the sets represented by r and s, respectively.
3. An expression is not regular unless it is built by the finite-wise application of the above rules 1 and 2.

A language that can be represented by a regular expression is called *regular language*.

Automaton. Automata serve as a compact graph-based representation of languages that is useful for visualization, storage and algorithmic processing. In this text, we consider only deterministic finite automata.

Definition 2.1 (Automaton [HU79])
A *deterministic finite automaton* is a 5-Tuple $G := (Q, \Sigma, \delta, q_0, Q_m)$ consisting of

- Q: the finite set of states
- Σ: the finite alphabet
- $\delta : Q \times \Sigma \to Q$ the unique partial transition function

[1]This notion should not be confused with the notion of complete behaviours in [Wil91].

- q_0: the initial state
- $Q_m \subseteq Q$: the set of marked states

□

Chapter 1 provides several examples of automata graphs.We write $\delta(q,\sigma)!$ if δ is defined at $q \in Q$ and $\sigma \in \Sigma$. We can extend δ to a partial function on $Q \times \Sigma^*$ by defining recursively:

1. $\delta(q,\epsilon) := q,\ \forall q \in Q$
2. $\delta(q,s\sigma) = \delta(\delta(q,s),\sigma)$ whenever both $\delta(q,s) = q' \in Q$ and $\delta(q',\sigma)!$.

The *active event set* of a state $q \in Q$ is defined as $\Lambda(q) := \{\sigma|\delta(q,\sigma)!\}$. A state q with $\Lambda(q) = \varnothing$ is called *deadlock*. Moreover, a state is called *reachable* or *accessible* if there exists a path from the initial state to this state. An automaton is reachable/accessible if all states are reachable/accessible. An automaton is *nonblocking* if from every reachable state there exists a path to a marked state. A nonblocking and reachable automaton is denoted *trim*. An automaton *generates* a prefix-closed language $\mathcal{L}(G)$ and *marks*[2] a language $\mathcal{L}_{\mathrm{m}}(G) \subseteq \mathcal{L}(G)$ as described in the subsequent definition.

Definition 2.2 (Generated and Marked Language, e.g. [CL08, Won08])
For an automaton $G = (Q,\Sigma,\delta,q_0,Q_m)$ the generated language is defined as

$$\mathcal{L}(G) := \{s \in \Sigma^* \mid \delta(q_0,s)!\}$$

and the marked language is

$$\mathcal{L}_{\mathrm{m}}(G) := \{s \in \Sigma^* \mid \delta(q_0,s) \in Q_m\}.$$

□

Hence, an automaton is nonblocking iff $\mathcal{L}(G) = \overline{\mathcal{L}_{\mathrm{m}}(G)}$.

Minimal automaton and Nerode Equivalence. A deterministic automaton defines a partition of $\mathcal{L}(G)$ and $\mathcal{L}_{\mathrm{m}}(G)$ into classes of strings leading to the same state. According to [HU79], for each regular language $\mathcal{L}$, there exists a (substantially) unique deterministic finite automaton, called *minimal automaton*, that marks $\mathcal{L}$ with a *minimal* number of states.[3] A minimal automaton G provides a partition of Σ^* into strings leading to the same state that equals the partition of Σ^* into strings that are *nerode-equivalent* w.r.t. $\mathcal{L}_{\mathrm{m}}(G)$:

[2] In the DES literature, also the terms "*accepts*" and "*recognizes*" are used adequately to the context.
[3] "substantially" means that the minimal automaton is unique except for isomorphisms like renaming of states.

Definition 2.3 (Nerode Equivalence, e.g. [Won08], orig. [Ner58])
The *Nerode equivalence* relation on Σ^* with respect to $\mathcal{L} \subseteq \Sigma^*$ is defined as follows. For $s, t \in \Sigma^*$,

$$s \equiv_{\mathcal{L}} t \;\Leftrightarrow\; (\forall u \in \Sigma^* : su \in \mathcal{L} \;\Leftrightarrow\; tu \in \mathcal{L}). \tag{2.2}$$

□

For $\mathcal{L} \subseteq \Sigma^*$, two strings $s, t \in \Sigma^*$ with $s \equiv_{\mathcal{L}} t$ are called *nerode-equivalent w.r.t.* $\mathcal{L}$. Note that all strings from the set $\Sigma^* - \overline{\mathcal{L}}$ are nerode-equivalent w.r.t. $\mathcal{L}$, as they have no extension to a string of $\mathcal{L}$. In the minimal automaton, these strings are represented by a single state state, usually denoted "dump-state", from which there exists no path to a marked state. The dump-state usually is omitted, as only the strings belonging to $\overline{\mathcal{L}}$ are of interest.

Natural projection and inverse projection. The natural projection allows to erase those events from strings whose observation is either impossible or undesirable.

Definition 2.4 (Natural Projection, e.g. [CL08, Won08])
The *natural projection* $p_\text{o}\colon \Sigma^* \to \Sigma_\text{o}^*$, $\Sigma_\text{o} \subseteq \Sigma$, is defined iteratively:

1. let $p_\text{o}(\epsilon) := \epsilon$;
2. for $s \in \Sigma^*$, $\sigma \in \Sigma$, let $p_\text{o}(s\sigma) := p_\text{o}(s)\sigma$ if $\sigma \in \Sigma_\text{o}$, or $p_\text{o}(s\sigma) := p_\text{o}(s)$ otherwise.

The set valued inverse of p_o is denoted $p_\text{o}^{-1}\colon \Sigma_\text{o}^* \to 2^{\Sigma^*}$ and defined

$$p_\text{o}^{-1}(s) := \{t \in \Sigma^* | p_\text{o}(t) = s\} \text{ for } s \in \Sigma_\text{o}^*$$

□

As the above definition indicates, the projection distributes over concatenation, i.e.

$$p_\text{o}(st) = p_\text{o}(s)p_\text{o}(t),\; s, t \in \Sigma^*.$$

The projection and its inverse are defined for languages by

$$p_\text{o}(\mathcal{L}) := \{p_\text{o}(s) | s \in \mathcal{L}\}$$

and

$$p_\text{o}^{-1}(\mathcal{L}_o) := \{s | p_\text{o}(s) \in \mathcal{L}_o\}$$

for $\mathcal{L} \subseteq \Sigma^*$ and $\mathcal{L}_o \subseteq \Sigma_\text{o}^*$, respectively.

When extended to languages, the projection distributes over unions but, in general, not over intersection, i.e.

$$\begin{aligned} p_o(\mathcal{L}_1 \cup \mathcal{L}_2) &= p_o(\mathcal{L}_1) \cup p_o(\mathcal{L}_2), \\ p_o(\mathcal{L}_1 \cap \mathcal{L}_2) &\subseteq p_o(\mathcal{L}_1) \cap p_o(\mathcal{L}_2) \text{ (Appendix, Lemma A.2)} \end{aligned} \tag{2.3}$$

for $\mathcal{L}_i \subseteq \Sigma^*$, $\Sigma_o \subseteq \Sigma$, and p_o as defined above. The inverse projection distributes over unions and intersection, i.e.

$$\begin{aligned} p_o^{-1}(\mathcal{L}_1 \cup \mathcal{L}_2) &= p_o^{-1}(\mathcal{L}_1) \cup p_o^{-1}(\mathcal{L}_2), \\ p_o^{-1}(\mathcal{L}_1 \cap \mathcal{L}_2) &= p_o^{-1}(\mathcal{L}_1) \cap p_o^{-1}(\mathcal{L}_2). \end{aligned}$$

Prefix closure commutes with projection and inverse projection:

$$\begin{aligned} p_o(\overline{\mathcal{L}}) &= \overline{p_o(\mathcal{L})}, \\ p_o^{-1}(\overline{\mathcal{L}}) &= \overline{p_o^{-1}(\mathcal{L})} \end{aligned}$$

for $\mathcal{L} \subseteq \Sigma^*$, $\Sigma_o \subseteq \Sigma$, and p_o as defined above.

Synchronous composition. An important operation on languages and automata is the synchronous composition, which is used to describe the interconnection of two DES.

Definition 2.5 (Synchronous Composition, e.g. [CL08, Won08])
The synchronous composition[4] of two languages $\mathcal{L}_i \subseteq \Sigma_i^*$, $i \in \{1,2\}$, is defined

$$\mathcal{L}_1 \parallel \mathcal{L}_2 := p_1^{-1}(\mathcal{L}_1) \cap p_2^{-1}(\mathcal{L}_2)$$

where the projections p_i are defined with domain $(\Sigma_1 \cup \Sigma_2)^*$ and range Σ_i^*.

The synchronous product of two deterministic automata $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ is

$$G_1 || G_2 := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{1||2}, (q_{0,1}, q_{0,2}), Q_{m,1} \times Q_{m,2})$$

with

$$\delta_{1||2}((q_1,q_2),\sigma) := \begin{cases} (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma)) & \text{if} \quad \sigma \in \Lambda_1(q_1) \cap \Lambda_2(q_2) \\ (\delta_1(q_1,\sigma), q_2) & \text{if} \quad \sigma \in \Lambda_1(q_1) - \Sigma_2 \\ (q_1, \delta_2(q_2,\sigma)) & \text{if} \quad \sigma \in \Lambda_2(q_2) - \Sigma_1 \\ \text{undefined} & \text{else} \end{cases}$$

□

[4] Also denoted parallel composition or synchronous product

The synchronous composition of two automata represents the synchronous composition of the corresponding languages: $\mathcal{L}(G_1 \parallel G_2) = \mathcal{L}(G_1) \parallel \mathcal{L}(G_2)$ and $\mathcal{L}_\mathrm{m}(G_1 \parallel G_2) = \mathcal{L}_\mathrm{m}(G_1) \parallel \mathcal{L}_\mathrm{m}(G_2)$.

Sequential behaviours and ω-languages.[KGM92, TW94b, TW94a] In order to describe and analyze the sequential (also called infinite) behaviour of DES, the notion of infinite-length, so-called ω-strings is useful. The set of *ω-strings* over $\Sigma \subseteq \Sigma$ is denoted

$$\Sigma^\omega = \{s | \forall\ i \in \mathbb{N}_0 : \sigma_i \in \Sigma,\ s = \sigma_0\sigma_1\sigma_2\cdots\}.$$

If for two strings $w \in \Sigma^\omega$, $r \in \Sigma^*$, there exists $v \in \Sigma^\omega$ such that $w = rv$, we say r is a *strict prefix* of w and write $r < w$. The strict prefix of w with length $n \in \mathbb{N}_0$ is denoted w^n. An *ω-language* over Σ^ω is a subset $\mathcal{L} \subseteq \Sigma^\omega$. The *prefix* of an ω-language $\mathcal{L} \subseteq \Sigma^\omega$ is defined

$$\mathrm{pr}(\mathcal{L}) = \{r | \exists\, s \in \mathcal{L} : r < s\} \subseteq \Sigma^*.$$

For convenience, for $\mathrm{pr}(\mathcal{L})$ we adopt the notation $\overline{\mathcal{L}}$ from the domain Σ^*, i.e. for $\mathcal{L} \subseteq \Sigma^\omega$ we denote $\overline{\mathcal{L}} := \mathrm{pr}(\mathcal{L})$. For a language $\mathcal{L} \subseteq \Sigma^*$ the *limit* is defined

$$\mathcal{L}^\infty = \{w \in \Sigma^\omega | \exists\, (n_i)_{i\in\mathbb{N}_0}, n_{i+1} > n_i : w^{n_i} \in \mathcal{L}\}.^5$$

We define the *ω-languages represented by an automaton* $G := (Q, \Sigma, \delta, q_0, Q_m)$ as (cf. [KGM92])

$$\begin{aligned} \mathcal{L}^\infty(G) &:= (\mathcal{L}(G))^\infty &= \{s | \exists\, (n_i)_{i\in\mathbb{N}_0}, n_{i+1} > n_i : \delta(q_0, s^{n_i})!\} \text{ and} \\ \mathcal{L}^\infty_\mathrm{m}(G) &:= (\mathcal{L}_\mathrm{m}(G))^\infty &= \{s | \exists\, (n_i)_{i\in\mathbb{N}_0}, n_{i+1} > n_i : \delta(q_0, s^{n_i}) \in Q_m\}. \end{aligned}$$

It is easily verified that the limit operator is monotonic:

Lemma 2.1
Let $\mathcal{L}_1,\ \mathcal{L}_2$ be regular languages over Σ^*. Then:

$$\mathcal{L}_1 \subseteq \mathcal{L}_2 \Rightarrow {\mathcal{L}_1}^\infty \subseteq {\mathcal{L}_2}^\infty \tag{2.4}$$

□

Proof Pick an arbitrary string $w \in {\mathcal{L}_1}^\infty$. Hence $\exists\, (n_i)_{i\in\mathbb{N}_0}, n_{i+1} > n_i : w^{n_i} \in \mathcal{L}_1$. As $\mathcal{L}_1 \subseteq \mathcal{L}_2$, $\forall n_i :\ w^{n_i} \in \mathcal{L}_2$ and thus $w \in {\mathcal{L}_2}^\infty$. □

In general, the reverse direction is false: consider $\mathcal{L}_1 = a^* \cup \{b\}$, $\mathcal{L}_2 = a^* \cup c^*$, both over the alphabet $\Sigma = \{a, b, c\}$, where ${\mathcal{L}_1}^\infty \subseteq {\mathcal{L}_2}^\infty$ but $\mathcal{L}_1 \nsubseteq \mathcal{L}_2$.

The completeness property has a strong impact on the relation between a language and its limit. The following lemma states that a string that is prefix of a string in a complete language at the same time is a prefix of an infinite string in the limit of this language, i.e. contributes to this limit.

[5] Observe that $(n_i)_{i\in\mathbb{N}_0}$ denotes an *infinite* sequence defining n_i for *all* $i \in \mathbb{N}_0$.

Lemma 2.2
For a language $\mathcal{L} \subseteq \Sigma^*$, the following equivalence holds:

$$\mathcal{L} \text{ is complete } \Leftrightarrow \overline{\mathcal{L}} = \overline{\mathcal{L}^\infty}$$

□

Proof

"⇐": Pick any $s \in \mathcal{L} \subseteq \overline{\mathcal{L}}$. Thus, $s \in \overline{\mathcal{L}^\infty}$, i.e. there exists $w \in \Sigma^\omega$ such that $sw \in \mathcal{L}^\infty$. Note that w is an infinite sequence. Hence, there exists $\sigma \in \Sigma$ such that $s\sigma < sw$ and consequently $s\sigma \in \overline{\mathcal{L}^\infty}$. Thus, $s\sigma \in \overline{\mathcal{L}}$, i.e. $\mathcal{L}$ is complete.

"⇒": $\overline{\mathcal{L}} \supseteq \overline{\mathcal{L}^\infty}$ is obvious. We show $\overline{\mathcal{L}} \subseteq \overline{\mathcal{L}^\infty}$. Pick an arbitrary string $s_1 \in \overline{\mathcal{L}}$ and proceed with the following algorithm:

(i) $i = 1$

(ii) As $s_i \in \overline{\mathcal{L}}$ there exists $r_i \in \Sigma^*$ such that $s_i r_i \in \mathcal{L}$.

(iii) As $\mathcal{L}$ is complete, there exists σ_i such that $s_i r_i \sigma_i \in \overline{\mathcal{L}}$.

(iv) Save $s_{i+1} := s_i r_i \sigma_i$, set $i = i + 1$ and proceed with step (ii).

By n-wise iteration of the above algorithm, a sequence $s_1 r_1 < s_2 r_2 < \ldots < s_n r_n$ can be constructed, where $n \in \mathbb{N}$ is arbitrary. Thus, there exists an infinite string $w \in \Sigma^\omega$ with $w^n = s_n r_n \in \mathcal{L}$ for infinitely many $n \in \mathbb{N}$. Hence, $w \in \mathcal{L}^\infty$. As $s_1 < s_1 r_1 < w$, it holds that $s_1 \in \overline{\mathcal{L}^\infty}$ and thus $\overline{\mathcal{L}} \subseteq \overline{\mathcal{L}^\infty}$.

□

By the above lemma, the prefix-closure of a complete language equals the prefix of its limit. Hence, we receive the following statement, if the language is additionally prefix-closed.

Lemma 2.3 ([KGM92])
For a language $\mathcal{L} \subseteq \Sigma^*$, the following equivalence holds:

$$\mathcal{L} \text{ is complete and prefix-closed } \Leftrightarrow \mathcal{L} = \overline{\mathcal{L}^\infty}$$

□

The natural projection for ω-strings carries over from finite strings in a straightforward way, see Definition 2.6. The range, however, is the union of finite and ω-strings. In contrast, the set valued inverse projection maps ω-strings to ω-languages.

Definition 2.6
The *natural projection of infinite strings* $p_\mathrm{o}\colon \Sigma^\omega \to \Sigma_\mathrm{o}^* \cup \Sigma_\mathrm{o}^\omega$, $\Sigma_\mathrm{o} \subseteq \Sigma$, is defined:

$$\text{for } s = \sigma_1\sigma_2\sigma_3\cdots \in \Sigma^\omega :\ p_\mathrm{o}(s) := p_\mathrm{o}(\sigma_1)p_\mathrm{o}(\sigma_2)p_\mathrm{o}(\sigma_3)\cdots.$$

The set valued inverse $p_\mathrm{o}^{-1}\colon \Sigma_\mathrm{o}^* \cup \Sigma_\mathrm{o}^\omega \to 2^{\Sigma^* \cup \Sigma^\omega}$ is defined

$$p_\mathrm{o}^{-1}(s) := \{t \in \Sigma^* \cup \Sigma^\omega | p_\mathrm{o}(t) = s\} \text{ for } s \in \Sigma_\mathrm{o}^* \cup \Sigma_\mathrm{o}^\omega$$

□

The projection and its inverse are defined for ω-languages by

$$p_\mathrm{o}(\mathcal{L}) := \{p_\mathrm{o}(s) \in \Sigma_\mathrm{o}^* \cup \Sigma_\mathrm{o}^\omega | s \in \mathcal{L}\}$$

and

$$p_\mathrm{o}^{-1}(\mathcal{L}_o) := \{s \in \Sigma^* \cup \Sigma^\omega | p_\mathrm{o}(s) \in \mathcal{L}_o\}$$

for $\mathcal{L} \subseteq \Sigma^* \cup \Sigma^\omega$ and $\mathcal{L}_o \subseteq \Sigma_\mathrm{o}^* \cup \Sigma_\mathrm{o}^\omega$, respectively. Accordingly, the Definition 2.5 of the synchronous product is extended to ω-languages. For prefix-closed languages $\mathcal{L}_1$ and $\mathcal{L}_2$ we have

$$\mathcal{L}_1{}^\infty \parallel \mathcal{L}_2{}^\infty \subseteq (\mathcal{L}_1 \parallel \mathcal{L}_2)^\infty,$$

see Appendix, Lemma A.3, where equality does not hold, in general (see Appendix, Lemma A.4).

This chapter introduced the notation, terminology, representation and properties of formal languages to an amount that provides a technical basis for the input-output based description of discrete event systems in the following chapter.

Chapter 3

Discrete Event Systems with Inputs and Outputs

In systems theory and especially in control theory, a major interest lies in how a system is influenced by its surroundings and how, in return, the system influences its surroundings. The input-output-based (I/O-based) representation of systems as it is widely used in control theory evolved from that perception. In this chapter, an I/O-based view is introduced for discrete event systems. First, we extend the language-based description of discrete event systems (Section 3.1) by the notion of I/O ports in Section 3.2 to describe the interaction of a system with its surroundings via inputs and outputs. The I/O based representation of a plant model and its liveness properties under external influence in Sections 3.3 to 3.5 is followed by an I/O-based view of a controller (Section 3.6). The main ideas of this chapter have been published in [PMS06] and [PMS07a].

3.1 System Description

We use formal languages to describe the dynamical behaviour of a discrete-event process as a mathematical system model. In our framework, a *system* consists of an alphabet that carries the totality of all possible events and a language over that alphabet describing all possible event sequences.

Definition 3.1 (System)
A *system* is a tuple $\mathcal{S} = (\Sigma, \mathcal{L})$ with the alphabet Σ and the language $\mathcal{L} \subseteq \Sigma^*$. □

This definition strongly corresponds to Willems' definition of a *mathematical model* of a phenomenon.[1] Moreover, Definition 3.1 leaves room for a separate definition of the terms input and

[1]For discrete-event processes, the mathematical model according to Willems' definition could be chosen as $(\Sigma^*, \mathcal{L})$, with the universe Σ^*.

output.

We say the system complete if $\mathcal{L}$ is complete, the system is regular if $\mathcal{L}$ is regular, the system is prefix-closed if $\mathcal{L}$ is prefix-closed etc.

Remark 3.1
Our notion of liveness is not expressed by marked strings, and in this thesis, we consider prefix-closed systems only. Ongoing research includes the consideration of non-prefix-closed systems $\mathcal{S} = (\Sigma, \mathcal{L})$ with the instantaneous behaviour $\overline{\mathcal{L}}$ and the infinite (i.e. long-term) behaviour $\mathcal{L}^\infty$ (rather than $(\overline{\mathcal{L}})^\infty$). An according automaton representation generates $\overline{\mathcal{L}}$ and marks $\mathcal{L}$. Such system, if complete, additionally features the property of "eventuality" in that it provides a persistent guarantee *(rather than a chance) for strings of $\overline{\mathcal{L}}$ to be extended to a marked string of $\mathcal{L}$. This extended system description (including the prefix-closed case) augments the expressiveness of the system models and allows to specify a wider range of control tasks.* □

We introduce inputs and outputs for discrete event systems by the notion of I/O ports, via which systems perform interaction.

3.2 I/O Ports

The interaction of a plant model with its surroundings via input and output is described by the following notion of a plant-I/O port.

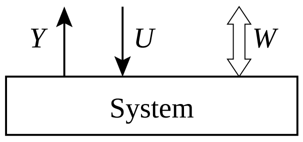


Figure 3.1: Plant-I/O port

Definition 3.2 (Plant-I/O Port)
A pair (U, Y) is a *plant-I/O port* of the system $(\Sigma, \mathcal{L})$ if

(i) $\Sigma = W \dot\cup U \dot\cup Y$, $U \neq \emptyset \neq Y$;

(ii) $\mathcal{L} \subseteq \overline{(YU + W)^*}$; and

(iii) $(\forall s \in \Sigma^* Y,\ \mu \in U)\ [\, s \in \mathcal{L} \Rightarrow s\mu \in \mathcal{L}\,]$.

□

By item (i), we separate input events $\mu \in U$ from output events $\nu \in Y$. Note that the partition into input and output alphabet does not coincide with the controllability of events: e.g. a sensor event may at the same time be an output of one system and an input of another system. By item (ii), we require alternation of output and input events aiming at a dependence between cause and effect. Remaining dynamics (e.g. dynamics performed on another I/O port) is captured by W^*. When the system *generates* some output event $\nu \in Y$ on the plant-I/O port it will *accept* any input event $\mu \in U$ as an immediate successor (item (iii)) respecting that the input can be imposed freely by the systems surroundings. Consistent with the definition, the incoming arrow in Figure 3.1 denotes that U is accepted, while the emanating arrow denotes that Y is generated.

The following definition of a controller-I/O port is complementary in the sense that it requires the system to accept any event $\nu \in Y$ as input and to reply by some event $\mu \in U$ as output, after an optional negotiation with some other system via the alphabet W; see Figure 3.2. A controller-I/O port can be connected with a plant-I/O port, see Proposition 3.1 below.

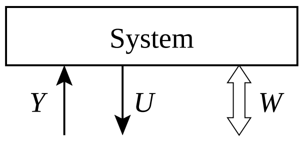


Figure 3.2: Controller-I/O port

Definition 3.3 (Controller-I/O Port)
A pair (U, Y) is a *controller-I/O port* of the system $(\Sigma, \mathcal{L})$ if

(i) $\Sigma = W \dot\cup U \dot\cup Y$, $U \neq \emptyset \neq Y$;

(ii) $\mathcal{L} \subseteq \overline{(YW^*U)^*}$; and

(iii) $(\,\forall s \in \Sigma^*U \cup \{\epsilon\},\ \nu \in Y\,)\,[\,s \in \mathcal{L} \Rightarrow s\nu \in \mathcal{L}\,]$.

□

The above notion of a I/O ports relates to Willems' description of I/O behaviours with free input and an output that does not anticipate the input. In contrast to e.g. [Wil91], we do not require the output to process the input and thereby account for non-deterministic external behaviour.

A controller-I/O port of one system can be connected with a plant-I/O of another system port to achieve a simple feedback structure as in Figure 1.10 b) that preserves completeness:

Proposition 3.1
Let (U, Y) be a plant-I/O port of the complete system $\mathcal{S}_1 = (\Sigma, \mathcal{L}_1)$, and let (U, Y) be a controller-I/O port of the complete system $\mathcal{S}_2 = (\Sigma, \mathcal{L}_2)$ with $\Sigma = U \cup Y$.

Then the feedback structure $\mathcal{S}_{1,2} = (\Sigma, \mathcal{L}_1 \parallel \mathcal{L}_2)$ is a complete system. □

Proof Note that $\mathcal{L}_1$ and $\mathcal{L}_2$ are languages over the same alphabet Σ, and the composition $\mathcal{L}_{1,2} := \mathcal{L}_1 \parallel \mathcal{L}_2$ evaluates to $\mathcal{L}_1 \cap \mathcal{L}_2$. Hence, $\mathcal{S}_{1,2}$ is complete whenever $\mathcal{L}_1 = \emptyset$ or $\mathcal{L}_2 = \emptyset$. Now consider

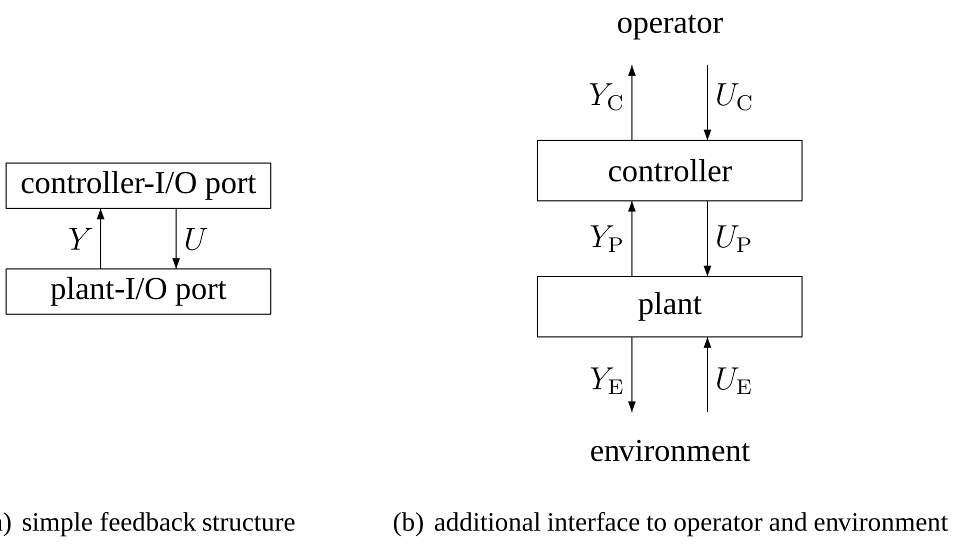
$\mathcal{L}_1 \neq \emptyset \neq \mathcal{L}_2$ and pick an arbitrary string $s \in \mathcal{L}_{1,2}$. Observe that $s \in \mathcal{L}_1$, $s \in \mathcal{L}_2$ and that the language format $\overline{(YU)^*}$ is met by both, $\mathcal{L}_1$ and $\mathcal{L}_2$. We distinguish the following possible cases:

(i) $s = \epsilon$ or $s = s'\mu,\ \mu \in U$: as $\mathcal{S}_1$ is complete and due to its language structure, $s\sigma \in \mathcal{L}_1$ for some $\sigma \in Y$. Also, $s\nu \in \mathcal{L}_2\ \forall \nu \in Y$, as (U, Y) is a controller-I/O port of $\mathcal{S}_2$. In particular, $s\sigma \in \mathcal{L}_2$. Hence $s\sigma \in \mathcal{L}_{1,2}$.

(ii) $s = s'\nu,\ \nu \in Y$: then, as $\mathcal{S}_2$ is complete and due to its language structure, $s\sigma \in \mathcal{L}_2$ for some $\sigma \in U$. Also, $s\mu \in \mathcal{L}_1\ \forall \mu \in U$, as (U, Y) is a plant-I/O port of $\mathcal{S}_1$. In particular, $s\sigma \in \mathcal{L}_1$. Hence $s\sigma \in \mathcal{L}_{1,2}$.

Together, for all $s \in \mathcal{L}_{1,2}$ it holds that there exists $\sigma \in \Sigma$ such that $s\sigma \in \mathcal{L}_{1,2}$, i.e. $\mathcal{L}_{1,2}$ is complete and thus $\mathcal{S}_{1,2}$ is complete. □

The setting of Proposition 3.1 already enables the restriction of some physical plant behaviour given as a plant I/O port using a controller I/O port that exhibits actuating events in reaction to measurement events, see Figure 3.3 a). However, a standard control loop is usually equipped with an interface to some operator (e.g. by the reference variable) and a variable describing the effect of control to the environment of the plant (e.g. the control variable that is usually a system output). Based on this consideration, we aim at a control loop that extends the simple feedback structure by an interface of the controller to the operator (index C) and an interface of the plant to the environment (index E), see Figure 3.3 b).



(a) simple feedback structure (b) additional interface to operator and environment

Figure 3.3: from simple feedback to control loop

3.3 I/O Plant

In order to achieve a high degree of modularity in our approach, we aim for a plant model description that is reusable within various configurations. To this end, we explicitly separate the plant model from its surroundings, which we identify as an operator and an environment. From the perspective of the operator, the plant models the mechanism by which the environment can be manipulated. Hence, an I/O plant is defined as a system equipped with two distinguished plant-I/O ports, see Figure 3.4 [2]. One port models the interaction of the plant with an operator (or controller) via events Σ_{P}, the other port models the interaction of the plant with the environment via the events Σ_{E} that are not directly observable to the operator (or controller).

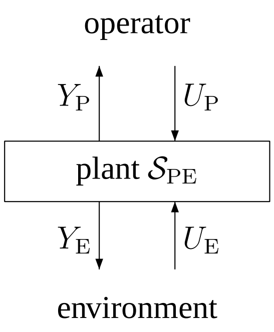


Figure 3.4: I/O plant

Definition 3.4 (I/O Plant)
An *I/O plant* is a tuple $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$, where

(i) $(\Sigma_{\mathrm{PE}}, \mathcal{L}_{\mathrm{PE}})$ is a system with $\Sigma_{\mathrm{PE}} := \Sigma_{\mathrm{P}} \dot{\cup} \Sigma_{\mathrm{E}}$, $\Sigma_{\mathrm{P}} := U_{\mathrm{P}} \dot{\cup} Y_{\mathrm{P}}$, $\Sigma_{\mathrm{E}} := U_{\mathrm{E}} \dot{\cup} Y_{\mathrm{E}}$; and

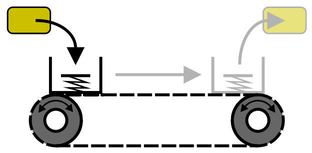
(ii) $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ and $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ are plant-I/O ports of $(\Sigma_{\mathrm{PE}}, \mathcal{L}_{\mathrm{PE}})$.

□

Note that an I/O plant always possesses the language format $\mathcal{L}_{\mathrm{PE}} \subseteq \overline{(Y_{\mathrm{P}} U_{\mathrm{P}} + Y_{\mathrm{E}} U_{\mathrm{E}})^*}$. To illustrate the above definition we introduce the following conceptional example.

[2] In this thesis the relationship between systems, alphabets and languages is consequently indicated by matching subscripts; e.g. the system $\mathcal{S}_{\mathrm{ABC}}$ always refers to the language $\mathcal{L}_{\mathrm{ABC}}$ over the alphabet Σ_{ABC}. Furthermore, Σ_{ABC} denotes the disjoint union of Σ_{A}, Σ_{B} and Σ_{C}, and when inputs and outputs are relevant we use e.g. $\Sigma_{\mathrm{A}} = U_{\mathrm{A}} \dot{\cup} Y_{\mathrm{A}}$. Similarly, the natural projection to Σ_{AB}^* is denoted p_{AB}; the natural projection to Y_{A}^* is denoted p_{YA}.

Example 3.1
Transport Unit. Consider a simple transport unit (TU) as depicted in Figure 3.5 a). Its behaviour can be modeled as an I/O plant $\mathcal{S}_{\mathrm{PE}} := (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ with $\mathcal{L}_{\mathrm{PE}}$ marked by the corresponding automaton model depicted in Figure 3.5 b). $\mathcal{S}_{\mathrm{PE}}$ is a prefix-closed system and hence, all states are marked, which is denoted by double-lined circles.


(a) physical layout

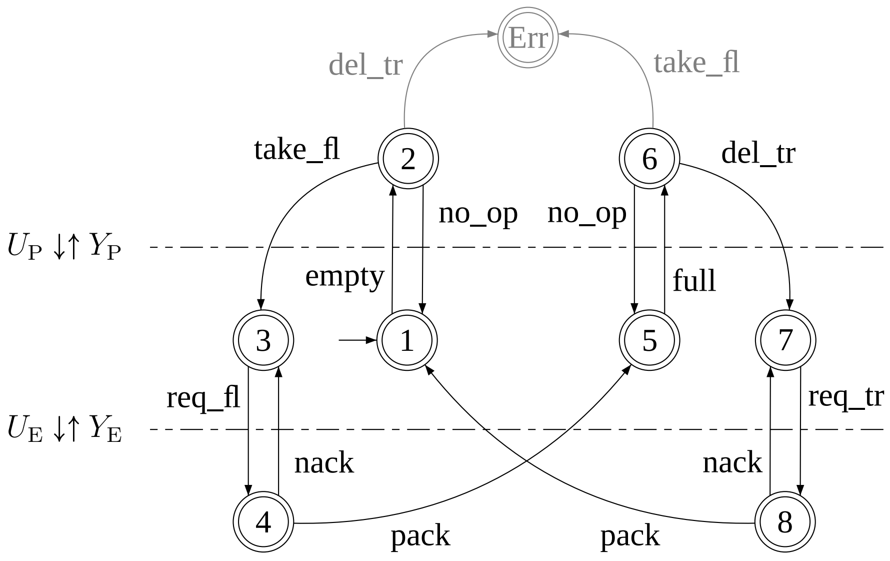

Err
del_tr
take_fl
take_fl
2
6
del_tr
no_op
no_op
$U_{\mathrm{P}} \downarrow\uparrow Y_{\mathrm{P}}$
empty
full
3
1
5
7
req_fl
req_tr
$U_{\mathrm{E}} \downarrow\uparrow Y_{\mathrm{E}}$
nack
nack
4
8
pack
pack
(b) I/O plant model

Figure 3.5: Conceptional example: Transport Unit

The TU consists of a conveyor belt carrying a box that can hold the workpiece to be transported. A spring sensor inside the box detects the absence or presence of a workpiece ($empty$, $full$). The initial state (state 1 in Fig. 3.5 b)) is defined such that the sensor reports $empty$. The operator can choose between three different commands (state 2). After the no_op (no operation) command, the TU does not move, and the system remains in the initial state. The command del_tr (deliver to right) leads to an error state as there is currently no workpiece present to deliver. Choosing the command $take\_fl$ (take from left) prompts the TU to move the box to its left border (state 3). Now it depends on the environment if a workpiece is provided from the left, which is modeled by the event req_fl unobservable to the operator. For a plant description that is independent from the environment, we introduce the environment-events $pack$ and $nack$ (positive/negative acknowledge) respecting that the environment may or may not comply with the requests of the plant. If the environment is not in the condition to provide a workpiece ($nack$), the request is repeated. When a workpiece is provided from the environment, the sensor reports $full$. Now (state 6), the command $take\_fl$ leads to an error behaviour (the box can carry only one workpiece), and after no_op the plant still reports $full$. By the command del_tr, the belt moves the box to the right border. The event req_tr models the need for the workpiece to be withdrawn to the right by the environment. In case of $pack$, the system returns to its initial state. By $(U_{\mathrm{P}}, Y_{\mathrm{P}}) := (\{no\_op, take\_fl, del\_tr\}, \{empty, full\})$, we identify the interaction with the operator, $(U_{\mathrm{E}}, Y_{\mathrm{E}}) := (\{pack, nack\}, \{req\_fl, req\_tr\})$ describes interaction with the environment. Note that $(U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ features all I/O-plant properties of Definition 3.4. □

Clearly, an automaton $G = (\Sigma, Q, \delta, q_0, Q_{\mathrm{m}})$ that represents an I/O plant must itself have a certain structure such as an I/O plant has, see e.g. Figure 3.5 b). The knowledge of this structure is helpful e.g. for the graph-based design of an I/O plant model, the graph-based test for I/O plant properties or a structured graph-based visualization of an I/O plant (like the hierarchical arrangement of the states in Figure 3.5 b)). The following definition provides such an automata structure that corresponds to an I/O plant.

Definition 3.5 (I/O-Plant Form)
A generator $G := (\Sigma, Q, \delta, q_0, Q_{\mathrm{m}})$ is in *I/O-plant form* if

(i) $\Sigma = U_{\mathrm{P}} \dot\cup Y_{\mathrm{P}} \dot\cup U_{\mathrm{E}} \dot\cup Y_{\mathrm{E}}$ with nonempty alphabets $U_{\mathrm{P}},\ Y_{\mathrm{P}},\ U_{\mathrm{E}},\ Y_{\mathrm{E}}$

(ii) $Q = Q_{\mathrm{Y}} \dot\cup Q_{\mathrm{UP}} \dot\cup Q_{\mathrm{UE}}$

(iii) $q_0 \in Q_{\mathrm{Y}}$

(iv) $[\forall q \in Q_{\mathrm{Y}}, \sigma \in \Sigma](\delta(q,\sigma)! \Rightarrow (\sigma \in Y_{\mathrm{P}} \wedge \delta(q,\sigma) \in Q_{\mathrm{UP}}) \vee (\sigma \in Y_{\mathrm{E}} \wedge \delta(q,\sigma) \in Q_{\mathrm{UE}}))$

(v) $[\forall q \in Q_{\mathrm{UP}}, \sigma \in \Sigma](\delta(q,\sigma)! \Rightarrow (\sigma \in U_{\mathrm{P}} \wedge \delta(q,\sigma) \in Q_{\mathrm{Y}}))$

(vi) $[\forall q \in Q_{\mathrm{UE}}, \sigma \in \Sigma](\delta(q,\sigma)! \Rightarrow (\sigma \in U_{\mathrm{E}} \wedge \delta(q,\sigma) \in Q_{\mathrm{Y}}))$

(vii) $[\forall q \in Q_{\mathrm{UP}}, \mu \in U_{\mathrm{P}}](\delta(q,\sigma)!)$

(viii) $[\forall q \in Q_{\mathrm{UE}}, \mu \in U_{\mathrm{E}}](\delta(q,\sigma)!)$

(ix) $Q_{\mathrm{m}} = Q$

(x) G is accessible.

□

Remark 3.2
Property (ix) guarantees that G represents a prefix-closed system. Properties (ix) and (x) imply that G is trim. □

Lemma 3.1
If a generator $G := (\Sigma, Q, \delta, q_0, Q_{\mathrm{m}})$ is in I/O-plant form, then the system $(\Sigma, \mathcal{L}_{\mathrm{m}}(G))$ is an I/O plant. □

Proof Preliminary note: Note that property (ix) implies $q_0 \in Q_\mathrm{m}$. Hence, $\mathcal{L}_\mathrm{m}(G) \neq \emptyset$. We now prove that $(\Sigma, \mathcal{L}_\mathrm{m}(G))$ provides all I/O-plant properties.

(i) $(\Sigma, \mathcal{L}_\mathrm{m}(G))$ is a system: by definition, G recognizes the language $\mathcal{L}_\mathrm{m}(G)$ over Σ. Property (i) requires $\Sigma = U_\mathrm{P} \dot\cup Y_\mathrm{P} \dot\cup U_\mathrm{E} \dot\cup Y_\mathrm{E}$, and we identify $\Sigma_\mathrm{PE} = \Sigma_\mathrm{P} \dot\cup \Sigma_\mathrm{E} := \Sigma$ with $\Sigma_\mathrm{P} := U_\mathrm{P} \dot\cup Y_\mathrm{P}$ and $\Sigma_\mathrm{E} := U_\mathrm{E} \dot\cup Y_\mathrm{E}$.

(ii) $(U_\mathrm{P}, Y_\mathrm{P})$ and $(U_\mathrm{E}, Y_\mathrm{E})$ are plant-I/O ports of $(\Sigma, \mathcal{L}_\mathrm{m}(G))$. Proof: we show that $(U_\mathrm{P}, Y_\mathrm{P})$ provides all plant-I/O port properties. The plant-I/O port property of $(U_\mathrm{P}, Y_\mathrm{P})$ carries over to $(U_\mathrm{E}, Y_\mathrm{E})$ by uniform substitution.

(ii.i) From property (i) in Definition 3.5 we directly conclude $\Sigma = W \dot\cup U_\mathrm{P} \dot\cup Y_\mathrm{P}$ (with $W = \Sigma - U_\mathrm{P} - Y_\mathrm{P} = U_\mathrm{E} \dot\cup Y_\mathrm{E}$) and $U_\mathrm{P} \neq \emptyset \neq Y_\mathrm{P}$.

(ii.ii) $\mathcal{L}_\mathrm{m}(G) \subseteq \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$ with $W^* = (Y_\mathrm{E}^* U_\mathrm{E}^*)^*$. Proof: If $\mathcal{L}_\mathrm{m}(G) = \{\epsilon\}$, obviously $\mathcal{L}_\mathrm{m}(G) \subseteq \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$. For $\mathcal{L}_\mathrm{m}(G) \supset \{\epsilon\}$, we continue with induction: Pick arbitrary $\sigma \in \mathcal{L}_\mathrm{m}(G) \cap \Sigma$. Hence, $\delta(q_0, \sigma)!$. As property (iii) requires $q_0 \in Q_\mathrm{Y}$, property (iv) implies $\sigma \in Y_\mathrm{P}$ or $\sigma \in Y_\mathrm{E}$. Both cases prove $\sigma \in \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$.
Now consider a nonempty string $s\sigma_{n+1} = \sigma_1\sigma_2 \ldots \sigma_n\sigma_{n+1},\ \sigma_i \in \Sigma,\ i = 1..n,\ n \in \mathbb{N}$ with $s\sigma_{n+1} \in \mathcal{L}_\mathrm{m}(G)$. Assume $s \in \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$. We show that $s\sigma_{n+1} \in \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$. Note that there exists some $q \in Q$ such that $\delta(q, \sigma_n)!$ and $\delta(q, \sigma_n\sigma_{n+1})!$ and distinguish two cases:

(a) $\sigma_n \in Y_\mathrm{E} \cup Y_\mathrm{P}$. In this case, properties (v) and (vi) rule out $q \in Q_\mathrm{UP} \cup Q_\mathrm{UE}$. Because of property (ii), we can conclude $q \in Q_\mathrm{Y}$. If $\sigma_n \in Y_\mathrm{E}$, property (iv) requires $\delta(q, \sigma_n) \in Q_\mathrm{UE}$. Consequently, property (vi) implies $\sigma_{n+1} \in U_\mathrm{E}$. Hence, $s\sigma_{n+1} \in \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*} Y_\mathrm{E} U_\mathrm{E} \subseteq \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$.
If $\sigma_n \in Y_\mathrm{P}$, property (iv) requires $\delta(q, \sigma_n) \in Q_\mathrm{UP}$. Consequently, property (vii) implies $\sigma_{n+1} \in U_\mathrm{P}$. Hence, $s\sigma_{n+1} \in \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*} Y_\mathrm{P} U_\mathrm{P} \subseteq \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$.

(b) $\sigma_n \in U_\mathrm{P} \cup U_\mathrm{E}$. Then, property (iv) rules out $q \in Q_\mathrm{Y}$. Because of property (ii), we can conclude $q \in Q_\mathrm{UP} \cup Q_\mathrm{UE}$. Hence, properties (v) and (vi) imply $\delta(q, \sigma_n) \in Q_\mathrm{Y}$. Consequently, $\sigma_{n+1} \in Y_\mathrm{E} \cup Y_\mathrm{P}$ follows from property (iv). Hence, $s\sigma_{n+1} \in [\overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}(U_\mathrm{P} \vee U_\mathrm{E}) \cap \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}](Y_\mathrm{P} \vee Y_\mathrm{E}) \subseteq \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$.

Thus, $s\sigma_{n+1} \in \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$ whenever $s \in \overline{(W^*(Y_\mathrm{P}U_\mathrm{P})^*)^*}$, which proves the induction step.

(ii.iii) $(\forall s \in \Sigma^* Y_\mathrm{P}, \mu \in U_\mathrm{P})[s \in \mathcal{L}_\mathrm{m}(G) \Rightarrow s\mu \in \mathcal{L}_\mathrm{m}(G)]$. Proof:
Pick arbitrary $s\nu \in \mathcal{L}_\mathrm{m}(G),\ \nu \in Y_\mathrm{P}$. Write $q := \delta(q_0, s)$ and observe $\delta(q, \nu)!$. As $\nu \notin U_\mathrm{P} \cup U_\mathrm{E}$, properties (v) and (vi) rule out $q \in Q_\mathrm{UP} \cup Q_\mathrm{UE}$. Because of property (ii), $q \in Q_\mathrm{Y}$. Thus, as $\nu \in Y_\mathrm{P}$, property (iv) implies that $q' := \delta(q, \nu) \in Q_\mathrm{UP}$. Consequently, property (vii) implies that for all $\mu \in U_\mathrm{P}$ it holds that $\delta(q', \mu)!$. Hence, $s\nu\mu \in \mathcal{L}_\mathrm{m}(G)$ for all $\mu \in U_\mathrm{P}$.

Thus, $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ and $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ are plant-I/O ports of $(\Sigma, \mathcal{L}_{\mathrm{m}}(G))$.

Consequently, $(\Sigma, \mathcal{L}_{\mathrm{m}}(G))$ is an I/O plant. □

With a given plant model, we now can approach the problem of controller design. In the field of discrete event systems, the usual design objectives are compliance with a desired behaviour that is expressed as a safety specification and compliance with certain liveness properties such that the desired behaviour is not only passively complied with but also exhibited actively. Safety properties can be expressed as a language inclusion, whereas the liveness properties of the plant strongly depend on its actual external configuration. For the discussion of the plant in a variety of different external configurations, we introduce the notion of constraints.

3.4 Constraints

Considering its definition, an I/O plant may be subject to constraints on the operator and/or the environment; e.g. the operator may or may not comply to the operator's guidelines and the environment may or may not provide resources.

Example 3.2
Transport Unit. Consider the transport unit that allows for transportation of workpieces. Its ability to *continually* transport workpieces depends a) on the operator, as he has to operate the events del_tr and $take\_fl$ in a reasonable order, and b) on the environment, as it has to provide or accept workpieces from time to time. □

In this framework, we describe those constraints as the variety of controller-I/O ports that can be connected to the I/O plant to obtain the desired liveness properties.

Definition 3.6 (Constraint)
A *constraint* is a tuple $(U, Y, \mathcal{L})$ if

(i) $(\Sigma, \mathcal{L})$ is a system with $\Sigma = U \dot{\cup} Y$;

(ii) (U, Y) is a controller-I/O port of $(\Sigma, \mathcal{L})$;

(iii) $\mathcal{L}$ is complete.

□

By item (iii), we rule out constraints that preclude liveness of any I/O plant under such constraint. We refer to the *minimal constraint* $(U, Y, \mathcal{L})$ with $\mathcal{L} = \overline{(YU)^*}$, if actually *no* constraint is considered, and the *maximal constraint* $(U, Y, \mathcal{L})$ with $\mathcal{L} = \emptyset$. The operator and the environment constraint are denoted $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$ and $\mathcal{S}_{\mathrm{E}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{E}})$, respectively.

The following definition provides an automata structure that corresponds to a constraint.

Definition 3.7 (Constraint Form)
A generator $G := (\Sigma, Q, \delta, q_0, Q_\mathrm{m})$ is in *constraint form* if

(i) $\Sigma = U \dot{\cup} Y$ with nonempty alphabets U, Y

(ii) $Q = Q_\mathrm{Y} \dot{\cup} Q_\mathrm{U}$

(iii) $q_0 \in Q_\mathrm{Y}$

(iv) $[\forall q \in Q_\mathrm{Y}, \sigma \in \Sigma](\delta(q, \sigma)! \Rightarrow (\sigma \in Y \wedge \delta(q, \sigma) \in Q_\mathrm{U})$

(v) $[\forall q \in Q_\mathrm{U}, \sigma \in \Sigma](\delta(q, \sigma)! \Rightarrow (\sigma \in U \wedge \delta(q, \sigma) \in Q_\mathrm{Y}))$

(vi) $[\forall q \in Q_\mathrm{Y}, \nu \in Y](\delta(q, \sigma)!)$

(vii) $[\forall q \in Q](\exists \sigma \in \Sigma : \delta(q, \sigma)!)$

(viii) $Q_\mathrm{m} = Q$

(ix) G is accessible.

□

Lemma 3.2
If a generator $G := (\Sigma, Q, \delta, q_0, Q_\mathrm{m})$ is in constraint form, then the system $(\Sigma, \mathcal{L}_\mathrm{m}(G))$ is a constraint. □

Proof See Appendix A.2. □

In our framework, the notion of liveness is consistently formulated as liveness under constraints.

3.5 Liveness

A majority of the approaches to control of DES that regard liveness use the technique of marking particular strings of plant and/or specification to express desired liveness properties. In most of these approaches, the respective objective of controller design is to achieve or preserve the permanent chance for any string of the closed loop to be extended to a marked string.

Our notion of liveness is different in that it requires output events to occur persistently rather than strings/states to be reachable and thus is not based on marking. First, we define the notion of a Y_P-live language and then identify two coupled liveness properties adequate for our setting.

Definition 3.8 (Y_{P}-Liveness)
Let $\mathcal{L}$ be a regular language and Y_{P} be an alphabet. If

$$(\,\forall w \in \mathcal{L}^{\infty}\,)[\,p_{\mathrm{YP}}(w) \in Y_{\mathrm{P}}^{\omega}\,],$$

then $\mathcal{L}$ is said to be Y_{P}-*live*.[3] □

Apparently, any subset of a Y_{P}-live language is also Y_{P}-live:

Lemma 3.3
Let $\mathcal{L}$ be a regular language and Y_{P} be an alphabet. If $\mathcal{L}$ is Y_{P}-live, then so is any sublanguage of $\mathcal{L}$. □

Proof Let $\mathcal{K}$ be an arbitrary sublanguage of $\mathcal{L}$. Because of Lemma 2.1, it holds that $\mathcal{K}^{\infty} \subseteq \mathcal{L}^{\infty}$. Thus, for all $w \in \mathcal{K}^{\infty}$ we have $w \in \mathcal{L}^{\infty}$ and, as $\mathcal{L}$ is Y_{P}-live, it holds that $p_{\mathrm{YP}}(w) \in Y_{\mathrm{P}}^{\omega}$. Consequently, $\mathcal{K}$ is Y_{P}-live.

□

The above statement is important for a) discussing the liveness of an I/O plant within different external configurations (modularity) and b) abstraction-based control. Using Y_{P}-liveness, we can describe the liveness properties of an I/O plant as follows.

Definition 3.9 (I/O Plant: Liveness Properties)
Let $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be an I/O plant and let $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$ and $\mathcal{S}_{\mathrm{E}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{E}})$ be constraints.

(i) If $\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ is complete, then $\mathcal{S}_{\mathrm{PE}}$ said to be *complete w.r.t. the constraints* $\mathcal{S}_{\mathrm{P}}$ *and* $\mathcal{S}_{\mathrm{E}}$.

(ii) If

$$(\,\forall w \in (\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})^{\infty}\,)[\,p_{\mathrm{YP}}(w) \in Y_{\mathrm{P}}^{\omega}\,],^{4} \tag{3.1}$$

then the plant said to be Y_{P}-*live w.r.t. the constraints* $\mathcal{S}_{\mathrm{P}}$ *and* $\mathcal{S}_{\mathrm{E}}$.

□

Completeness requires the plant to persistently issue events, i.e. prohibits deadlocks. Moreover, the completeness property guarantees that each sequence of events contributes to an *infinite* sequence of events in the language limit considered by Y_{P}-liveness, see Proposition 2.2. The second liveness property, Y_{P}-liveness, requires that any infinite sequence of events must include an infinite number of measurement events reported to the operator (no livelocks between any two Y_{P}-events). Hence, properties (i) and (ii) when put together indeed *guarantee* that an infinite sequence of measurement events $\nu \in Y_{\mathrm{P}}$ is generated by the plant under constraints and, in return, influence by the operator is persistently possible. Technically, $\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} = \varnothing$ provokes both liveness conditions to be met trivially.

[3]If different alphabets such as Y_{P2} or Y_{C} are concerned, we speak of Y_{P2}- or Y_{C}-liveness, respectively.

[4]i.e. if $\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ is Y_{P}-live

Example 3.3
Transport Unit. Temporarily, assume minimal (i.e. no) constraints $\mathcal{S}_{\mathrm{P}min}$ and $\mathcal{S}_{\mathrm{E}min}$ for the model of the TU which corresponds to arbitrary external configurations. Note that $\mathcal{S}_{\mathrm{PE}}$ is neither complete nor Y_{P}-live with respect to these constraints. As seen in Figure 3.5 b), completeness is violated in the $error$ state because no further event is possible. Obviously, this *deadlock* is avoided by any operator that meets a constraint $\mathcal{S}_{\mathrm{P}}$ on the correct alternation of the commands $take\_fl$ and del_tr, see Figure 3.6.

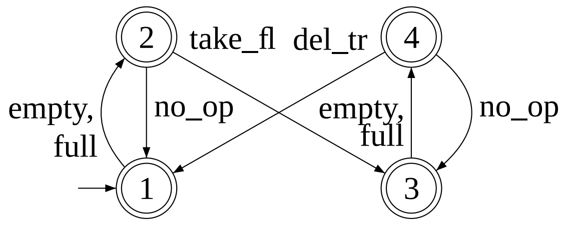


Figure 3.6: Operator constraint for the TU

Moreover, as the TU plant model is designed independently of the environment, the extremal case that the environment *never* complies with requests of the plant is included in the model. The resulting *livelock* violates the Y_{P}-liveness and is represented by a $(req\_fl\ nack)$-loop between states 3 and 4 and a $(req\_tr\ nack)$-loop between states 7 and 8 in Figure 3.5 b).

The environment constraint $\mathcal{S}_{\mathrm{E}} := (\Sigma_{\mathrm{E}}, \overline{((req\_fl + req\_tr)\ pack)^*})$ models the prohibition of the event $nack$, i.e., the assumption that requests of the plant are *always* accepted by the environment; see Figure 3.7.

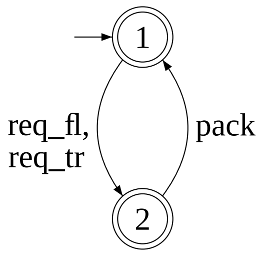


Figure 3.7: Environment constraint for the TU

The liveness properties of the plant are preserved if a controller connected to the plant complies with the operator constraint and if the external configuration meets the environment constraint, see Proposition 3.2. The resulting behaviour of the TU under the chosen constraints is shown in Figure 3.8.

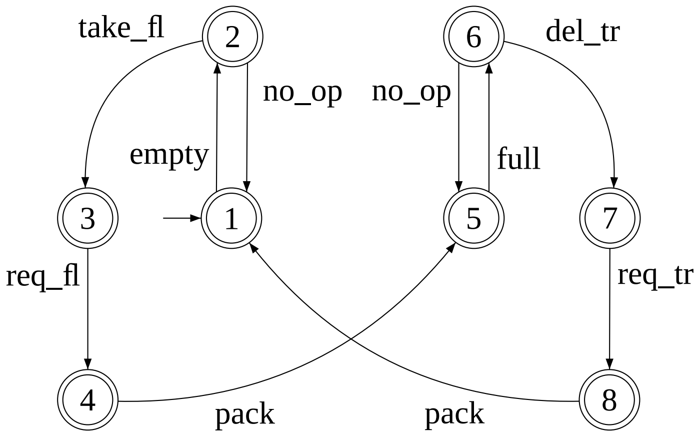


Figure 3.8: Transport Unit under constraints

As can be seen, the TU is complete and Y_P-live w.r.t. the chosen constraints, as a) there is no deadlock state (state with empty active event set) and b) a state is never visited twice unless at least one Y_P-event occurs. □

The following proposition shows that constraints can indeed be used as *conditions* for liveness of an I/O plant, as its liveness is guaranteed whenever its surroundings pose a subset of the constraints. Note that this result is a consequence of the I/O structure and the Y_P-liveness-property that is preserved in any subset.

Proposition 3.2
Let $\mathcal{S}_\mathrm{PE} = (U_\mathrm{P}, Y_\mathrm{P}, U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{PE})$ be an I/O plant that is complete and Y_P-live w.r.t. the constraints $\mathcal{S}_\mathrm{P} = (U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_\mathrm{P})$ and $\mathcal{S}_\mathrm{E} = (U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{E})$.
Then, $\mathcal{S}_\mathrm{PE}$ is complete and Y_P-live w.r.t. any constraints $\tilde{\mathcal{S}}_\mathrm{P} = (U_\mathrm{P}, Y_\mathrm{P}, \tilde{\mathcal{L}}_\mathrm{P})$ and $\tilde{\mathcal{S}}_\mathrm{E} = (U_\mathrm{E}, Y_\mathrm{E}, \tilde{\mathcal{L}}_\mathrm{E})$ with $\tilde{\mathcal{L}}_\mathrm{P} \subseteq \mathcal{L}_\mathrm{P}$ and $\tilde{\mathcal{L}}_\mathrm{E} \subseteq \mathcal{L}_\mathrm{E}$. □

Proof We show that $\tilde{\mathcal{L}}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \tilde{\mathcal{L}}_\mathrm{E}$ is complete and Y_P-live. Recall that, technically $\emptyset$ is complete and Y_P-live. Now, we consider $\tilde{\mathcal{L}}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \tilde{\mathcal{L}}_\mathrm{E} \neq \emptyset$. Observe that $\tilde{\mathcal{L}}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \tilde{\mathcal{L}}_\mathrm{E} \subseteq \mathcal{L}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E} \subseteq \mathcal{L}_\mathrm{PE}$.

Completeness. Pick an arbitrary string $s \in \tilde{\mathcal{L}}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \tilde{\mathcal{L}}_\mathrm{E}$, i.e. $s \in \mathcal{L}_\mathrm{PE}$, $p_\mathrm{P}(s) \in \tilde{\mathcal{L}}_\mathrm{P}$ and $p_\mathrm{E}(s) \in \tilde{\mathcal{L}}_\mathrm{E}$. Observing the language format $\tilde{\mathcal{L}}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \tilde{\mathcal{L}}_\mathrm{E} \subseteq \overline{(Y_\mathrm{P} U_\mathrm{P} + Y_\mathrm{E} U_\mathrm{E})^*}$ we distinguish:

- $s = \epsilon$, or $s = s'\mu$ with $\mu \in U_\mathrm{E} \cup U_\mathrm{P}$:

  As $\mathcal{L}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}$ is complete, there exists $\sigma \in Y_\mathrm{P} \cup Y_\mathrm{E}$ such that $s\sigma \in \mathcal{L}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}$.

  If $\sigma \in Y_\mathrm{P}$, then $p_\mathrm{E}(s\sigma) = p_\mathrm{E}(s) \in \tilde{\mathcal{L}}_\mathrm{E}$. For $p_\mathrm{P}(s)$, it holds that either $p_\mathrm{P}(s) = \epsilon$ or $p_\mathrm{P}(s) = s''\mu_\mathrm{P}$ with $\mu_\mathrm{P} \in U_\mathrm{P}$ because of the language format of $\mathcal{L}_\mathrm{PE}$. As $(U_\mathrm{P}, Y_\mathrm{P})$ is a controller-I/O port of $\tilde{\mathcal{S}}_\mathrm{P}$, it holds that $p_\mathrm{P}(s)\nu \in \tilde{\mathcal{L}}_\mathrm{P}\ \forall \nu \in Y_\mathrm{P}$. In particular, $p_\mathrm{P}(s)\sigma \in \tilde{\mathcal{L}}_\mathrm{P}$.

Analogously: If $\sigma \in Y_{\mathrm{E}}$, then $p_{\mathrm{P}}(s\sigma) = p_{\mathrm{P}}(s) \in \tilde{\mathcal{L}}_{\mathrm{P}}$. For $p_{\mathrm{E}}(s)$, it holds that either $p_{\mathrm{E}}(s) = \epsilon$ or $p_{\mathrm{E}}(s) = s''\mu_{\mathrm{E}}$ with $\mu_{\mathrm{E}} \in U_{\mathrm{E}}$ because of the language format of $\mathcal{L}_{\mathrm{PE}}$. As $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ is a controller-I/O port of $\tilde{\mathcal{S}}_{\mathrm{E}}$, it holds that $p_{\mathrm{E}}(s)\nu \in \tilde{\mathcal{L}}_{\mathrm{E}}\ \forall \nu \in Y_{\mathrm{E}}$. In particular, $p_{\mathrm{E}}(s)\sigma \in \tilde{\mathcal{L}}_{\mathrm{E}}$.

Together, $s\sigma \in \tilde{\mathcal{L}}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \tilde{\mathcal{L}}_{\mathrm{E}}$.

- $s = s'\nu$ with $\nu \in Y_{\mathrm{P}}$: As $p_{\mathrm{P}}(s) = p_{\mathrm{P}}(s')\nu \in \tilde{\mathcal{L}}_{\mathrm{P}}$ and $\tilde{\mathcal{L}}_{\mathrm{P}}$ is complete per definition of I/O constraints, there exists $\sigma \in U_{\mathrm{P}}$ such that $p_{\mathrm{P}}(s)\sigma$ in $\tilde{\mathcal{L}}_{\mathrm{P}}$. As $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ is a plant-I/O port of $\mathcal{S}_{\mathrm{PE}}$, it holds that $s\mu \in \mathcal{L}_{\mathrm{PE}} \forall \mu \in U_{\mathrm{P}}$. In particular, $s\sigma \in \mathcal{L}_{\mathrm{PE}}$. Moreover, $p_{\mathrm{E}}(s\sigma) = p_{\mathrm{E}}(s) \in \tilde{\mathcal{L}}_{\mathrm{E}}$. Together, $s\sigma \in \tilde{\mathcal{L}}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \tilde{\mathcal{L}}_{\mathrm{E}}$.

- *Analogously:* $s = s'\nu$ with $\nu \in Y_{\mathrm{E}}$: As $p_{\mathrm{E}}(s) = p_{\mathrm{E}}(s')\nu \in \tilde{\mathcal{L}}_{\mathrm{E}}$ and $\tilde{\mathcal{L}}_{\mathrm{E}}$ is complete per definition of I/O constraints, there exists $\sigma \in U_{\mathrm{E}}$ such that $p_{\mathrm{E}}(s)\sigma$ in $\tilde{\mathcal{L}}_{\mathrm{E}}$. As $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ is a plant-I/O port of $\mathcal{S}_{\mathrm{PE}}$, it holds that $s\mu \in \mathcal{L}_{\mathrm{PE}} \forall \mu \in U_{\mathrm{E}}$. In particular, $s\sigma \in \mathcal{L}_{\mathrm{PE}}$. Moreover, $p_{\mathrm{P}}(s\sigma) = p_{\mathrm{P}}(s) \in \tilde{\mathcal{L}}_{\mathrm{P}}$. Together, $s\sigma \in \tilde{\mathcal{L}}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \tilde{\mathcal{L}}_{\mathrm{E}}$.

Observe that the above items cover all possible cases for s. Thus, for an arbitrary string $s \in \tilde{\mathcal{L}}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \tilde{\mathcal{L}}_{\mathrm{E}}$, there exists $\sigma \in \Sigma_{\mathrm{PE}}$ such that $s\sigma \in \tilde{\mathcal{L}}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \tilde{\mathcal{L}}_{\mathrm{E}}$, i.e. $\mathcal{S}_{\mathrm{PE}}$ is complete w.r.t. $\tilde{\mathcal{S}}_{\mathrm{P}}$ and $\tilde{\mathcal{S}}_{\mathrm{E}}$.

Y_{P}-liveness. As $\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ is Y_{P}-live and as $\tilde{\mathcal{L}}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \tilde{\mathcal{L}}_{\mathrm{E}} \subseteq \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$, also $\tilde{\mathcal{L}}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \tilde{\mathcal{L}}_{\mathrm{E}}$ is Y_{P}-live, see Lemma 3.3. □

Note that, in practice, the chosen constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$ usually are not fulfilled a priori by the surroundings of the plant and thus must be respected by the operator/controller or else be passed on as *requirements* to superposed operators/controllers.

Remark 3.3
Regarding the Definition 3.8 of Y_{P}-liveness, it is interesting to note that, in the framework [BW94] for supervisory control of *timed* DES, a state-based, but effectively identical property is used to postulate persistent passage of time (as stated in [OW90]): using a (clock-) *tick*-event representing the passage of one unit of time, a finite-state model of a timed discrete event system (TDES) is supposed to be *activity-loop-free*, meaning that, starting from a state of the TDES, there must be no loop (sequence of transitions leading back to the same state) that is free of *tick*-events. Consequently, as stated in [BW94], any infinite string generated by the TDES must include the occurrence of infinitely many *tick*-events. □

In the following section, we define the term of an I/O controller enforcing a safety specification and identify admissibility conditions for a complete and Y_{P}-live closed loop.

3.6 I/O Controller

The task of the I/O controller is to assist the operator in manipulating the environment according to a given specification; see Figure 3.10 a) and c). We propose to draft the specification as an I/O-plant model $\mathcal{S}_{\text{specCE}} = (\Sigma_{\text{CE}}, \mathcal{L}_{\text{specCE}})$ of the *desired* external closed loop, see Figure 3.10 c): by its plant-I/O port $(U_{\text{C}}, Y_{\text{C}})$ we introduce a set U_{C} of abstract desired tasks (modes of operation) for the closed loop and a set Y_{C} of desired responses of the closed loop to the operator. For each task $\mu \in U_{\text{C}}$, the specification expresses the desired behaviour of the closed loop w.r.t. the environment via sequences on the $(U_{\text{E}}, Y_{\text{E}})$-port and one or more associated responses denoting status, failure or completion of the task. To take into account and to exclude misbehaviour by the operator[5], an operator constraint $\mathcal{S}_{\text{C}} := (\Sigma_{\text{C}}, \mathcal{L}_{\text{C}})$ can be introduced. The original constraint $\mathcal{S}_{\text{E}}$ for liveness of the plant $\mathcal{S}_{\text{PE}}$ may also be assumed for $\mathcal{S}_{\text{specCE}}$ such that all in all $\mathcal{S}_{\text{specCE}}$ is reasonably designed to be complete and Y_{C}-live w.r.t. $\mathcal{S}_{\text{C}}$ and $\mathcal{S}_{\text{E}}$.

Example 3.4
Transport Unit. For the TU, a specification can be designed by the system $\mathcal{S}_{\text{specCE}} = (\Sigma_{\text{C}} \dot{\cup} \Sigma_{\text{E}}, \mathcal{L}_{\text{specCE}})$ with $\Sigma_{\text{C}} := U_{\text{C}} \dot{\cup} Y_{\text{C}} = \{stby, l2r\} \dot{\cup} \{idle\}$ and $\mathcal{L}_{\text{specCE}}$ as seen in Figure 3.9.

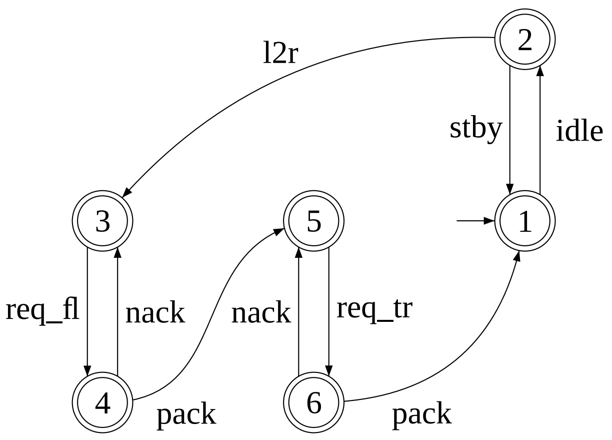


Figure 3.9: Specification for the TU

By the measurement event $idle$ we introduce a feedback to the operator notifying that the TU is ready for transport of the next workpiece. We specify that the operator can choose between two operational modes. After the command $stby$ (standby), no interaction with the environment is desired. With the command $l2r$ (left to right) we specify that a workpiece from left is requested from the environment (req_fl). In case of positive acknowledge, the workpiece shall be provided

[5] e.g. operator tries to trigger a final task before a respective initial task

to the right (req_tr). Note that the specification is complete and Y_{C}-live w.r.t. a *minimal* $\mathcal{S}_{\mathrm{C}}$ and the given $\mathcal{S}_{\mathrm{E}}$, i.e. we allow the operator for arbitrary orders of the commands $stby$ and $l2r$ and may assume the same constraints on the environment as for the original plant. Now, it is the controller's task to enforce appropriate Σ_{P}-sequences on the plant to achieve the specified behaviour with respect to the environment. □

In order to provide the operator with the desired view on the closed loop, the controller must provide the plant-I/O port $(U_{\mathrm{C}}, Y_{\mathrm{C}})$ to the operator. Events $\mu \in U_{\mathrm{C}}$ issued by the operator trigger more or less complex tasks to be performed by the controller and the plant. Occasionally, an abstract measurement event $\nu \in Y_{\mathrm{C}}$ has to be issued by the controller to indicate the status of the current task. Hence, the controller performs both, control and measurement aggregation and thereby provides an abstract external view $\mathcal{S}_{\mathrm{CE}} = (\Sigma_{\mathrm{CE}}, \mathcal{L}_{\mathrm{CE}})$ of the closed loop between operator and environment.
Formally, we define the I/O controller as a system with a controller-I/O port and a plant-I/O port that interact with the plant and the operator, respectively.

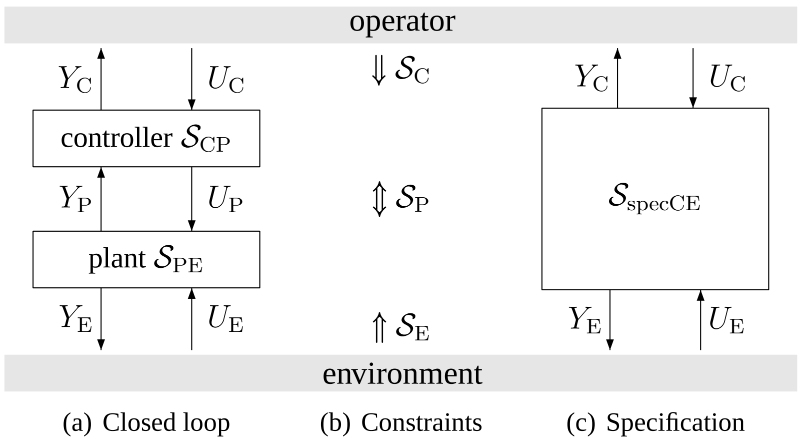

operator
Y_{C}
U_{C}
controller $\mathcal{S}_{\mathrm{CP}}$
Y_{P}
U_{P}
plant $\mathcal{S}_{\mathrm{PE}}$
Y_{E}
U_{E}
$\Downarrow \mathcal{S}_{\mathrm{C}}$
$\Updownarrow \mathcal{S}_{\mathrm{P}}$
$\Uparrow \mathcal{S}_{\mathrm{E}}$
$\mathcal{S}_{\mathrm{specCE}}$
environment
(a) Closed loop
(b) Constraints
(c) Specification

Figure 3.10: I/O Controller Synthesis Problem

Definition 3.10 (I/O Controller)
An *I/O controller* is a tuple $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$, where

(i) $(\Sigma_{\mathrm{CP}}, \mathcal{L}_{\mathrm{CP}})$ is a system with $\Sigma_{\mathrm{CP}} = \Sigma_{\mathrm{C}} \dot\cup \Sigma_{\mathrm{P}}$, $\Sigma_{\mathrm{C}} := U_{\mathrm{C}} \dot\cup Y_{\mathrm{C}}$, $\Sigma_{\mathrm{P}} := U_{\mathrm{P}} \dot\cup Y_{\mathrm{P}}$;

(ii) $(U_{\mathrm{C}}, Y_{\mathrm{C}})$ and $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ are a plant- and a controller-I/O port for $(\Sigma_{\mathrm{CP}}, \mathcal{L}_{\mathrm{CP}})$, respectively;

(iii) $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{((Y_{\mathrm{P}} U_{\mathrm{P}})^* (Y_{\mathrm{P}} Y_{\mathrm{C}} U_{\mathrm{C}} U_{\mathrm{P}})^*)^*}$;

(iv) $\mathcal{L}_{\mathrm{CP}}$ is complete.

□

Items (i) and (ii) enforce the language format $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{P}}(Y_{\mathrm{C}}U_{\mathrm{C}})^*U_{\mathrm{P}})^*)^*}$. Thus, item (iii) forbids the loop $(Y_{\mathrm{C}}U_{\mathrm{C}})^*$ and hence ensures that each command $\mu_{\mathrm{C}} \in U_{\mathrm{C}}$ from the operator is actually applied to the plant beginning with a control event $\mu_{\mathrm{P}} \in U_{\mathrm{P}}$. Operator commands without effect on the plant being controlled are thereby avoided. Note that controller and plant synchronize only via the alphabet Σ_{P}; from the perspective of the plant, the controller conforms with the alternation $\overline{(Y_{\mathrm{P}}U_{\mathrm{P}})^*}$ and, in particular, the controller cannot observe environment events.

When connecting a controller $\mathcal{S}_{\mathrm{CP}}$ and a plant $\mathcal{S}_{\mathrm{PE}}$ we obtain the *full closed loop* $(\Sigma_{\mathrm{CPE}}, \mathcal{L}_{\mathrm{CPE}})$ and the *external closed loop* $(\Sigma_{\mathrm{CE}}, \mathcal{L}_{\mathrm{CE}})$ with the full closed-loop behaviour $\mathcal{L}_{\mathrm{CPE}} := \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$ and the external closed-loop behaviour $\mathcal{L}_{\mathrm{CE}} := p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}})$, respectively. For the language format of the full closed loop under constraints, we obtain

$$\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{P}}Y_{\mathrm{C}}U_{\mathrm{C}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}.$$

As an important result of the I/O structure, the external closed-loop behaviour itself can be seen to be an I/O plant:

Proposition 3.3
Let $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be an I/O plant, and let $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$ be an I/O controller.
Then, the external closed-loop system $\mathcal{S}_{\mathrm{CE}} := \mathcal{S}_{\mathrm{CP}} \parallel_{\mathrm{ex}} \mathcal{S}_{\mathrm{PE}} := (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{CE}})$ with $\mathcal{L}_{\mathrm{CE}} = p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}})$ is an I/O plant. □

Proof We show that $\mathcal{S}_{\mathrm{CE}}$ provides all I/O-plant properties.

(i) $\mathcal{L}_{\mathrm{CE}} \subseteq p_{\mathrm{CE}}[\overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{P}}Y_{\mathrm{C}}U_{\mathrm{C}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}] = \overline{((Y_{\mathrm{C}}U_{\mathrm{C}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*} \subseteq {\Sigma_{\mathrm{CE}}}^*$
$\Rightarrow$ $(\Sigma_{\mathrm{CE}}, \mathcal{L}_{\mathrm{CE}})$ is a system.

(ii)
- $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ is plant-I/O port of $(\Sigma_{\mathrm{CE}}, \mathcal{L}_{\mathrm{CE}})$. Proof:
  Obviously, $\Sigma_{\mathrm{CE}} = W \dot\cup U_{\mathrm{E}} \dot\cup Y_{\mathrm{E}}$ with $W = U_{\mathrm{C}} \dot\cup Y_{\mathrm{C}} = \Sigma_{\mathrm{CE}} - Y_{\mathrm{E}} - U_{\mathrm{E}}$.

  As shown in (i), $\mathcal{L}_{\mathrm{CE}} \subseteq \overline{(W^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}$.

  $\Sigma_{\mathrm{E}} \cap \Sigma_{\mathrm{CP}} = \emptyset \;\Rightarrow\; (\forall s \in \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}},\ \forall u_{\mathrm{E}} \in U_{\mathrm{E}})\ [p_{\mathrm{PE}}(s)u_{\mathrm{E}} \in \mathcal{L}_{\mathrm{PE}} \;\Rightarrow\; su_{\mathrm{E}} \in \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}]$. Hence, $p_{\mathrm{CE}}(su_{\mathrm{E}}) = p_{\mathrm{CE}}(s)u_{\mathrm{E}} \in \mathcal{L}_{\mathrm{CE}}$. Thus, the free input property of U_{E} of I/O-plant $(\Sigma_{\mathrm{PE}}, \mathcal{L}_{\mathrm{PE}})$ is retained under control and projection.
- $(U_{\mathrm{C}}, Y_{\mathrm{C}})$ is plant-I/O port of $(\Sigma_{\mathrm{CE}}, \mathcal{L}_{\mathrm{CE}})$: as above.

□

Moreover, persistent feedback to the operator has to be preserved in the closed loop, i.e. Y_{C}-liveness is required. We observe that the I/O structure itself is not sufficiently strong to imply completeness and Y_{C}-liveness for the full or external closed loop under arbitrary control action.

As, for example, the controller may not comply with the operator constraint $\mathcal{S}_\mathrm{P}$ identified for liveness of the plant, the closed-loop system may run into a *deadlock* situation, which is considered undesirable. More subtle is the fact that arbitrary length strings $s \in (\Sigma_\mathrm{P} \cup \Sigma_\mathrm{E})^*$ may occur between each pair of control and measurement events $\mu \in U_\mathrm{C}$ and $\nu \in Y_\mathrm{C}$, which amounts to measurement aggregation. For the considered prefix-closed languages this implies that the closed-loop could also evolve on an infinite length string $s \in (\Sigma_\mathrm{P} \cup \Sigma_\mathrm{E})^\omega$. In this *livelock* situation the operator no longer receives measurement events $\nu \in Y_\mathrm{C}$ and, hence, can not issue further control events.

The following admissibility condition addresses both issues in that it implies completeness and Y_C-liveness for the closed-loop system; see Proposition 3.4 and Theorem 3.1.

Definition 3.11 (Admissibility)
Let $\mathcal{S}_\mathrm{PE} = (U_\mathrm{P}, Y_\mathrm{P}, U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{PE})$ be an I/O plant and let $\mathcal{S}_\mathrm{C} = (U_\mathrm{C}, Y_\mathrm{C}, \mathcal{L}_\mathrm{C})$, $\mathcal{S}_\mathrm{P} = (U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_\mathrm{P})$ and $\mathcal{S}_\mathrm{E} = (U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{E})$ be constraints. Then, an I/O controller $\mathcal{S}_\mathrm{CP} = (U_\mathrm{C}, Y_\mathrm{C}, U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_\mathrm{CP})$ is *admissible* to the plant $\mathcal{S}_\mathrm{PE}$ w.r.t. the constraints $\mathcal{S}_\mathrm{C}$, $\mathcal{S}_\mathrm{P}$, and $\mathcal{S}_\mathrm{E}$ if

(i) $p_\mathrm{P}(\mathcal{L}_\mathrm{C} \parallel \mathcal{L}_\mathrm{CP} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}) \subseteq \mathcal{L}_\mathrm{P}$;

(ii) $\mathcal{L}_\mathrm{CP} \parallel \mathcal{L}_\mathrm{PE}$ is Y_C-live w.r.t. $\mathcal{S}_\mathrm{C}$ and $\mathcal{S}_\mathrm{E}$.

□

Remark 3.4
Note that item (i) in the above definition implies $p_\mathrm{PE}(\mathcal{L}_\mathrm{C} \parallel \mathcal{L}_\mathrm{CP} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}) \subseteq \mathcal{L}_\mathrm{P} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}$, i.e. the plant sees the controller as a subset of the constraint $\mathcal{S}_\mathrm{P}$. □

The above definition provides each constraint depicted in Fig. 3.10 b) with a certain role. While $\mathcal{S}_\mathrm{C}$ and $\mathcal{S}_\mathrm{E}$ must be fulfilled by the external configuration in both items (i) and (ii), condition (i) requires that the setting $\mathcal{L}_\mathrm{C} \parallel \mathcal{L}_\mathrm{CP}$ complies with the constraint $\mathcal{S}_\mathrm{P}$. Hence $\mathcal{S}_\mathrm{P}$ has to be met by the controller. This condition already ensures completeness of the full and the external closed loop behaviour, see Proposition 3.4. As a technical consequence, the set $(\mathcal{L}_\mathrm{C} \parallel \mathcal{L}_\mathrm{CP} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E})^\infty$ is non-empty, which is relevant to condition (ii) that demands Y_C-liveness of the full closed loop behaviour. For the full closed, we obtain the following result.

Proposition 3.4
Let $\mathcal{S}_\mathrm{CP} = (U_\mathrm{C}, Y_\mathrm{C}, U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_\mathrm{CP})$ be an I/O controller, let $\mathcal{S}_\mathrm{PE} = (U_\mathrm{P}, Y_\mathrm{P}, U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{PE})$ be an I/O plant, and let $\mathcal{S}_\mathrm{C} = (U_\mathrm{C}, Y_\mathrm{C}, \mathcal{L}_\mathrm{C})$, $\mathcal{S}_\mathrm{P} = (U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_\mathrm{P})$ and $\mathcal{S}_\mathrm{E} = (U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{E})$ be constraints.

(i) If $\mathcal{S}_\mathrm{PE}$ is complete w.r.t. $\mathcal{S}_\mathrm{P}$ and $\mathcal{S}_\mathrm{E}$, and $\mathcal{S}_\mathrm{CP}$ meets the admissibility condition (i), then $\mathcal{L}_\mathrm{C} \parallel \mathcal{L}_\mathrm{CP} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}$ is complete.

(ii) If in addition $\mathcal{S}_\mathrm{CP}$ meets the admissibility condition (ii), then $\mathcal{L}_\mathrm{C} \parallel p_\mathrm{CE}(\mathcal{L}_\mathrm{CP} \parallel \mathcal{L}_\mathrm{PE}) \parallel \mathcal{L}_\mathrm{E}$ is complete.

□

Proof See Appendix A.2 □

Note that, in general, the natural projection of a language can both, artificially produce completeness of the result by hiding deadlocks or abolish completeness of the original language by hiding all extensions of some string. Hence, Proposition 3.4 also states that both is not the case for the full and external closed loop.

For the external closed loop, obtain the following important result.

Theorem 3.1 (External Closed Loop)
Let the I/O plant $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be complete and Y_{P}-live w.r.t. the constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$, and let $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$ be admissible to $\mathcal{S}_{\mathrm{PE}}$ w.r.t. the constraints $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$, and $\mathcal{S}_{\mathrm{E}}$. Then the external closed-loop system $\mathcal{S}_{\mathrm{CE}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{CE}})$, $\mathcal{L}_{\mathrm{CE}} = p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}})$, is

(i) an I/O plant;

(ii) complete w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$;

(iii) Y_{C}-live w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$.

□

Proof
(i) $\mathcal{S}_{\mathrm{CE}}$ is an I/O plant according to Proposition 3.3.

(ii) $\mathcal{S}_{\mathrm{CE}}$ is complete w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$ according to Proposition 3.4, item (ii).

(iii) $\mathcal{S}_{\mathrm{CE}}$ is Y_{C}-live w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$. Proof: Note that the full closed loop behaviour $\mathcal{S}_{\mathrm{CPE}}$ is Y_{C}-live w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$. This means $p_{\mathrm{YC}}(w') \in Y_{\mathrm{C}}^{\omega}$ for all $w' \in (\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CPE}} \parallel \mathcal{L}_{\mathrm{E}})^{\infty}$. Observe also that for all $w \in (\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CE}} \parallel \mathcal{L}_{\mathrm{E}})^{\infty}$ it holds that $w = p_{\mathrm{CE}}(w')$ for some $w' \in (\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CPE}} \parallel \mathcal{L}_{\mathrm{E}})^{\infty}$, and $p_{\mathrm{YC}}(w) = p_{\mathrm{YC}}(p_{\mathrm{CE}}(w')) = p_{\mathrm{YC}}(w')$. Hence, $p_{\mathrm{YC}}(w) \in Y_{\mathrm{C}}^{\omega}$ for all $w \in \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CPE}} \parallel \mathcal{L}_{\mathrm{E}}$.

□

According to this result, the admissibility condition implies that the external closed loop $\mathcal{S}_{\mathrm{CE}}$ is an I/O plant that is complete and Y_{C}-live with respect to the given constraints. Thus, in a hierarchical control architecture, the closed loop can serve as a plant model for the design of the next layer of control and measurement aggregation.

Hence, the problem to be solved is the synthesis of an admissible I/O controller. The controller synthesis problem is given by the setting depicted in Figure 3.10, with the I/O controller as the desired solution.

Definition 3.12 (I/O Controller Synthesis Problem)
An *I/O controller synthesis problem* is a tuple $(\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$ where $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ is an I/O plant, $\mathcal{S}_{\mathrm{C}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, \mathcal{L}_{\mathrm{C}})$, $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$ and $\mathcal{S}_{\mathrm{E}} =$

$(U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{E})$ are constraints, and $\mathcal{S}_\mathrm{specCE} = (U_\mathrm{C}, Y_\mathrm{C}, U_\mathrm{E}, Y_\mathrm{E}, \mathcal{L}_\mathrm{specCE})$ is a *safety specification*. A *solution for the I/O controller synthesis problem* is an I/O controller $\mathcal{S}_\mathrm{CP} = (U_\mathrm{C}, Y_\mathrm{C}, U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_\mathrm{CP})$ that is admissible to $\mathcal{S}_\mathrm{PE}$ w.r.t. $\mathcal{S}_\mathrm{C}$, $\mathcal{S}_\mathrm{P}$, and $\mathcal{S}_\mathrm{E}$ and that enforces the safety specification $\mathcal{S}_\mathrm{specCE}$ on $\mathcal{S}_\mathrm{PE}$ w.r.t. $\mathcal{S}_\mathrm{C}$ and $\mathcal{S}_\mathrm{E}$, i.e. $p_\mathrm{CE}(\mathcal{L}_\mathrm{C} \parallel \mathcal{L}_\mathrm{CP} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}) \subseteq \mathcal{L}_\mathrm{specCE}$. □

Example 3.5
Transport Unit. The I/O-plant model $\mathcal{S}_\mathrm{PE}$ of the TU as in Figure 3.5 b), the constraints $\mathcal{S}_\mathrm{P}$ and $\mathcal{S}_\mathrm{E}$ as in Figures 3.6 and 3.7, a minimal constraint $\mathcal{S}_\mathrm{C}$ and the specification $\mathcal{S}_\mathrm{specCE}$ as in Figure 3.9 pose an I/O controller synthesis problem. □

As the environment events Σ_E are not observable by the controller, the above problem amounts to a controller synthesis problem under partial observation; we again refer to [MR99, KGM92] where related problems are addressed. Note that the trivial controller (with empty language) solves the synthesis problem. Hence, the following theorem establishes unique existence of a least restrictive solution within a *finite* family of solutions.

Theorem 3.2
Given an I/O controller synthesis problem $(\mathcal{S}_\mathrm{PE}, \mathcal{S}_\mathrm{C}, \mathcal{S}_\mathrm{P}, \mathcal{S}_\mathrm{E}, \mathcal{S}_\mathrm{specCE})$, let $\mathcal{S}_{\mathrm{CP}\alpha} = (U_\mathrm{C}, Y_\mathrm{C}, U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_{\mathrm{CP}\alpha})$, $\alpha \in A$, denote a finite family of solutions. Then $\mathcal{S}_\mathrm{CP} = (U_\mathrm{C}, Y_\mathrm{C}, U_\mathrm{P}, Y_\mathrm{P}, \mathcal{L}_\mathrm{CP})$, $\mathcal{L}_\mathrm{CP} := \cup_{\alpha \in A} \mathcal{L}_{\mathrm{CP}\alpha}$, also solves the problem. □

Proof We show that $\mathcal{S}_\mathrm{CP}$ is admissible and enforces $\mathcal{S}_\mathrm{specCE}$. We begin with proving that $\mathcal{S}_\mathrm{CP}$ is a controller that provides all admissibility properties.

- $\mathcal{S}_\mathrm{CP}$ is an I/O-controller. Proof:

  (i) It is obvious that $\mathcal{S}_\mathrm{CP}$ is a system.

  (iia) $(U_\mathrm{C}, Y_\mathrm{C})$ is a plant-I/O port. Proof: pick an arbitrary $s\nu \in \mathcal{L}_\mathrm{CP},\ \nu \in Y_\mathrm{C}$ and an arbitrary $\mu \in U_\mathrm{C}$. Thus, there exists some α such that $s\nu \in \mathcal{L}_{\mathrm{CP}\alpha}$. Consequently, $s\nu\mu \in \mathcal{L}_{\mathrm{CP}\alpha}$ and thus $s\nu\mu \in \mathcal{L}_\mathrm{CP}$.

  (iia) $(U_\mathrm{P}, Y_\mathrm{P})$ is a controller-I/O port. Proof: pick an arbitrary $s\mu \in \mathcal{L}_\mathrm{CP},\ \mu \in U_\mathrm{P}$ and an arbitrary $\nu \in Y_\mathrm{P}$. Thus, there exists some α such that $s\mu \in \mathcal{L}_{\mathrm{CP}\alpha}$. Consequently, $s\mu\nu \in \mathcal{L}_{\mathrm{CP}\alpha}$ and thus $s\mu\nu \in \mathcal{L}_\mathrm{CP}$.

  (iii) $\mathcal{L}_{\mathrm{CP}\alpha} \subseteq \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*},\ \forall \alpha \in A.$

  $\Rightarrow \cup_\alpha \mathcal{L}_{\mathrm{CP}\alpha} \subseteq \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*}.$

  (iv) $\mathcal{S}_\mathrm{CP}$ is complete. Proof: pick an arbitrary $s \in \mathcal{L}_\mathrm{CP}$. Thus, there exists some α such that $s \in \mathcal{L}_{\mathrm{CP}\alpha}$. As $(\Sigma_\mathrm{CP}, \mathcal{L}_{\mathrm{CP}\alpha})$ is complete, there exists $\sigma \in \Sigma_\mathrm{CP}$ such that $s\sigma \in \mathcal{L}_{\mathrm{CP}\alpha}$. Consequently, $s\sigma \in \mathcal{L}_\mathrm{CP}$.

- $p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{P}}$.
  Proof:
  $p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \cup_\alpha[\mathcal{L}_{\mathrm{CP}\alpha}] \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) =^{\text{Lemma A.1}}$
  $p_{\mathrm{P}}(\cup_\alpha[(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}\alpha} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})]) =$
  $\cup_\alpha p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}\alpha} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{P}}$.

- $\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$ is Y_{C}-live w.r.t. $\mathcal{L}_{\mathrm{C}}$ and $\mathcal{L}_{\mathrm{E}}$. Proof:

  Pick $w \in (\mathcal{L}_{\mathrm{C}} \parallel \cup_\alpha[\mathcal{L}_{\mathrm{CP}\alpha}] \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})^\infty =^{\text{Lemma A.1}}$
  $(\cup_\alpha[\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}\alpha} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}])^\infty =^{\text{LemmaA.6}}$
  $\cup_\alpha[(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}\alpha} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})^\infty]$.

  Thus, there exists some α such that $w \in (\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}\alpha} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})^\infty$. As $\mathcal{L}_{\mathrm{CP}\alpha}$ is admissible to $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{L}_{\mathrm{C}}$ and $\mathcal{L}_{\mathrm{E}}$, it holds that $p_{\mathrm{YC}}(w) \in Y_{\mathrm{C}}^\omega$.

Finally, $\mathcal{S}_{\mathrm{CP}}$ enforces $\mathcal{S}_{\mathrm{specCE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$. Proof:
$p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{C}} \parallel \cup_\alpha(\mathcal{L}_{\mathrm{CP}\alpha}) \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) =^{\text{Lemma A.1}}$
$p_{\mathrm{CE}}(\cup_\alpha(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}\alpha} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})) =^{\text{Equation 2.3}}$
$\cup_\alpha(p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}\alpha} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})) \subseteq \mathcal{L}_{\mathrm{specCE}}$ □

Note that this result does not hold for the union of an infinite set of solutions, as Y_{C}-liveness is not necessarily preserved under infinite union.

Proposition 3.5
Given an I/O controller synthesis problem $\Pi := (\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$, let $\mathcal{S}_{\mathrm{CP}\alpha} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}\alpha})$, $\alpha \in \mathbb{N}$, denote an infinite family of solutions of Π.
Then $\mathcal{S}_{\mathrm{CP}} = (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{CP}})$, $\mathcal{L}_{\mathrm{CP}} := \cup_{\alpha \in \mathbb{N}_0} \mathcal{L}_{\mathrm{CP}\alpha}$, may not be a solution of Π, in general. □

Proof Given all entities of the above proposition, we show that there exist I/O controller synthesis problems such that $\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$ is *not* Y_{C}-live w.r.t. $\mathcal{L}_{\mathrm{C}}$ and $\mathcal{L}_{\mathrm{E}}$.
Consider a simple counterexample[6] $\Pi := (\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$ with $\mathcal{S}_{\mathrm{PE}} := (\{y_{\mathrm{P}}, u_{\mathrm{P}}, y_{\mathrm{E}}, u_{\mathrm{E}}\}, \overline{(y_{\mathrm{P}} u_{\mathrm{P}})^*})$, $\mathcal{S}_{\mathrm{specCE}} := (\{y_{\mathrm{C}}, u_{\mathrm{C}}, y_{\mathrm{E}}, u_{\mathrm{E}}\}, \overline{(y_{\mathrm{C}} u_{\mathrm{C}})^*})$ and all constraints minimal. From the following infinite family of controllers $\mathcal{S}_{\mathrm{CP}\alpha} = (\Sigma_{\mathrm{CP}}, \overline{(y_{\mathrm{P}}(u_{\mathrm{P}} y_{\mathrm{P}})^\alpha y_{\mathrm{C}} u_{\mathrm{C}} u_{\mathrm{P}})^*})$, $\alpha \in \mathbb{N}_0$, obviously each single member is a solution. However, the infinite union of all solutions leads to $\mathcal{S}_{\mathrm{CP}} = (\Sigma_{\mathrm{CP}}, \overline{(y_{\mathrm{P}}(u_{\mathrm{P}} y_{\mathrm{P}})^* y_{\mathrm{C}} u_{\mathrm{C}} u_{\mathrm{P}})^*})$. When attached to the plant, the full closed loop behaviour is $\mathcal{L}_{\mathrm{CPE}} = \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} = \overline{(y_{\mathrm{P}}(u_{\mathrm{P}} y_{\mathrm{P}})^* y_{\mathrm{C}} u_{\mathrm{C}} u_{\mathrm{P}})^*}$. One can see that the limit $\mathcal{L}_{\mathrm{CPE}}^\infty$ contains the string $w = y_{\mathrm{P}}(u_{\mathrm{P}} y_{\mathrm{P}})^\omega$ with $p_{\mathrm{YC}}(w) = \epsilon \notin Y_{\mathrm{C}}^\omega$. Hence, $\mathcal{L}_{\mathrm{CPE}}$ is not Y_{C}-live w.r.t. $\mathcal{L}_{\mathrm{P}}$ and $\mathcal{L}_{\mathrm{E}}$, and consequently $\mathcal{S}_{\mathrm{CP}}$ is not a solution of Π. □

[6]This example is constructed for simplicity of the proof. There also exist praxis relevant examples.

Note that the above proposition does not affect any of the results presented in this framework. However, the controller design has to be implemented with additional requirements such that Y_C-liveness is achieved also under infinite unions of solutions or the set of all solutions is finite. The shape of these requirements depends on the respective application. A possible elaboration of these requirements respecting the application point of view together with a respective controller design algorithm are proposed in Chapter 4.

Example 3.6
Transport Unit. For the I/O controller synthesis problem of the TU, our synthesis algorithm returns the controller $\mathcal{S}_{\mathrm{CP}}$ with $\mathcal{L}_{\mathrm{CP}}$ as depicted in Figure 3.11.

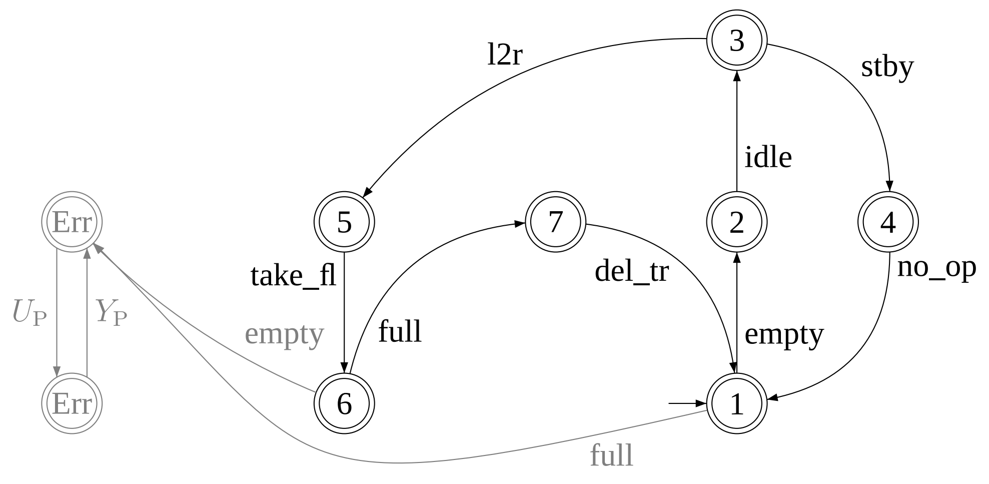


Figure 3.11: Controller for the TU

Formally, the I/O controller accepts all measurement events of the plant, even those that can actually not occur; the respective transitions are denoted by gray arrows leading to error states that represent an error behaviour $\mathcal{K}_{\mathrm{CP}}^{err}$ (see Chapter 4) and are never reached. It is verified that if the environment constraint $\mathcal{S}_{\mathrm{E}}$ is fulfilled, the closed loop is complete and Y_C-live and features the external behaviour specified by $\mathcal{S}_{\mathrm{specCE}}$. □

We arrive at one of the central statements of this contribution: Our framework makes similar use of the I/O structure as [MR99] and thereby allows for *abstraction based controller synthesis*; i.e. solutions obtained for a plant abstraction are guaranteed to solve the original problem.

Theorem 3.3 (Abstraction-Based Control)
Given an I/O plant $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$, let $\tilde{\mathcal{S}}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \tilde{\mathcal{L}}_{\mathrm{PE}})$ be a plant abstraction, i.e. $\mathcal{L}_{\mathrm{PE}} \subseteq \tilde{\mathcal{L}}_{\mathrm{PE}}$. If the plant $\mathcal{S}_{\mathrm{PE}}$ is complete and Y_{P}-live w.r.t. the constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$

and if $\mathcal{S}_{\mathrm{CP}}$ solves the I/O controller synthesis problem $(\tilde{\mathcal{S}}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$, then $\mathcal{S}_{\mathrm{CP}}$ also solves $(\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$. □

Proof

(I) $\mathcal{S}_{\mathrm{CP}}$ is admissible to $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$. Proof:

- As $\mathcal{S}_{\mathrm{CP}}$ is admissible to $\tilde{\mathcal{S}}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$, it holds that $\mathcal{S}_{\mathrm{CP}}$ is a controller by definition of admissibility.
- As $\mathcal{S}_{\mathrm{CP}}$ is admissible to $\tilde{\mathcal{S}}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$, it holds that $p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \tilde{\mathcal{L}}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{P}}$. Note that, with $\mathcal{L}_{\mathrm{PE}} \subseteq \tilde{\mathcal{L}}_{\mathrm{PE}}$, it follows that
$p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \tilde{\mathcal{L}}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{P}}$.
- As $(\Sigma_{\mathrm{CPE}}, \mathcal{L}_{\mathrm{CP}} \parallel \tilde{\mathcal{L}}_{\mathrm{PE}})$ is Y_{C}-live w.r.t. $\mathcal{L}_{\mathrm{C}}$ and $\mathcal{L}_{\mathrm{E}}$,[7] for all $w \in (\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \tilde{\mathcal{L}}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})^{\infty}$ it holds that $p_{\mathrm{YC}}(w) \in Y_{\mathrm{C}}^{\omega}$. In particular, this holds for all $w \in (\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})^{\infty} \subseteq (\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \tilde{\mathcal{L}}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})^{\infty}$. Hence $(\Sigma_{\mathrm{CPE}}, \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}})$ is Y_{C}-live w.r.t. $\mathcal{L}_{\mathrm{C}}$ and $\mathcal{L}_{\mathrm{E}}$.

(II) $\mathcal{S}_{\mathrm{CP}}$ enforces $\mathcal{L}_{\mathrm{specCE}}$ on $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$. Proof: As $\mathcal{L}_{\mathrm{CP}}$ enforces $\mathcal{L}_{\mathrm{specCE}}$ on $\tilde{\mathcal{S}}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$, it holds that $p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \tilde{\mathcal{L}}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{specCE}}$. Note that $p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \tilde{\mathcal{L}}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{specCE}}$. Hence, $\mathcal{S}_{\mathrm{CP}}$ enforces $\mathcal{L}_{\mathrm{specCE}}$ on $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$.

□

If the abstraction is of less complexity (number of states) the computational effort for controller synthesis is reduced accordingly. However, as a well-known downside of abstraction-based control, there is no guarantee that there exists a non-trivial controller for the plant abstraction even if there does exist one for the original plant. Hence the question, how to obtain a "good" abstraction, i.e. an abstraction that can be realized on a state space that is small in comparison to the original plant while still allowing for successful controller synthesis. We propose the safety specification $\mathcal{S}_{\mathrm{specCE}}$ of the preceding design step as a plant abstraction of the external closed-loop behaviour $\mathcal{S}_{\mathrm{CE}}$, as, by being enforced on the plant, it meets the abstraction condition $\mathcal{L}_{\mathrm{CE}} \subseteq \mathcal{L}_{\mathrm{specCE}}$ and represents those aspects of the preceding design step that are relevant for subsequent controller design. Consequently, we expect to obtain a non-trivial solution based on that abstraction. This line of thought has been further elaborated in the context of hybrid systems [MR05, MRD03].

Example 3.7
Transport Unit. For the design of superposed controllers for a *chain of TU's* explained in Chapter 5, we do not compute the external closed-loop behaviour of each locally controlled TU, but rather

[7]Follows from admissibility of $\mathcal{S}_{\mathrm{CP}}$ to $\tilde{\mathcal{S}}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$.

use the specification as seen in Figure 3.9 as an abstracted plant model of the locally controlled behaviour. □

The following definition provides an automata structure that corresponds to an I/O controller.

Definition 3.13 (I/O Controller Form)
A generator $G := (\Sigma, Q, \delta, q_0, Q_\mathrm{m})$ is in *I/O-controller form* if

(i) $\Sigma = U_\mathrm{C} \dot\cup Y_\mathrm{C} \dot\cup U_\mathrm{P} \dot\cup Y_\mathrm{P}$ with nonempty alphabets $U_\mathrm{C},\ Y_\mathrm{C},\ U_\mathrm{P},\ Y_\mathrm{P}$

(ii) $Q = Q_\mathrm{UC} \dot\cup Q_\mathrm{YC,UP} \dot\cup Q_\mathrm{UP} \dot\cup Q_\mathrm{YP}$

(iii) $q_0 \in Q_\mathrm{YP}$

(iv) $[\forall q \in Q_\mathrm{YP}, \sigma \in \Sigma](\delta(q,\sigma)! \Rightarrow (\sigma \in Y_\mathrm{P} \wedge \delta(q,\sigma) \in Q_\mathrm{YC,UP} \cup Q_\mathrm{UP}))$

(v) $[\forall q \in Q_\mathrm{UP}, \sigma \in \Sigma](\delta(q,\sigma)! \Rightarrow (\sigma \in U_\mathrm{P} \wedge \delta(q,\sigma) \in Q_\mathrm{YP}))$

(vi) $[\forall q \in Q_\mathrm{YC,UP}, \sigma \in \Sigma](\delta(q,\sigma)! \Rightarrow (\sigma \in U_\mathrm{P} \wedge \delta(q,\sigma) \in Q_\mathrm{YP})) \vee (\sigma \in Y_\mathrm{C} \wedge \delta(q,\sigma) \in Q_\mathrm{UC}))$

(vii) $[\forall q \in Q_\mathrm{UC}, \sigma \in \Sigma](\delta(q,\sigma)! \Rightarrow (\sigma \in U_\mathrm{C} \wedge \delta(q,\sigma) \in Q_\mathrm{UP}))$

(viii) $[\forall q \in Q_\mathrm{UC}, \mu \in U_\mathrm{C}](\delta(q,\sigma)!)$

(ix) $[\forall q \in Q_\mathrm{YP}, \mu \in Y_\mathrm{P}](\delta(q,\sigma)!)$

(x) $Q_\mathrm{m} = Q$

(xi) $[\forall q \in Q](\exists \sigma \in \Sigma : \delta(q,\sigma)!)$

(xii) G is accessible.

□

Lemma 3.4
If a generator $G := (\Sigma, Q, \delta, q_0, Q_\mathrm{m})$ is in I/O-controller form, then the system $(\Sigma, \mathcal{L}_\mathrm{m}(G))$ is an I/O controller. □

Proof See appendix, Proof A.2. □

Chapter 4

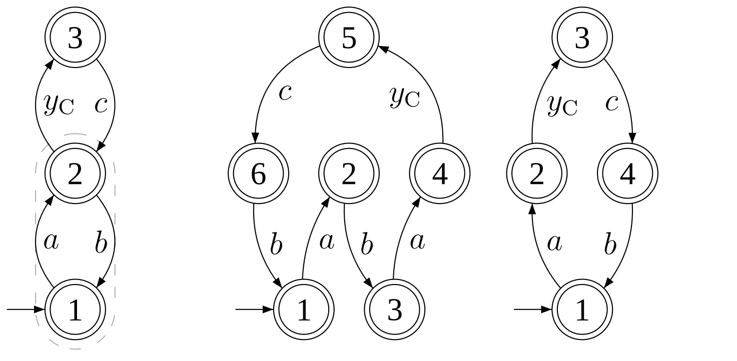
Controller Synthesis

It is an approved method of discrete event controller synthesis to first reduce the possible plant behaviour to a desired (but maybe infeasible) behaviour by composition with the specification. From the desired behaviour, the closed-loop behaviour, i.e. a behaviour that features the desired liveness properties and that can be achieved by a controller, is deduced by subset construction. From this result, the solution, i.e. the controller that achieves the closed-loop behaviour is then extracted. Our synthesis procedure, which is presented in this chapter, conforms with this method. The basic ideas of the procedure have also been published in [PMS08]. Thereby, a major aspect is to restrict a given language to a Y_C-live sublanguage, see admissibility condition (ii) in Definition 3.11.

4.1 Y_C-Acyclic Sublanguage

The calculation of a Y_C-live sublanguage involves the detection of strings in the composition of plant and specification that compromise Y_C-liveness. In the automata representation of the considered language, such a string is indicated by a so-called Y_C-less cycle of states, within which each state can be visited arbitrarily often without the occurrence of any Y_C-event.

Example 4.1
Consider the generator of the language $\mathcal{L}$ in Figure 4.1 a) over $\Sigma = \{a, b, c, y_\mathrm{C}\}$ with $Y_\mathrm{C} := \{y_\mathrm{C}\}$.



(a) Language $\mathcal{L}$ (b) Sublanguage $\mathcal{K}_1$ (c) Sublanguage $\mathcal{K}_2$

Figure 4.1: Y_C-live and Y_C-acyclic sublanguage

The indicated cycle of the states 1 and 2 is a Y_C-less cycle, as they can be visited infinitely often without the occurrence of the event y_C. □

We observe that a string that violates Y_C-liveness features two properties:

a) Nerode-equivalence to at least one of its own strict prefixes (i.e., a cycle is closed)

b) The extension from each Nerode-equivalent prefix to the considered string does not contain any Y_C-event.

Example 4.2
The states 1 and 2 in Figure 4.1 a) represent e.g. the string aba and its nerode-equivalent prefix a, where the continuation ba from a to aba does not contain the event y_C (i.e. items a) and b) above are met). Due to the nerode-equivalence, we can append ba to a arbitrarily often to obtain a string of $\mathcal{L}$, i.e. $a(ba)^* \subseteq \mathcal{L}$. Consequently, $a(ba)^\omega \in \mathcal{L}^\infty$ with $p_{Y_C}(a(ba)^\omega) = \epsilon \notin Y_C^\omega$, i.e. $\mathcal{L}$ is not Y_C-live. □

A language that does not contain such strings is Y_C-live. This can be shown by the following proposition introducing an equivalent formulation of the Y_C-liveness property based on the above items a) and b). From this property, we will deduce the family of so-called Y_C-*Acyclic sublanguages* featuring a unique supremal element.

Proposition 4.1
Let $\mathcal{K}$ be a regular language over the alphabet $\Sigma \supseteq Y_C$. $\mathcal{K}$ is Y_C-live if and only if

$$\forall s \in \mathcal{K} : (\forall t \neq \epsilon)[st \equiv_{\mathcal{K}} s \Rightarrow p_{Y_C}(t) \neq \epsilon] \tag{4.1}$$

where $\equiv_{\mathcal{K}}$ denotes the Nerode equivalence over Σ^* w.r.t. $\mathcal{K}$.

□

Proof Both directions of the equivalence are shown separately:
Part A) Let $\mathcal{K}$ be a regular language over the alphabet $\Sigma \supseteq Y_{\mathrm{C}}$ with the limit $\mathcal{K}^\infty$. The following implication is true:
$\mathcal{K}$ meets Property (4.1) of Proposition 4.1 $\Rightarrow$ $\mathcal{K}$ is Y_{C}-live.
Proof: Pick some arbitrary $w \in \mathcal{K}^\infty$. Observe according to the definition of the limit: $\exists\,(n_i)_{i\in\mathbb{N}_0}, n_{i+1} > n_i : w^{n_i} \in \mathcal{K}$, i.e. infinitely many finite prefixes of w are element of $\mathcal{K}$.
As $\mathcal{K}$ is regular, we can partition $\mathcal{K}$ into a finite set of Nerode cells. Consequently, at least one Nerode cell has to hold an infinite number of (but not necessarily all) prefixes w^{n_i}, and we can conclude:
$\exists\,(\tilde{n}_j)_{j\in\mathbb{N}_0}, \tilde{n}_{j+1} > \tilde{n}_j : w^{\tilde{n}_{j+1}} \equiv_{\mathcal{K}} w^{\tilde{n}_j}$, where, for each j, $w^{\tilde{n}_j} = w^{n_i}$ for some i. With $w^{\tilde{n}_0}$ we start the following procedure:
As $w^{\tilde{n}_0} < w^{\tilde{n}_1}$, we can write $w^{\tilde{n}_1} = w^{\tilde{n}_0}t_0$ for some $t_0 \in \Sigma^* - \{\epsilon\}$. Note that, as $\mathcal{K}$ meets Property (4.1) and as $w^{\tilde{n}_1} \equiv_{\mathcal{K}} w^{\tilde{n}_0}$, it holds that $p_{\mathrm{YC}}(t_0) \neq \epsilon$. Accordingly, $w^{\tilde{n}_2} = w^{\tilde{n}_0}t_0t_1$ with $p_{\mathrm{YC}}(t_1) \neq \epsilon$. Repeating this procedure for each j, we obtain $w^{\tilde{n}_0}t_0t_1t_2\cdots = w$. As $p_{\mathrm{YC}}(t_j) \neq \epsilon$ for all j, it holds that $p_{\mathrm{YC}}(w) = p_{\mathrm{YC}}(w^{\tilde{n}_0}t_0t_1\dots) = p_{\mathrm{YC}}(w^{\tilde{n}_0})p_{\mathrm{YC}}(t_0)p_{\mathrm{YC}}(t_1)\cdots \in Y_{\mathrm{C}}^\omega$.
Part B) Let $\mathcal{K}$ be a regular language over the alphabet $\Sigma \supseteq Y_{\mathrm{C}}$ with the limit $\mathcal{K}^\infty$. The following implication is true:
$\mathcal{K}$ is Y_{C}-live $\Rightarrow$ $\mathcal{K}$ meets Property (4.1) of Proposition 4.1.
Proof: For arbitrary $s \in \mathcal{K}$, consider the following set of extensions of s in $\mathcal{K}'$:

$$T_{\mathcal{K},s} := \{t \neq \epsilon | st \equiv_{\mathcal{K}} s\}$$

For all $s \in \mathcal{K}$ with $T_{\mathcal{K},s} = \emptyset$, Property (4.1) of Proposition 4.1 is obviously met. For all $s \in \mathcal{K}$ with $T_{\mathcal{K},s} \neq \emptyset$, pick arbitrary $t \in T$ and observe $st \equiv_{\mathcal{K}} s$. As s can be extended by t such that $st \in \mathcal{K}$ and st is Nerode equivalent to s, also st can be extended by t such that $stt \in \mathcal{K}$. Following up this deliberation we obtain $s(t)^* \subseteq \mathcal{K}$. Hence, for the ω-string $w = s(t)^\omega$, we have: $\exists\,(n_i)_{i\in\mathbb{N}_0}, n_{i+1} > n_i : w^{n_i} := s(t)^i \in \mathcal{K}$. Hence, $w \in \mathcal{K}^\infty$ and, as $\mathcal{K}$ is Y_{C}-live, $p_{\mathrm{YC}}(w) = p_{\mathrm{YC}}(s)p_{\mathrm{YC}}((t)^\omega) \in Y_{\mathrm{C}}^\omega$. As s is of finite length, it has to hold that $p_{\mathrm{YC}}((t)^\omega) \in Y_{\mathrm{C}}^\omega$, which implies $p_{\mathrm{YC}}(t) \neq \epsilon$. As s was chosen arbitrarily, $\mathcal{K}$ meets Property (4.1). □

Note that any sublanguage of a Y_{C}-live language is Y_{C}-live, too (Lemma 3.3). Due to the equivalence to Y_{C}-liveness, this equally holds for the above property (4.1). A Y_{C}-live sublanguage of an arbitrary language is achieved by allowing only finite sequences of transitions between states within a Y_{C}-less cycle. Unfortunately, in general, the *supremal* Y_{C}-live sublanguage of a given language does not exist. Given the family of all Y_{C}-live sublanguages of some language, the *supremal* Y_{C}-live sublanguage is *not* automatically given by their union, which we have shown in Proof 3.6 and can also be seen in the example in Figure 4.1.

Example 4.3
Again, we examine the automaton representation of $\mathcal{L}$ in Figure 4.1 a). A solution for avoid an infinite repetition of the indicated loop is to cancel the transition a or b closing the loop after an

arbitrary but finite number n of repetitions. For each fixed $n \in \mathbb{N}$, such a solution $\mathcal{K} \subseteq \mathcal{L}$ can easily be determined (see Figure 4.1 b) and c), for example). In contrast, the infinite union of all these solutions would lead to a language in which the n-wise repetition of all loops is turned into an *arbitrary* repetition (cf. definition of the Kleene-Closure). Thus, the infinite union of all Y_{C}-live sublanguages results in the original language $\mathcal{L}$, which is known to be non-Y_{C}-live. □

But, along with the problem, this example also makes evident how to resolve it. From an application point of view, even the *finite* iteration of Y_{C}-less cycles is undesirable, as it poses a back step on the path to the next Y_{C}-event. Hence, we propose to derive a so-called Y_{C}-Acyclic sublanguage that guarantees that a Y_{C}-less cycle of the original language is *never* closed.

Definition 4.1 (Y_{C}-Acyclic Sublanguage)
Let $\mathcal{L}$ be a regular language over Σ, and let $Y_{\mathrm{C}} \subseteq \Sigma$ be an alphabet. A string $t \in \Sigma^*$ is Y_{C}-Acyclic w.r.t. $\mathcal{L}$, if

$$\forall r, s \in \Sigma^*,\ r < t :\ (rs = t \wedge rs \equiv_{\mathcal{L}} r) \Rightarrow p_{\mathrm{YC}}(s) \neq \epsilon$$

where $\equiv_{\mathcal{L}}$ denotes the Nerode equivalence over Σ^* w.r.t. $\mathcal{L}$.

The language $\mathcal{K}$ is a Y_{C}-*Acyclic sublanguage of* $\mathcal{L}$ if

- $\mathcal{K} \subseteq \mathcal{L}$
- $\forall s \in \mathcal{K} :\ s$ is Y_{C}-Acyclic w.r.t $\mathcal{L}$

□

Note that in the above definition, different from Proposition 4.1, nerode equivalence w.r.t. another language $\mathcal{L}$ but not w.r.t. $\mathcal{K}$ itself is checked. This slight but important difference guarantees that $\mathcal{K}$ contains only strings that do not close a y_{C}-free cycle in $\mathcal{L}$, which is important for the existence of a *supremal* Y_{C}-Acyclic sublanguage. It is readily shown that a Y_{C}-Acyclic sublanguage w.r.t. some other language is always Y_{C}-live, see Proposition 4.2. In general, the reverse does not hold, i.e. Definition 4.1 indeed confines the family of Y_{C}-live languages.

Proposition 4.2
Let $\mathcal{K}$ be a Y_{C}-Acyclic sublanguage of a language $\mathcal{L}$ over the alphabet $\Sigma \supseteq Y_{\mathrm{C}}$. Then, $\mathcal{K}$ is Y_{C}-live. □

Proof
To prove that $\mathcal{K}$ is Y_{C}-live, we use Proposition 4.1 and show that property (4.1) holds for $\mathcal{K}$:
pick arbitrary $s \in \mathcal{K}$ and $\epsilon < t \in \Sigma^*$ such that $st \equiv_{\mathcal{K}} s$. Note that, in $\mathcal{K}$, s can be extended by t. As $st \equiv_{\mathcal{K}} s$, also st can be extended by t such that $stt \in \mathcal{K}$. Following up this deliberation, we get $st^n \in \mathcal{K}$ for arbitrary $n \in \mathbb{N}_0$, i.e. $st^* \in \mathcal{K}$. As $\mathcal{K} \subseteq \mathcal{L}$, it holds that $st^* \in \mathcal{L}$. As $\mathcal{L}$ is regular, $\mathcal{L}$ can

be partitioned into a finite set of nerode cells. Hence, $\exists\ n_1 \in \mathbb{N}_0, n_2 \in \mathbb{N}$ such that $st^{n_1}t^{n_2} \equiv_\mathcal{L} st^{n_1}$. As $\mathcal{K}$ is a Y_C-Acyclic sublanguage of $\mathcal{L}$, it holds that $p_{\mathrm{YC}}(t^{n_2}) \neq \epsilon$. Consequently, $p_{\mathrm{YC}}(t) \neq \epsilon$. Summing up, as s and t were chosen arbitrarily, we conclude:

$$\forall s \in \mathcal{K}:\ (\forall t \neq \epsilon \text{ with } st \in \mathcal{K})[st \equiv_\mathcal{K} s \Rightarrow p_{\mathrm{YC}}(t) \neq \epsilon].$$

I.e. property (4.1) is met for $\mathcal{K}$. Hence, according to Proposition (4.1), $\mathcal{K}$ is Y_C-live. □

Example 4.4
Consider the sublanguages $\mathcal{K}_1$ and $\mathcal{K}_2$ in Figure 4.1 b) and c) of the language $\mathcal{L}$ in Figure 4.1 a). While both, $\mathcal{K}_1$ and $\mathcal{K}_2$ are Y_C-live, only $\mathcal{K}_2$ is a Y_C-Acyclic sublanguage of $\mathcal{L}$. □

The least restrictive way to achieve a Y_C-Acyclic sublanguage of some language $\mathcal{L} \subseteq \Sigma^*$ is to remove only those strings $s\sigma \in \mathcal{L}$, $\sigma \in \Sigma$, that just close a Y_C-less cycle, i.e. s is Y_C-Acyclic w.r.t. $\mathcal{L}$, but not $s\sigma$. As a result, the *supremal* Y_C-Acyclic sublanguage of a language $\mathcal{L} \subseteq \Sigma^*$ is the set of *all* strings of $\mathcal{L}$ that are Y_C-Acyclic w.r.t. $\mathcal{L}$.

Proposition 4.3
Let $\mathcal{K}$ be a regular language over the alphabet $\Sigma \supseteq Y_\mathrm{C}$. Then,

$$Y_\mathrm{C}\text{Acyclic}(\mathcal{K}) := \{t \in \mathcal{K} \mid (\forall r, s \in \Sigma^*)[rs = t \wedge rs \equiv_\mathcal{K} r \Rightarrow s = \epsilon \vee p_{\mathrm{YC}}(s) \neq \epsilon]\} \tag{4.2}$$

is the *supremal* Y_C-Acyclic sublanguage w.r.t. $\mathcal{K}$. □

Proof Let $\mathcal{K}$ be a regular language over the alphabet $\Sigma \supseteq Y_\mathrm{C}$ and $\mathcal{K}_{Y_\mathrm{C}} := Y_\mathrm{C}\text{Acyclic}(\mathcal{K})$. We show that (a) $\mathcal{K}_{Y_\mathrm{C}}$ is a Y_C-Acyclic sublanguage w.r.t. $\mathcal{K}$ and (b) any Y_C-Acyclic sublanguage w.r.t. $\mathcal{K}$ is contained in $\mathcal{K}_{Y_\mathrm{C}}$.

(a) $\mathcal{K}_{Y_\mathrm{C}}$ is a Y_C-Acyclic sublanguage w.r.t. $\mathcal{K}$. Proof: Obviously, $\mathcal{K}_{Y_\mathrm{C}} \subseteq \mathcal{K}$ by definition of the operator Y_C-Acyclic(). We show

$$\forall s \in \mathcal{K}_{Y_\mathrm{C}}:\ (\forall t \neq \epsilon \text{ with } st \in \mathcal{K}_{Y_\mathrm{C}})[st \equiv_\mathcal{K} s \Rightarrow p_{\mathrm{YC}}(t) \neq \epsilon]\}.$$

Pick some arbitrary $s \in \mathcal{K}_{Y_\mathrm{C}}$ and consider the following set of extensions of s in $\mathcal{K}_{Y_\mathrm{C}}$:

$$T_{\mathcal{K}_{Y_\mathrm{C}},s} := \{t \neq \epsilon | st \in \mathcal{K}_{Y_\mathrm{C}} \wedge st \equiv_\mathcal{K} s\}$$

W.l.o.g. assume that $T_{\mathcal{K}_{Y_\mathrm{C}},s}$ is nonempty and pick some arbitrary $t \in T_{\mathcal{K}_{Y_\mathrm{C}},s}$. Hence, $st \in \mathcal{K}_{Y_\mathrm{C}}$ and $st \equiv_{\mathcal{K}_{Y_\mathrm{C}}} s$. Consequently, property (4.2) has to hold for st and thus $p_{\mathrm{YC}}(t) \neq \epsilon$. As s was chosen arbitrarily, we have:

$$\forall s \in \mathcal{K}_{Y_\mathrm{C}}:\ (\forall t \neq \epsilon \text{ with } st \in \mathcal{K}_{Y_\mathrm{C}})[st \equiv_\mathcal{K} s \Rightarrow p_{\mathrm{YC}}(t) \neq \epsilon]\},$$

i.e. $\mathcal{K}_{Y_\mathrm{C}}$ is a Y_C-Acyclic sublanguage w.r.t. $\mathcal{K}$.

(b) Let $\mathcal{K}'$ be a Y_C-Acyclic sublanguage w.r.t. $\mathcal{K}$. We show $\mathcal{K}' \subseteq \mathcal{K}_{Y_\mathrm{C}}$, i.e. $t \in \mathcal{K}' \Rightarrow t \in \mathcal{K}_{Y_\mathrm{C}}$. So, pick arbitrary $t \in \mathcal{K}'$. Consider an arbitrary concatenation of strings r and s such that $rs = t$. We show that the property

$$rs \equiv_\mathcal{K} r \Rightarrow s = \epsilon \vee p_{\mathrm{YC}}(s) \neq \epsilon \tag{4.3}$$

is fulfilled. Property (4.3) obviously holds if $rs \not\equiv_\mathcal{K} r$ or $s = \epsilon$. Now consider the nontrivial case $rs \equiv_\mathcal{K} r$ and $s \neq \epsilon$. Note that $\mathcal{K}'$ is a Y_C-Acyclic sublanguage w.r.t. $\mathcal{K}$ and $r \in \mathcal{K}'$, as $r \leq t$. Thus, for all extensions $\tilde{s}$ with $r\tilde{s} \in \mathcal{K}'$, it holds that $r\tilde{s} \equiv_\mathcal{K} r \Rightarrow p_{\mathrm{YC}}(\tilde{s}) \neq \epsilon$. In particular $p_{\mathrm{YC}}(s) \neq \epsilon$.
Hence, property (4.3) is fulfilled for arbitrary concatenations rs with $rs = t$, and we have $t \in \mathcal{K}_{Y_\mathrm{C}}$.
As $t \in \mathcal{K}'$ was chosen arbitrarily, it holds that $\mathcal{K}' \subseteq \mathcal{K}_{Y_\mathrm{C}}$. □

The supremal Y_C-Acyclic sublanguage of a language $\mathcal{L}$ is computed according to Proposition 4.3 by separating Y_C-Acyclic strings of $\mathcal{L}$ from those that are not. As this partition need not be as coarse as the Nerode-Equivalence over $\mathcal{L}$, the state space (and thus the complexity) of the canonical recognizer of Y_CAcyclic$(\mathcal{L})$ may be greater than that of $\mathcal{L}$, respectively. As a consequence, Y_CAcyclic$(\mathcal{L})$ cannot be achieved by simply erasing transitions the canonical recognizer of $\mathcal{L}$, in general.

Example 4.5
Reconsider Example 4.1. Indeed, $\mathcal{K}_2$ is the supremal Y_C-Acyclic sublanguage of $\mathcal{L}$. As can be seen, $\mathcal{K}_2$ is not retrieved by just canceling the transition labeled by u_P in the generator of $\mathcal{L}$, as this transition represents the last event of Y_C-Acyclic strings as well as non-Y_C-Acyclic strings. In order to achieve the supremal Y_C-Acyclic sublanguage of $\mathcal{L}$, state 2 has to be split in states 2 and 4, where state 2 is reached only from state 1 and state 4 is reached from state 3 only, see Figure 4.2.

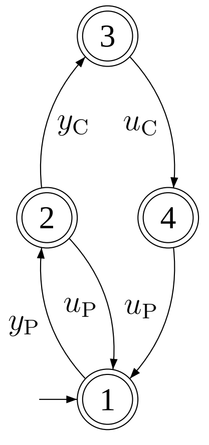


Figure 4.2: Transformation of the generator of $\mathcal{L}$

After this transformation introducing equivalent states, the result can now be obviously obtained by canceling the transition from state 2 to state 1. □

A graph based algorithmic construction of the supremal Y_{C}-Acyclic sublanguage is presented in section 4.2. Note that, as an obvious but useful property, all subsets of the supremal Y_{C}-Acyclic sublanguage are also Y_{C}-Acyclic and Y_{C}-live. Our I/O controller synthesis procedure computes the supremal Y_{C}-Acyclic sublanguage to yield an I/O controller that fulfills the admissibility condition (ii) in Definition 3.11.

4.2 Supremal Y_{C}-Acyclic Sublanguage: Graph-Based Computation

Let $\mathcal{K}$ be a regular language and Y_{C} be an alphabet. In this section, we provide a method to compute the finite automata representation of the supremal Y_{C}-Acyclic sublanguage $\mathcal{K}_1 := \mathrm{YcAcyclic}(\mathcal{K})$. Recall that a non-$Y_{\mathrm{C}}$-live string features nerode equivalence to at least one of its own strict prefixes and that the extension from this prefix to the string is free of Y_{C}-events. To identify such strings, we refine the informal notion of a Y_{C}-less cycle to the following definition, which has been derived from the notion of strongly connected components in [AHU75].

Definition 4.2 (Y_{C}-less Strongly Connected Components)
Let $G := (\Sigma, Q, \delta, q_0, Q_{\mathrm{m}})$ be a finite state automaton. We can partition Q into equivalence classes Q_i, $1 \le i \le |Q|$, such that states $q_1 \in Q$ and $q_2 \in Q$ are equivalent if and only if there is a path s_1 with $\delta(q_1, s_1) = q_2$ and $p_{\mathrm{YC}}(s_1) = \epsilon$ and a path s_2 with $\delta(q_2, s_2) = q_1$ and $p_{\mathrm{YC}}(s_2) = \epsilon$.
A state $q_i \in Q_i$ is denoted *entry state of* Q_i if $\delta(q, \sigma) = q_i$ for some $q \in Q - Q_i,\ \sigma \in \Sigma$.
A class Q_j of the above partition is called *Y_{C}-less strongly connected component (Y_{C}-less SCC)* if either $|Q_j| > 1$, or $Q_j = q_j$ and $\delta(q_j, \sigma) = q_j$ for some $\sigma \in \Sigma - Y_{\mathrm{C}}$.
Such class Q_j is called *strictly* Y_{C}-less SCC if additionally

$$\forall s \in \Sigma^*,\ q \in Q_j :\ \delta(q, s) \in Q_j \Rightarrow p_{\mathrm{YC}}(s) = \epsilon.$$

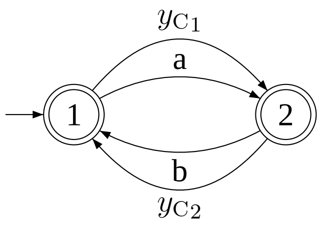
□

The absence of Y_{C}-less SCC's in an automaton G coincides with the Y_{C}-liveness of the language generated by G.

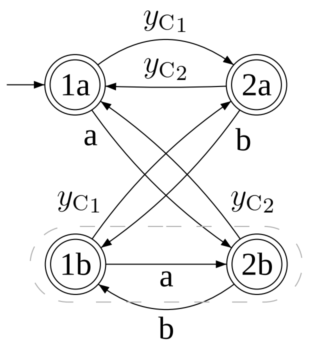
Theorem 4.1
Let $G = (\Sigma, Q, \delta, q_0, Q_\mathrm{m})$ be a (deterministic) finite state automaton and Y_C be an alphabet. $\mathcal{L}(G)$ is Y_C-live if and only if G is free of Y_C-less SCC's. □

Proof See Appendix A.2 □

Our procedure for finding the supremal Y_C-Acyclic sublanguage requires that all Y_C-less SCC's in the generator of the original language are strictly Y_C-less. This can be achieved by a simple transformation of the generator that does not change the generated and the marked language. We explain the transformation by a simple but representative example, see Figure 4.3.



(a) G_1: ambiguous Y_C-less SCC



(b) G_2: ambiguity resolved by state-splitting transformation

Figure 4.3: Transformation to achieve strictly Y_C-less SCC's

In automaton G_1 in Figure 4.3 a), the states 1 and 2 pose a Y_C-less SCC. However, the SCC can be traversed by the execution of Y_C-transitions without leaving the SCC. The automaton in Figure 4.3 b) generates the same language as the automaton in Figure 4.3 a). Now, the SCC $\{1b, 2b\}$ is strictly Y_C-less and is left whenever a Y_C-event occurs. This transformation is necessary whenever Y_C-transitions are in parallel to non-Y_C-transitions within a Y_C-less SCC. In the worst case, the state space is doubled by this transformation. However, in our approach, such transformation is never necessary because of the involved language structures.

To achieve the supremal Y_{C}-Acyclic sublanguage of some given language $\mathcal{K}$ according to Proposition 4.3, we proceed according to the following informal, but vivid description. An illustrating example is provided below.

Procedure: YCACYCLIC(G)

(1) The procedure starts with a state minimal automaton G that generates the language $\mathcal{K}$. We require a minimal state space such that any two paths lead to the same state iff the corresponding strings are nerode equivalent w.r.t. $\mathcal{K}$, which is helpful to examine all nerode equivalent strings in order to verify property (4.2).

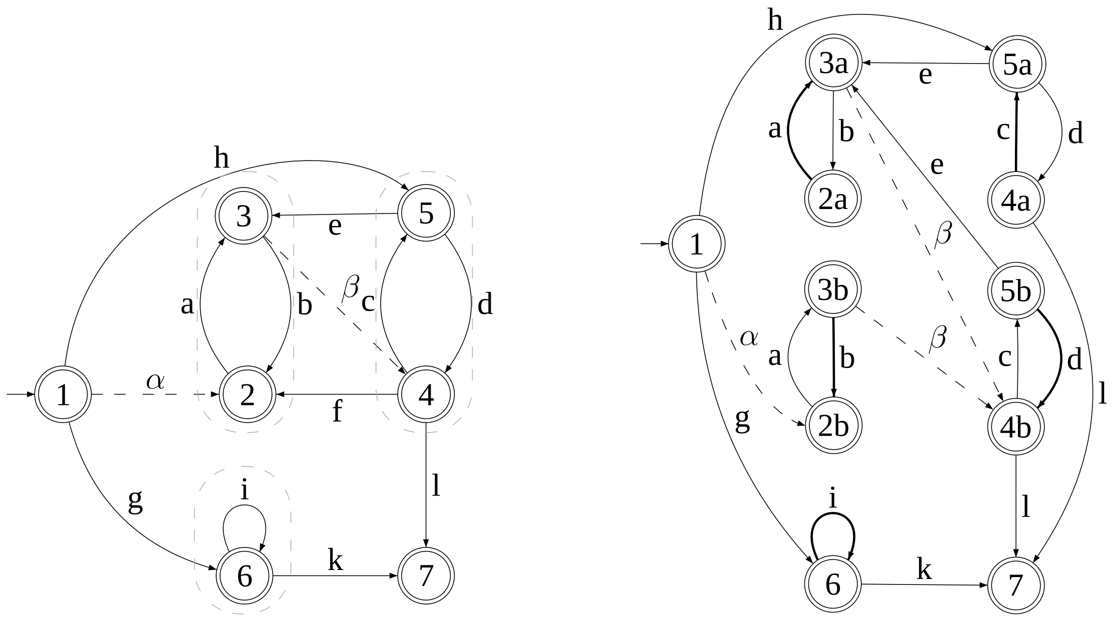
(2) Identify all Y_{C}-less SCC's of G. Technically, this is achieved by an efficient variant of depth-first search algorithm for finding SCC's presented in [AHU75]. If Y_{C}-less SCC's were found, proceed with (3). Else, G is the result.

(3) If not all Y_{C}-less SCC's are strictly Y_{C}-less, transform G as above. If there are Y_{C}-less SCC's with more than one entry state, transform G by duplicating the affected Y_{C}-less SCC's, such that each duplicate has a unique entry state, see also the below example. The idea for such transformation is provided in [JMRT08]. Note that the resulting automaton still generates $\mathcal{K}$.

(4) For each Y_{C}-less SCC, cancel transitions leading from a state of this Y_{C}-less SCC to its own entry state (denoted *back transitions*), unless the transition is triggered by a Y_{C}-Event. Each string in $\mathcal{K}$ corresponding to a path ending with the canceled transitions violates the property required for the elements of Y_{C}Acyclic$(\mathcal{K})$. However, in the resulting transformed automaton G' generating a sublanguage $\mathcal{K}' \subseteq \mathcal{K}$, there still might remain Y_{C}-less SCC's as subsets of the Y_{C}-less SCC's found in this iteration step. Hence, we set $G = G'$ and proceed with Step (2). At this point, it is interesting to note that, as transitions have been deleted, some strings in $\mathcal{K}'$ may now be nerode equivalent w.r.t. $\mathcal{K}'$ though *not* w.r.t. $\mathcal{K}$. However, as the corresponding automaton G' is still defined over the same state space Q of the automaton G, the states of G' still refer to the nerode cells of the original language $\mathcal{K}$, which is of interest. Hence it would be unwise to exhibit a state space minimization on G' before proceeding with step (2), as the result of the procedure would deviate from Y_{C}Acyclic$(\mathcal{K})$.

We illustrate the above steps by the following example.

Example 4.6
Consider the automaton G that generates a language $\mathcal{K}$ as depicted in Figure 4.4 a). The Y_{C}-events are $Y_{\mathrm{C}} = \{\alpha, \beta\}$.



(a) Automaton G with Y_{C}-events α and β.

(b) Transformation: Y_{C}-less SCC's with unique entry states

Figure 4.4: Example for computation of the supremal Y_{C}-Acyclic sublanguage

Note that $\mathcal{K}$ is not Y_{C}-live. E.g. the string $s = \alpha(ab)^*$ is contained in $\mathcal{K}$. Hence, $w = \alpha(ab)^\omega \in \mathcal{K}^\infty$ with $p_{Y\mathrm{C}}(w) = \alpha \notin Y_{\mathrm{C}}^\omega$. An automaton generating the supremal Y_{C}-Acyclic sublanguage is constructed by the procedure YCACYCLIC$(\mathcal{K})$:
Step (1). G is a state minimal realization of $\mathcal{K}$.
Step (2). The Y_{C}-less SCC's of G are highlighted in Figure 4.4 a) by gray dashed margins. These SCC's have to be broken by canceling transitions. Consider the Y_{C}-less SCC $\{2, 3\}$. It can be entered via the path $(1, \alpha, 2)$ or via the path $(1, h, 5)(5, e, 3)$. Hence, the states 2 and 3 are entry states of the SCC $\{2, 3\}$, and $(1, \alpha, 2)$ and $(5, e, 3)$ are called entry transitions. The least restrictive way to break the SCC {2,3} is to cancel the transition b after the occurrence of the string αa or to cancel a after the occurrence of heb. I.e. depending on the past string, the transitions a and b have to be either canceled or not. This means that the states 2 and 3 have to be split to be able to distinguish the respective cases. This motivates step (3).
Step (3). If a Y_{C}-less SCC has $n > 1$ entry states, then this SCC is replaced by n duplicates with

one unique entry state each, see Figure 4.4 b). E.g. the SCC $\{2, 3\}$ with the entry states 2 and 3 is replaced by an SCC $\{2a, 3a\}$ with the entry state $2a$ and an SCC {2b,3b} with the entry state $3b$. Entry transitions are shifted to the duplicate of the corresponding entry state, i.e. $(1, \alpha, 2)$ is replaced by $(1, \alpha, 2a)$, and $(5, e, 3)$ is replaced by $(5, e, 3b)$. All transitions having their origin in a state of the Y_C-less SCC are replaced by n duplicates. Note that also the SCC $\{4, 5\}$ has to be duplicated. Note that those states whose label only differs in the suffix “a” or “b” are equivalent, and the generated language is still $\mathcal{K}$.

Step (4). After the transformation in step (3), transitions that definitely have to be canceled are easily identified by transitions starting from a state of some Y_C-less SCC and leading back to the (unique) entry state of the same Y_C-less SCC. These transitions are highlighted in Figure 4.4 b) by bold edges. The resulting automaton G' after cancellation is shown in the figure below. To check if G' still contains Y_C-less SCC's, we return to:

Step(2). As can also be seen in the figure below, G' does not contain Y_C-less SCC's. Moreover, it is readily observed that G' generates the supremal Y_C-Acyclic sublanguage.

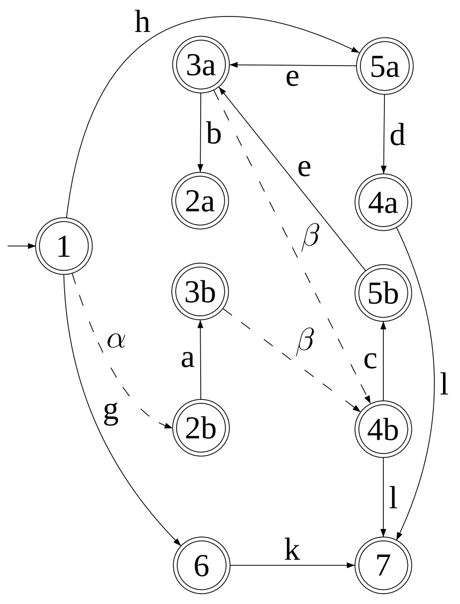


Figure 4.5: Result: automaton generating the supremal Y_C-Acyclic sublanguage

□

We informally show that the procedure YCACYCLIC(G) indeed leads to the supremal Y_C-acyclic sublanguage.

Proposition 4.4

If G generates the language $\mathcal{K}$ and marks the language $\mathcal{K}_m$, then the result G' := YCACYCLIC(G)

of the above procedure generates and marks the supremal Y_{C}-Acyclic sublanguage of $\mathcal{K}$ and $\mathcal{K}_m$, respectively. □

Proof (informal)

- *The result G' is obtained after a finite number of steps.* In step (3), the Y_{C}-less SCC's are duplicated according to their number of entry states. In step (4), however, the size of each Y_{C}-less SCC is reduced by at least one state, as the former entry states loose membership in the according SCC by the canceling of back transitions. Hence, all Y_{C}-less SCC's and their duplicates have to vanish after a finite number of iterations.

- *The result G' generates and marks a Y_{C}-Acyclic sublanguage of $\mathcal{K}$ and $\mathcal{K}_m$, respectively.* Note that G' is free of Y_{C}-less SCC's and hence, the language generated by G' is a Y_{C}-Acyclic sublanguage of $\mathcal{K}$ according to Theorem 4.1. For $\mathcal{K}_m$ consider Definition 4.1 and observe that[1] $s_1 \equiv_{\mathcal{K}_m} s_2 \Rightarrow s_1 \equiv_{\mathcal{K}} s_2$ to conclude that G' marks a Y_{C}-Acyclic sublanguage of $\mathcal{K}_m$.

- *$\mathcal{L}(G')$ and $\mathcal{L}_{\mathrm{m}}(G')$ are the supremal Y_{C}-Acyclic sublanguages of $\mathcal{K}$ and $\mathcal{K}_m$, respectively.* Note that, by step (3), all Y_{C}-less SCC's considered in step (4) are strictly Y_{C}-less. Hence it is ensured that, by canceling of the back transitions, only non-Y_{C}-Acyclic strings w.r.t. $\mathcal{K}$ and $\mathcal{K}_m$ are removed from $\mathcal{L}(G')$ and $\mathcal{L}_{\mathrm{m}}(G')$.

□

Accordingly, during I/O controller synthesis, the supremal Y_{C}-Acyclic sublanguage can be computed by the procedure YCACYCLIC(G) to account for admissibility condition (ii) in Definition 3.11.

Moreover, the controller synthesis procedure has to account for admissibility condition (i), for the I/O structure required in Definition 3.10 of the I/O controller and for the problem of partial observation, as the controller cannot directly observe the environment events Σ_{E}. These issues are treated by the next section.

4.3 Complete, Controllable and Normal Sublanguage

As mentioned before, the I/O controller synthesis procedure first computes full the closed-loop behaviour $\mathcal{L}_{\mathrm{CPE}} = \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ achieved by the solution (i.e. the controller to be

[1]This property holds for the generated and marked language of any automaton. The reverse direction need not hold, in general.

synthesized) and then extracts the I/O controller from it. Therefore, the closed-loop behaviour must meet the conditions *completeness*, *controllability* and *normality*.

Completeness of the closed-loop is a direct consequence of admissibility of the I/O controller, see Proposition 3.4.

Language controllability is a property that has been introduced by the SCT to describe those closed-loop languages that can be enforced on the plant by disabling only controllable events and is defined as follows.

Definition 4.3 (Controllability [RW87b])
Let $\mathcal{L} \subseteq \Sigma^*$ be a prefix-closed language, and let $\Sigma_{\mathrm{uc}} \subseteq \Sigma$ be the set of uncontrollable events. The language $\mathcal{K} \subseteq \mathcal{L}$ is said to be *controllable* w.r.t. $\mathcal{L}$ and the set of uncontrollable events Σ_{uc} if

$$\overline{\mathcal{K}}\Sigma_{\mathrm{uc}} \cap \mathcal{L} \subseteq \overline{\mathcal{K}}.$$

□

This means that the occurrence of uncontrollable events in $\mathcal{L}$ has to be accepted by $\mathcal{K}$. In our framework, the I/O controller has to accept the inputs U_{C} and Y_{P} and has no direct access at all to the environment events Σ_{E}. For the language of the full closed loop, this implies that it must be controllable w.r.t. $\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ and the set of uncontrollable events $\Sigma_{\mathrm{uc}} := U_{\mathrm{C}} \cup Y_{\mathrm{P}} \cup \Sigma_{\mathrm{E}}$.

Language normality is a property that guarantees that a closed-loop behavior is achieved by a controller also in the case of partial observation and defined as follows:

Definition 4.4 (Normality (e.g. [Won08]))
Let $\mathcal{K} \subseteq \mathcal{L}$ be prefix-closed languages over the alphabet Σ, and let $p_{\mathrm{o}}\colon \Sigma^* \to \Sigma_{\mathrm{o}}^*$ be the natural projection, with $\Sigma_{\mathrm{o}} \subseteq \Sigma$. Then, $\mathcal{K}$ is said to be normal w.r.t. $\mathcal{L}$ and p_{o} if

$$\mathcal{K} = p_{\mathrm{o}}^{-1}(p_{\mathrm{o}}(\mathcal{K})) \cap \mathcal{L}$$

□

This property can be rewritten as $\mathcal{K} = p_{\mathrm{o}}(\mathcal{K}) \parallel \mathcal{L}$. The following proposition states that an I/O controller can only enforce normal sublanguages on the I/O plant.

Proposition 4.5
Let $\mathcal{K} \subseteq \mathcal{L}$ be languages over the alphabet Σ, and let $p_{\mathrm{o}}\colon \Sigma^* \to \Sigma_{\mathrm{o}}^*$ be the natural projection to the alphabet $\Sigma_{\mathrm{o}} \subseteq \Sigma$. A language $\mathcal{K}_o \subseteq \Sigma_o^*$ such that

$$\mathcal{K}_o \parallel \mathcal{L} = \mathcal{K} \tag{4.4}$$

exists if and only if $\mathcal{K}$ is normal w.r.t. p_{o} and $\mathcal{L}$. □

Proof
if. If $\mathcal{K}$ is normal w.r.t. p_o and $\mathcal{L}$, then $\mathcal{K}_o$ is obviously given by $p_\mathrm{o}(\mathcal{K})$.
only if. We show that $\mathcal{K}_o$ does not exist if $\mathcal{K}$ is not normal w.r.t. p_o and $\mathcal{L}$. First observe that $\mathcal{K}_o \parallel \mathcal{L} = p_\mathrm{o}^{-1}(\mathcal{K}_o) \cap \mathcal{L}$ and, as $\mathcal{K}$ is not normal w.r.t. p_o and $\mathcal{L}$, the following inequality holds:

$$p_\mathrm{o}^{-1}(p_\mathrm{o}(\mathcal{K})) \cap \mathcal{L} \supset \mathcal{K} \tag{4.5}$$

We distinguish the cases $\mathcal{K}_o \supseteq p_\mathrm{o}(\mathcal{K})$ and $\mathcal{K}_o \subset p_\mathrm{o}(\mathcal{K})$:

- $\mathcal{K}_o \supseteq p_\mathrm{o}(\mathcal{K})$: then, $p_\mathrm{o}^{-1}(\mathcal{K}_o) \cap \mathcal{L} \supset \mathcal{K}$ follows directly from the above inequality (4.5)
- $\mathcal{K}_o \subset p_\mathrm{o}(\mathcal{K})$:
  Proof by contradiction: Assume $\mathcal{K}_o \parallel \mathcal{L} = \mathcal{K}$. Note that here Definition 2.5 of the synchronous composition evaluates to $\mathcal{K}_o \parallel \mathcal{L} = p_\mathrm{o}^{-1}(\mathcal{K}_o) \cap \mathcal{L}$ and observe

$$\begin{aligned}
p_\mathrm{o}^{-1}(\mathcal{K}_o) \cap \mathcal{L} &= \mathcal{K} \\
&\Downarrow \\
p_\mathrm{o}[p_\mathrm{o}^{-1}(\mathcal{K}_o) \cap \mathcal{L})] &= p_\mathrm{o}(\mathcal{K}) \\
&\Downarrow \text{ Lemma A.2} \\
p_\mathrm{o}(p_\mathrm{o}^{-1}(\mathcal{K}_o)) \cap p_\mathrm{o}(\mathcal{L}) &\supseteq p_\mathrm{o}(\mathcal{K}) \\
&\Downarrow \\
\mathcal{K}_o \cap p_\mathrm{o}(\mathcal{L}) &\supseteq p_\mathrm{o}(\mathcal{K}) \\
&\Downarrow \ (\mathcal{K}_o \subset p_\mathrm{o}(\mathcal{K}) \subseteq p_\mathrm{o}(\mathcal{L})) \\
\mathcal{K}_o &\supseteq p_\mathrm{o}(\mathcal{K}).
\end{aligned}$$

  As the last consequence contradicts $\mathcal{K}_o \subset p_\mathrm{o}(\mathcal{K})$, we have $\mathcal{K}_o \parallel \mathcal{L} \neq \mathcal{K}$.

Consequently, $\mathcal{K}_o$ does not exist. □

Hence, an I/O controller with the language $\mathcal{L}_\mathrm{CP}$ over Σ_CP enforcing the closed-loop behaviour $\mathcal{L}_\mathrm{CPE}$ on $\mathcal{L}_\mathrm{C} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}$ (both over Σ_CPE) does not exist unless $\mathcal{L}_\mathrm{CPE}$ is normal w.r.t. $\mathcal{L}_\mathrm{C} \parallel \mathcal{L}_\mathrm{PE} \parallel \mathcal{L}_\mathrm{E}$ and $p_\mathrm{CP}\colon \Sigma_\mathrm{CPE}^* \to \Sigma_\mathrm{CP}^*$.

Remark 4.1
In particular, also observable languages (see e.g. [Won08] for definition) cannot be enforced by an I/O controller unless they are normal languages as above. □

According to the above considerations, during I/O controller synthesis, a complete, controllable and normal sublanguage has to be calculated. Unfortunately, results on the existence and computation of the *supremal* complete, controllable and normal sublanguage have not been presented up to now in DES literature. On the other hand, the supremal complete and controllable sublanguage as well as an efficient algorithm for its computation are presented in [KGM92]. Moreover, the supremal normal sublanguage is presented in [BGK+90] as a compact formula that can be evaluated without iteration. In the I/O controller synthesis algorithm presented in the next section, a complete, controllable and normal sublanguage is derived by iteration of (I) computing the supremal complete and controllable sublanguage according to [KGM92] and (II) computing the supremal normal sublanguage according to [BGK+90] until a fixpoint is reached. Hence, the resulting sublanguage is guaranteed to be complete, controllable and normal. It is denoted by the operator $(\cdot)^{cCN}$.

Remark 4.2
In the TU example and all examples considered during the development of this framework, this procedure led to nontrivial results after a small number of steps. However, neither finite-step conversion nor supremality of the result are considered in this contribution. □

4.4 I/O Controller Synthesis Procedure

Let $\Pi := (\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$ be an I/O controller synthesis problem according to Definition 3.12. To illustrate the details of each step of the controller synthesis, we introduce the conceptional example of a simple machine.

Example 4.7
Simple Machine. We consider a production cell, whose complex tasks are internally controlled, such that a very simple view from the outside is provided for superposed logic control: whenever not busy, the machine reports rdy, and the operator can start or stop the machine. After the stp command, the machine remains ready. After the $start$ command, the machine starts some process, during which a shared resource is requested. If the resource is provided, the machine successfully finishes the process and reports rdy again. This logical behaviour can directly be modeled as an I/O plant $\mathcal{S}_{\mathrm{PE}} := (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$. We identify the plant-I/O port that models interaction with the operator with $(U_{\mathrm{P}}, Y_{\mathrm{P}}) := (\{stp, start\}, \{rdy\})$, interaction with the environment is captured by $(U_{\mathrm{E}}, Y_{\mathrm{E}}) := (\{pack, nack\}, \{req\})$. By the unobservable environment-event req, we model the machines requirement for the shared resource. For a plant description that is independent from the environment, we introduce the unobservable environment-events $nack$ (negative acknowledge) in case of unavailable shared resource and $pack$ denoting that the resource is provided. The possible behaviour $\mathcal{L}_{\mathrm{PE}}$ can be modeled as depicted in the following automaton model.

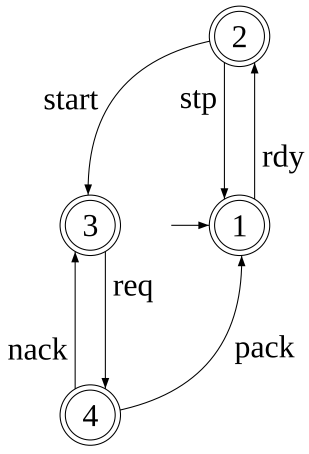
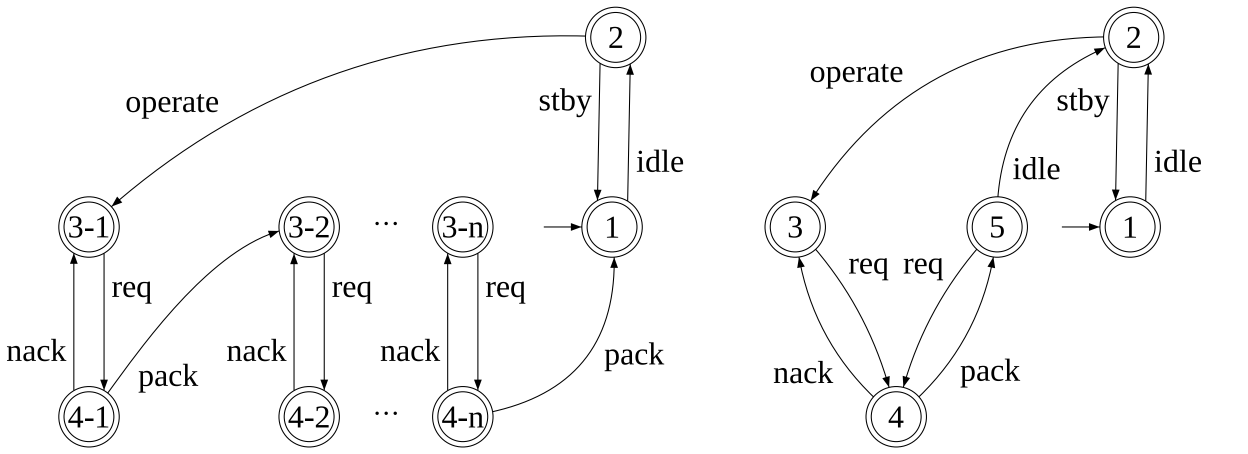


Figure 4.6: I/O plant model of a simple machine

Liveness and constraints:
Temporarily, assume minimal constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$, which corresponds to arbitrary external configurations. Note that $\mathcal{L}_{\mathrm{PE}}$ is complete but not Y_{P}-live w.r.t. these constraints; as the plant model is designed independently from the environment and the constraint $\mathcal{S}_{\mathrm{E}}$ is minimal, the extreme case that the shared resource is *never* provided when requested is included. The resulting livelock is represented by a $(req\ nack)$ loop between states 3 and 4 in the automaton model. It corresponds to the nonempty set of strings $\Sigma_{\mathrm{PE}}^{*}(req\ nack)^{*} \cap \mathcal{L}_{\mathrm{PE}}$. Hence, the limit $(\mathcal{L}_{\mathrm{PE}})^{\infty}$ contains ω-strings of the sort $w := s(req\ nack)^{\omega},\ s \in \Sigma_{\mathrm{PE}}^{*}$ with $p_{\mathrm{YP}}(w) \notin Y_{\mathrm{P}}^{\omega}$.
Thus liveness has to be discussed by the introduction of reasonable constraints. One approach is to restrict $\mathcal{S}_{\mathrm{P}}$ to a guideline for the operator or controller such that $\mathcal{L}_{\mathrm{PE}}$ is Y_{P}-live w.r.t. $\mathcal{S}_{\mathrm{P}}$ and an arbitrary i.e. minimal environmental configuration $\mathcal{S}_{\mathrm{E}}$. The least restrictive constraint $\mathcal{S}_{\mathrm{P}}$ that can be found is given by $\mathcal{S}_{\mathrm{P}} = (\Sigma_{\mathrm{P}}, \overline{(rdy\ stp)^{*}})$, i.e. $start$ is never enabled.
A more reasonable approach is to relax the requirement of Y_{P}-liveness to configurations in which a shared resource is provided after a finite amount of requests. This configuration is given by a minimal constraint $\mathcal{S}_{\mathrm{P}}$ and any environment constraint $(\Sigma_{\mathrm{E}}, \mathcal{L}_{\mathrm{E}})$ with $\Sigma_{\mathrm{E}}^{*}(req\ nack)^{\omega} \cap {\mathcal{L}_{\mathrm{E}}}^{\infty} = \emptyset$
. For this example, we choose $\mathcal{S}_{\mathrm{E}} := (\Sigma_{\mathrm{E}}, \overline{(req\ pack)^{*}})$, which means that shared resources are always provided when requested. Note that in practice, this constraint usually is not fulfilled a priori. Our approach addresses this fact by passing on the constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$ as *requirements* to the hierarchy of superposed controllers. This can be seen in Equations 5.1 and 5.2 in Theorem 5.1.
Safety Specification:
Assume that a standby for maintenance has to be possible after a certain amount of n processes. To formulate a respective specification for the external behaviour, we introduce the set

$U_{\mathrm{C}} := \{operate, stby\}$ of desired modes of operation and the desired feedback $Y_{\mathrm{C}} := \{idle\}$. The desired effect of the modes on the environment can be described by the system $\mathcal{S}_{\mathrm{specCE}} := (U_{\mathrm{C}}, Y_{\mathrm{C}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{specCE}})$, see Figure 4.7 a). From now on, for a better illustration of the computation of Y_{C}-live sublanguages, we discuss the specification depicted in Figure 4.7 b), which requires the possibility for a standby after an unspecified amount of processes.



(a) Standby after n processes (b) Standby after undefined number of processes

Figure 4.7: Specification $\mathcal{S}_{\mathrm{specCE}}$ for a simple machine

This specification $\mathcal{S}_{\mathrm{specCE}}$ together with the I/O plant model $\mathcal{S}_{\mathrm{PE}}$, minimal constraints $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$ and the exclusion of $nack$ by the above constraint $\mathcal{S}_{\mathrm{E}}$ completes the I/O controller synthesis problem of this example: $\Pi := (\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$. $\square$

For being a solution for Π, the system $\mathcal{S}_{\mathrm{CP}}$ has to be an admissible I/O controller that enforces $\mathcal{S}_{\mathrm{specCE}}$ on $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$. In the following, we propose an algorithmic procedure to compute the minimal restrictive solution $\mathcal{S}_{\mathrm{CP}}$ to Π within the family of so-called Y_{C}-Acyclic sublanguages.

I/O Controller Synthesis Algorithm (I/O CSA)
Let $\Pi := (\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$ be an I/O controller synthesis problem, where $\mathcal{S}_{\mathrm{specCE}}$ is an I/O plant model of the desired external closed-loop behaviour. The system $\mathcal{S}_{\mathrm{CP}} = (\Sigma_{\mathrm{CP}}, \mathcal{L}_{\mathrm{CP}})$ is computed as follows.

(I) Restrict the behaviour of the full closed loop:

$$\mathcal{K}_0 := \mathcal{L}_{\mathrm{PE}c} \parallel \mathcal{L}_{\mathrm{P}} \parallel \overline{(Y_{\mathrm{P}}(\epsilon + Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{specCE}}$$

where $\mathcal{L}_{\mathrm{PE}c}$ is the plant under constraints $\mathcal{L}_{\mathrm{PE}c} := \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$.

(II) Compute the supremal Y_{C}-Acyclic sublanguage:

$$\mathcal{K}_1 := Y_{\mathrm{C}}\mathrm{Acyclic}(\mathcal{K}_0)$$

(III) Define the events $\Sigma_{\mathrm{uc}} := U_{\mathrm{C}} \cup Y_{\mathrm{P}} \cup \Sigma_{\mathrm{E}}$ uncontrollable and the events $\Sigma_{\mathrm{o}} := \Sigma_{\mathrm{CP}}$ observable. Compute a complete, controllable and normal sublanguage of $\mathcal{K}_1$ w.r.t. $\mathcal{L}_{\mathrm{PE}c}$, Σ_{uc} and Σ_{o}:

$$\mathcal{K}_2 := (\mathcal{K}_1)^{(cCN)}$$

(IV) Compute the projection to the controller alphabet:

$$\mathcal{K}_{\mathrm{CP}} := p_{\mathrm{CP}}(\mathcal{K}_2)$$

(V) Add error behaviour to make Y_{P} and U_{C} free in $\mathcal{L}_{\mathrm{CP}}$:

$$\mathcal{L}_{\mathrm{CP}} := \mathcal{K}_{\mathrm{CP}} \cup \mathcal{K}_{\mathrm{CP}}^{err}$$

with $\mathcal{K}_{\mathrm{CP}}^{err} := (\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \cup \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}})\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$, see Definition 4.5.

We consider the first step of the presented algorithm.

Step (I): Desired behaviour of the full closed loop
By parallel composition, we restrict the possible plant behaviour $\mathcal{L}_{\mathrm{PE}c}$ to the language format $\overline{(Y_{\mathrm{P}}(\epsilon + Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})}$ (required by Definition 3.10 of the I/O controller), to the constraint $\mathcal{L}_{\mathrm{P}}$ (required by the admissibility condition (i) in Definition 3.11) and to the safety specification $\mathcal{L}_{\mathrm{specCE}}$. Note that, conversely, any superlanguage of $\mathcal{K}_0$ inevitably leads to violation of one of these properties.

Example 4.8
Simple Machine. The result of this step for the problem Π is depicted in the following figure.

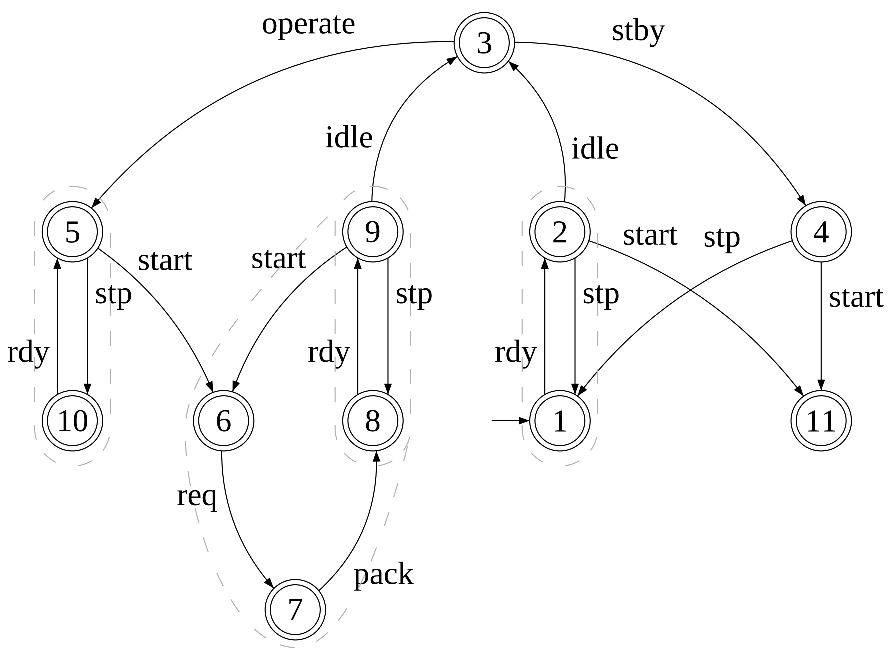


Figure 4.8: Desired closed loop behaviour of a simple machine

Observe the livelocks indicated by the dashed grey cycles. Hence, this candidate for the full closed loop behaviour violates the Y_{C}-liveness property required by admissibility in Definition 3.11. □

The next step of the synthesis algorithm addresses Y_{C}-liveness.

Step II) Computation of Y_{C}-live sublanguages The supremal Y_{C}-Acyclic sublanguage of $\mathcal{K}_0$ is computed according to Definition 4.3. Note that any sublanguage of $Y_{\mathrm{C}}\mathrm{Acyclic}(\mathcal{L})$ is also a Y_{C}-Acyclic (and thus Y_{C}-live) sublanguage of $\mathcal{L}$. Hence, the restriction in the following step does not compromise this property.

Example 4.9
Simple Machine. The supremal Y_{C}-Acyclic sublanguage of the desired behaviour as seen in Figure 4.8 is shown in the following figure.

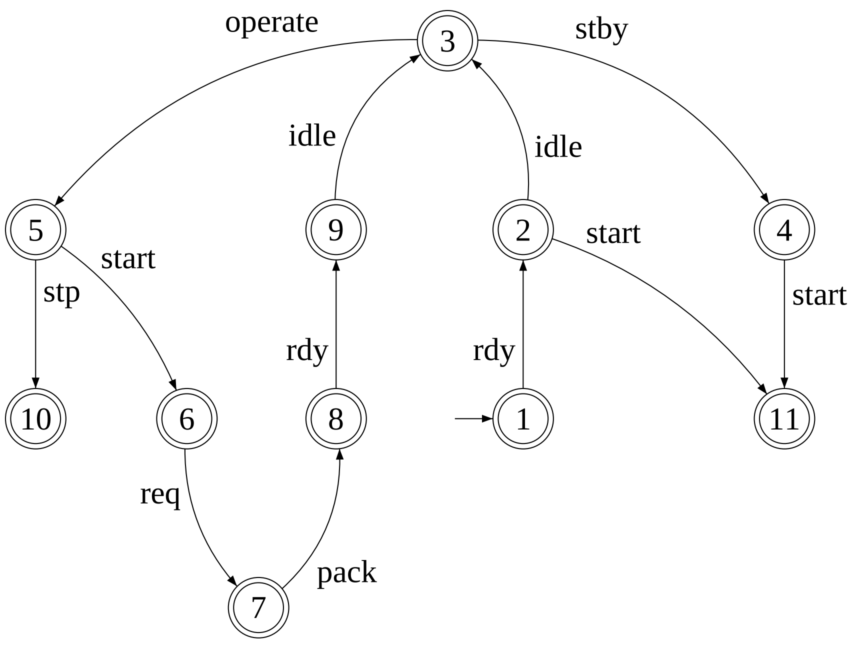


Figure 4.9: Simple machine: supremal Y_{C}-Acyclic sublanguage of the desired behaviour

In this intermediate result, completeness is violated by the deadlock states 10 and 11. Additionally, controllability fails in state 10 due to a missing rdy-transition. Moreover both, the normality and the controllability condition require a transition with the uncontrollable and unobservable event req in state 11. □

Step III) Computation of a complete, controllable and normal sublanguage By this step, a full closed-loop behaviour $\mathcal{K}_2$ is obtained, that can be realized by an admissible I/O controller, see Section 4.3.

Example 4.10
Simple Machine. The full closed-loop behaviour that is achieved for the simple machine is represented by the automaton in the subsequent figure.

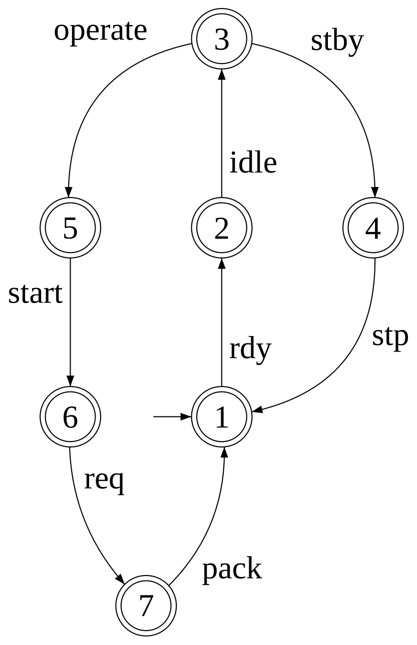


Figure 4.10: Simple machine: full closed-loop behaviour

□

The following steps are concerned with educing the I/O controller from $\mathcal{K}_2$.

Step IV) Compute the projection to Σ_{CP}. By this, the unobservability of the environment alphabet for the controller is taken into account. The normality achieved by step III) guarantees that the I/O controller will enforce $\mathcal{K}_2$ in the closed loop.

Example 4.11
Simple Machine. For the simple machine example, the projection to Σ_{CP} yields the result shown in Figure 4.11

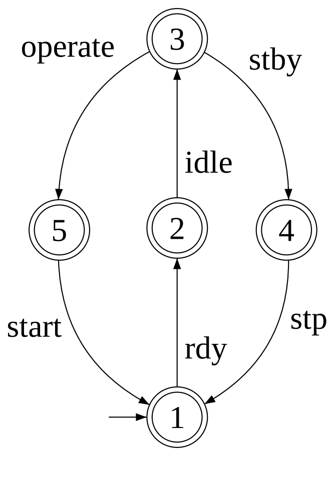


Figure 4.11: Simple machine: projection to Σ_{CP}

□

Step V) Make Y_{P} and U_{C} free. Note that Y_{P} is not necessarily free in $\mathcal{K}_{\mathrm{CP}}$, as a string s of $\mathcal{K}_{\mathrm{CP}}$ can only be extended by an event $\nu_{\mathrm{P}} \in Y_{\mathrm{P}}$ if there exists a corresponding string $s'\nu_{\mathrm{P}}$ in the plant behaviour $\mathcal{L}_{\mathrm{PE}}$ with $p_{\mathrm{CP}}(s') = s$, i.e. if $s'\nu_{\mathrm{P}}$ is possible plant behaviour. Similarly, U_{C} need not be free in $\mathcal{K}_{\mathrm{CP}}$, as the constraint $\mathcal{S}_{\mathrm{C}}$, whose language is a component of $\mathcal{L}_{\mathrm{PE}c}$, might here and there exclude the occurrence of U_{C}-events by its controller-I/O port property. To formally account for these Y_{P}- and U_{C}-events that do not occur in the closed-loop behavior, we insert the strategic error behavior $\mathcal{K}_{\mathrm{CP}}^{err}$ that does not contribute to the closed-loop behaviour, i.e. $\mathcal{K}_{\mathrm{CP}}^{err} \parallel (\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c}) = \varnothing$. The construction of $\mathcal{K}_{\mathrm{CP}}^{err}$ is based on the following definition.

Definition 4.5
Given a language $\mathcal{K}$ and an alphabet Σ, the language $\mathcal{K}^{\Sigma}$ is defined as follows:

$$\mathcal{K}^{\Sigma} := \{s\sigma,\ \sigma \in \Sigma \mid (\exists\sigma' \in \Sigma)[s\sigma' \in \mathcal{K} \wedge s\sigma \notin \mathcal{K}]\},$$

□

Note that $\mathcal{K}^{\Sigma} \subseteq \mathcal{K}\Sigma$ and $\mathcal{K}^{\Sigma} \cap \mathcal{K} = \varnothing$. In in the error behaviour $\mathcal{K}_{\mathrm{CP}}^{err}$ in Step V) of the I/O controller synthesis algorithm, $\mathcal{K}^{Y_{\mathrm{P}}}$ identifies all strings in $\mathcal{K}$ that can be extended by at least one but not *any* y_{P}-event without leaving $\mathcal{K}$ and adds the missing y_{P}-events not accepted in $\mathcal{K}$. The role of $\mathcal{K}^{U_{\mathrm{C}}}$ is analogous.

Example 4.12
For the simple machine, we obtain $\mathcal{K}_{\mathrm{CP}}^{err} = \varnothing$, as already in the automaton in Figure 4.11 each Y_{P}-event, i.e. the only y_{P}-event rdy, is accepted after the u_{P}-events stp and $start$. Also, each u_{C}-event

$(stby,\ operate)$ is accepted after the y_{C}-event $idle$. Hence, Step V) preserves the result shown in Figure 4.11.

The Transport Unit, however, provides an example with non-empty error-behaviour, see Figure 3.11. □

The following important lemma shows that the extension by $\mathcal{K}_{\mathrm{CP}}^{err}$ does not affect the closed-loop behaviour under constraints:

Lemma 4.1
Let $\Pi := (\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$ be an I/O controller synthesis problem according to Definition 3.12 and let $\mathcal{K}_{\mathrm{CP}}$ and $\mathcal{L}_{\mathrm{CP}}$ be constructed according to Steps IV and V of the I/O Controller Synthesis Algorithm, respectively. Then, it holds that

$$\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c} = \mathcal{K}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c}.$$

□

Interestingly, for the proof of this lemma, the normality property of $\mathcal{K}_2$ (I/O CSA, Step III) is needed.

Proof See Appendix A.2. □

By the above lemma, it is ensured that the extension by Y_{P}- and U_{C}-events does not extend the closed-loop behaviour, in particular no undesired behaviour is added. This is an important fact, needed to prove the following statement, which is one of the main results of our work. We are now able to state that the above algorithm leads to a solution of Π.

Theorem 4.2 (Solution for the I/O Controller Synthesis Problem)
Let $\Pi := (\mathcal{S}_{\mathrm{PE}}, \mathcal{S}_{\mathrm{C}}, \mathcal{S}_{\mathrm{P}}, \mathcal{S}_{\mathrm{E}}, \mathcal{S}_{\mathrm{specCE}})$ be an I/O controller synthesis problem according to Definition 3.12, where $\mathcal{S}_{\mathrm{specCE}}$ is an I/O plant (describing the desired external closed-loop behaviour). If the language $\mathcal{L}_{\mathrm{CP}}$ is constructed according to the I/O controller Synthesis Algorithm applied to Π, then:
$\mathcal{S}_{\mathrm{CP}} := (\Sigma_{\mathrm{CP}}, \mathcal{L}_{\mathrm{CP}})$ is a solution for Π.

□

Proof (outline)
At this place, we provide an outline on the items that have to be shown. The complete proof is found in the appendix, see Appendix A.2. We have to show the following items:

1) $\mathcal{S}_{\mathrm{CP}}$ is an I/O controller:

(i) $\mathcal{S}_{\mathrm{CP}}$ is a system with $\Sigma_{\mathrm{CP}} = \Sigma_{\mathrm{C}} \dot{\cup} \Sigma_{\mathrm{P}}$, $\Sigma_{\mathrm{C}} := U_{\mathrm{C}} \dot{\cup} Y_{\mathrm{C}}$, $\Sigma_{\mathrm{P}} := U_{\mathrm{P}} \dot{\cup} Y_{\mathrm{P}}$;

(ii) $(U_{\mathrm{C}}, Y_{\mathrm{C}})$ and $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ are a plant- and a controller-I/O port for $\mathcal{S}_{\mathrm{CP}}$, respectively;

(iii) $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^{*}(Y_{\mathrm{P}}Y_{\mathrm{C}}U_{\mathrm{C}}U_{\mathrm{P}})^{*})^{*}}$;

(iv) $\mathcal{L}_{\mathrm{CP}}$ is complete.

2) $\mathcal{S}_{\mathrm{CP}}$ is admissible to $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$:

(i) $p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{P}}$

(ii) $\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$ is Y_{C}-live w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$

3) $\mathcal{S}_{\mathrm{CP}}$ enforces $\mathcal{S}_{\mathrm{specCE}}$ on $\mathcal{S}_{\mathrm{PE}c}$. $\square$

Example 4.13
Simple Machine. By Theorem 4.2, it is shown that the automaton in Figure 4.11 indeed represents an admissible I/O controller that enforces the specification shown in Figure 4.7 $\square$

Chapter 5

Hierarchical Control System

Suppose we are provided an overall system consisting of n plant components that in their particular configuration interact via shared resources. According to the previous chapters, the individual plant components can be modeled independently (no shared events) as I/O plants with corresponding constraints, see Chapter 3. This step leads to one I/O plant per component and corresponding constraints; i.e. for $i = 1..n$, $\mathcal{S}_{\mathrm{PE}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{PE}i})$, $\mathcal{S}_{\mathrm{P}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, \mathcal{L}_{\mathrm{Pi}})$ and $\mathcal{S}_{\mathrm{E}i} = (U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{Ei}})$ where each I/O plant $\mathcal{S}_{\mathrm{PE}i}$ is complete and $Y_{\mathrm{P}i}$-live w.r.t. the constraints $\mathcal{S}_{\mathrm{P}i}$ and $\mathcal{S}_{\mathrm{E}i}$. As at this stage all components are regarded as independent entities with no synchronization built in, all alphabets $\Sigma_{\mathrm{P}i} := U_{\mathrm{P}i} \dot{\cup} Y_{\mathrm{P}i}$ and $\Sigma_{\mathrm{E}i} := U_{\mathrm{E}i} \dot{\cup} Y_{\mathrm{E}i}$ are disjunct.

For each component, a local I/O controller can be designed as in Chapter 4 according to an individual specification $\mathcal{S}_{\mathrm{specCE}_i}$.

Example 5.1
Transport Unit. Consider a chain of an arbitrary number of TU's as in Figure 5.1. Each single TU can be provided with a local I/O controller as in Figure 3.11 designed according to the previous chapters.

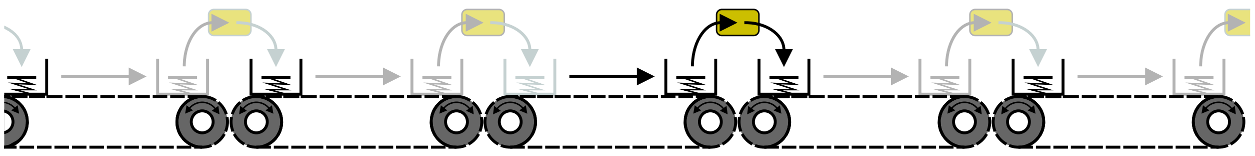


Figure 5.1: Chain of transport units

□

By Theorem 3.1, the overall system is still given as a set of n I/O plants, where the I/O-plant model of each *controlled* component is given as the external closed loop of the uncontrolled I/O plant and its I/O controller.

However, due to their particular configuration, the components usually interact via shared resources, and most control objectives explicitly involve the cooperation of plant components. By the next section, we enable control of the concurrent behaviour of a group of interacting components. Based on this result, Section 5.2 provides a guidance how to alternate hierarchical control and subsystem composition to achieve an overall hierarchy as in Figure 1.11. The core results of this chapter have been published in [PMS06] and [PMS07a].

5.1 Control of Composed Systems

For convenience, we consider groups of only two components $\mathcal{S}_{\mathrm{PE}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{PE}i})$, $i \in \{1, 2\}$; the behaviour of each group is described by a system architecture as seen in Figure 5.2 a).

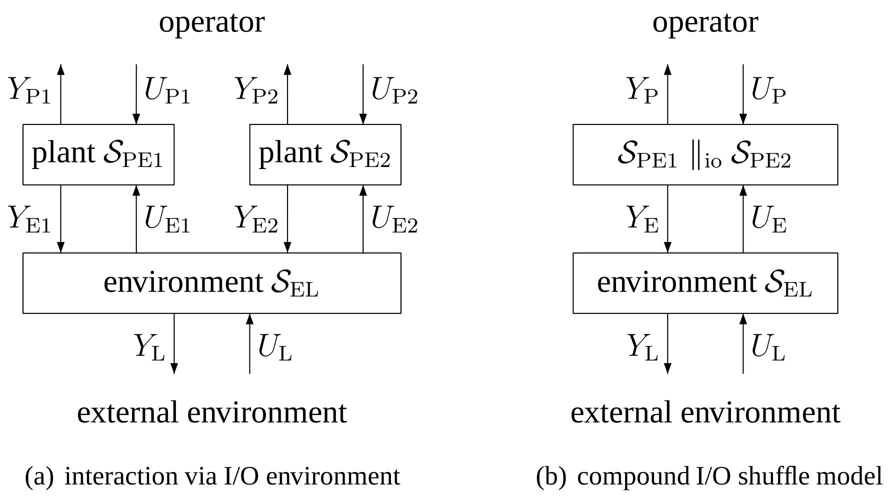


Figure 5.2: Group of I/O plants with I/O environment

First, the individual and independently designed I/O plants $\mathcal{S}_{\mathrm{PE}i}$ are composed by a shuffle composition to technically form a compound model as in Figure 5.2 b), see Section 5.1.1. Then, the restrictions due to interaction of the components with each other and with the external configuration are described in a subordinate environment model, see Section 5.1.2. We show that controller design for the resulting compound of the group and the interaction model can be conducted according to the previous chapters with liveness of the individual interacting components preserved.

5.1.1 I/O Shuffle

To technically capture the behaviour of both plants $\mathcal{S}_{\mathrm{PE}i}$ in one mathematical model, we introduce the *I/O shuffle* operation $\mathcal{S}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{S}_{\mathrm{PE2}}$. It is based on the ordinary shuffle product (parallel composition under absence of shared events), but restricted by the additional condition $\mathcal{L}_{\mathrm{io}}$ on the ordering of input-output event-pairs and extended by a well-defined error behaviour $\mathcal{L}_{\mathrm{err}}$. The latter accounts for situations where $\mathcal{L}_{\mathrm{io}}$ is violated, i.e. a measurement event from the one plant component is replied to by a control event to the other plant component.

Definition 5.1 (I/O shuffle)
Given two I/O plants $\mathcal{S}_{\mathrm{PE}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{PE}i})$, $i \in \{1,2\}$, the *I/O shuffle* $\mathcal{S}_{\mathrm{PE}} = \mathcal{S}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{S}_{\mathrm{PE2}}$ is defined as a tuple $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$, where:

(i) $U_{\mathrm{P}} := U_{\mathrm{P1}} \dot\cup U_{\mathrm{P2}}$, $Y_{\mathrm{P}} := Y_{\mathrm{P1}} \dot\cup Y_{\mathrm{P2}}$, $U_{\mathrm{E}} := U_{\mathrm{E1}} \dot\cup U_{\mathrm{E2}}$, $Y_{\mathrm{E}} := Y_{\mathrm{E1}} \dot\cup Y_{\mathrm{E2}}$;

(ii) $\mathcal{L}_{\mathrm{PE}} := [(\mathcal{L}_{\mathrm{PE1}} \parallel \mathcal{L}_{\mathrm{PE2}}) \cap \mathcal{L}_{\mathrm{io}}] \cup \mathcal{L}_{\mathrm{err}} := \mathcal{L}_{\parallel} \cup \mathcal{L}_{\mathrm{err}}$, with

(iii) $\mathcal{L}_{\mathrm{io}} := \overline{(\Sigma_{\mathrm{PE1}}\Sigma_{\mathrm{PE1}} + \Sigma_{\mathrm{PE2}}\Sigma_{\mathrm{PE2}})^{*}}$ and

(vi) $\mathcal{L}_{\mathrm{err}} := \cup_{i=1}^{4} \mathcal{L}_{\mathrm{i}}$ with
$\mathcal{L}_{1} := (\mathcal{L}_{\parallel} Y_{\mathrm{P1}} \cap \mathcal{L}_{\parallel}) U_{\mathrm{P2}}$,
$\mathcal{L}_{2} := (\mathcal{L}_{\parallel} Y_{\mathrm{P2}} \cap \mathcal{L}_{\parallel}) U_{\mathrm{P1}}$,
$\mathcal{L}_{3} := (\mathcal{L}_{\parallel} Y_{\mathrm{E1}} \cap \mathcal{L}_{\parallel}) U_{\mathrm{E2}}$,
$\mathcal{L}_{4} := (\mathcal{L}_{\parallel} Y_{\mathrm{E2}} \cap \mathcal{L}_{\parallel}) U_{\mathrm{E1}}$;

□

Observe that the I/O shuffle of prefix-closed systems $\mathcal{S}_{\mathrm{PE}i}$ is prefix-closed (without any effect, the languages $\mathcal{L}_{\mathrm{i}}$ in item (vi) can be replaced by $\overline{\mathcal{L}_{\mathrm{i}}}$). It is readily shown that the I/O shuffle indeed is a shuffle composition in the sense that the behaviour of neither plant is restricted, i.e.

$$\text{for } i = 1,2: \ \mathcal{L}_{\mathrm{PE}i} \subseteq \mathcal{L}_{\mathrm{PE}},$$

see Appendix, Lemma A.7. Moreover, the I/O shuffle retains the I/O structure of its arguments:

Proposition 5.1
If $\mathcal{S}_{\mathrm{PE}i}$, $i \in \{1,2\}$ are I/O plants, so is $\mathcal{S}_{\mathrm{PE}} = \mathcal{S}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{S}_{\mathrm{PE2}}$. □

Proof We show that $\mathcal{S}_{\mathrm{PE}}$ provides all I/O-plant properties.

(i) It is obvious that $\mathcal{S}_{\mathrm{PE}}$ is a system.

(ii) $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ is plant-I/O port for $\mathcal{S}_{\mathrm{PE}}$:

- $\Sigma = W\dot{\cup}U_{\mathrm{P}}\dot{\cup}Y_{\mathrm{P}}$ with $W = \Sigma_{\mathrm{PE}} - U_{\mathrm{P}} - Y_{\mathrm{P}} = U_{\mathrm{E}}\dot{\cup}Y_{\mathrm{E}}$
- $\mathcal{L}_{\mathrm{PE}} \subseteq \overline{(W^*(Y_{\mathrm{P}}U_{\mathrm{P}})^*)^*}$
- $(\forall s \in \Sigma^* Y_{\mathrm{P}},\ \mu \in U_{\mathrm{P}})\ [s \in \mathcal{L}_{\mathrm{PE}} \Rightarrow s\mu \in \mathcal{L}_{\mathrm{PE}}]$. Proof: Pick $s = ry_{\mathrm{P1}} \in \mathcal{L}_{\mathrm{PE}}$ with $y_{\mathrm{P1}} \in Y_{\mathrm{P1}}$. Consider two cases: (1) $s \in \mathcal{L}_{\mathrm{err}} - \mathcal{L}_{\|}$. Then, $s\mu \in \mathcal{L}_{\mathrm{err}}\ \forall \mu \in U_{\mathrm{P}}$ by construction of $\mathcal{L}_{\mathrm{err}}$. (2) $s \in \mathcal{L}_{\|}$ As $s = ry_{\mathrm{P1}}$, $s \in \mathcal{L}_{\|}Y_{\mathrm{P}} \cap \mathcal{L}_{\|}$. Hence, $s\mu \in \mathcal{L}_1 \subseteq \mathcal{L}_{\mathrm{err}} \subseteq \mathcal{L}_{\mathrm{PE}}\ \forall \mu \in U_{\mathrm{P2}}$. Note furthermore that $p_{\mathrm{PE1}}(s) = p_{\mathrm{PE1}}(r)y_{\mathrm{P1}} \in \mathcal{L}_{\mathrm{PE1}}$. As U_{P1} is a free input of $\mathcal{S}_{\mathrm{PE1}}$, it holds that $p_{\mathrm{PE1}}(r)y_{\mathrm{P1}}\mu \in \mathcal{L}_{\mathrm{PE1}}$ for all $\mu \in U_{\mathrm{P1}}$. As $\mathcal{L}_{\mathrm{PE1}}$ and $\mathcal{L}_{\mathrm{PE2}}$ do not share events, $p_{\mathrm{PE2}}(s\mu) = p_{\mathrm{PE2}}(s) \in \mathcal{L}_{\mathrm{PE2}}$ and consequently $s\mu \in \mathcal{L}_{\mathrm{PE1}} \parallel \mathcal{L}_{\mathrm{PE2}}$. Note also that $s\mu = ry_{\mathrm{P1}}\mu \in \mathcal{L}_{\mathrm{io}}$ and thus $s\mu \in \mathcal{L}_{\|} \subseteq \mathcal{L}_{\mathrm{PE}}$
  In summary, $\forall s = ry_{\mathrm{P1}} \in \mathcal{L}_{\mathrm{PE}}$ with $y_{\mathrm{P1}} \in Y_{\mathrm{P1}}$ and $\forall \mu \in U_{\mathrm{P}}$, it holds that $s\mu \in \mathcal{L}_{\mathrm{PE}}$.
  Note that the same holds $\forall s = ry_{\mathrm{P2}} \in \mathcal{L}_{\mathrm{PE}}$ with $y_{\mathrm{P2}} \in Y_{\mathrm{P2}}$ for symmetry reasons.

$(U_{\mathrm{E}}, Y_{\mathrm{E}})$ is plant-I/O port for $\mathcal{S}_{\mathrm{PE}}$: as above.

□

Accordingly, the constraints $\mathcal{S}_{\mathrm{P}i}$ and $\mathcal{S}_{\mathrm{E}i}$ of the individual I/O plants have to be composed such that the liveness properties are represented correctly by the I/O shuffle under the compound constraints. We merge the constraints of the individual plants by the standard shuffle product restricted to the I/O structure $\mathcal{L}_{\mathrm{io}}$ of Definition 5.1. This way, the resulting constraint (that has to be met by the superposed controller) also includes the avoidance of the error-behaviour $\mathcal{L}_{\mathrm{err}}$.

Proposition 5.2
Let $\mathcal{S}_{\mathrm{PE1}}$ and $\mathcal{S}_{\mathrm{PE2}}$ be I/O-plants, and let $\mathcal{L}_{\mathrm{Pi}},\ \mathcal{L}_{\mathrm{Ei}},\ i \in \{1,2\}$ be constraints. Then, for the I/O-shuffle $\mathcal{S}_{\mathrm{PE}} = (\Sigma_{\mathrm{PE}}, \mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}})$ and the constraints $\mathcal{L}_{\mathrm{P}} := (\mathcal{L}_{\mathrm{P1}} \parallel \mathcal{L}_{\mathrm{P2}}) \cap \mathcal{L}_{\mathrm{io}}$ and $\mathcal{L}_{\mathrm{E}} := (\mathcal{L}_{\mathrm{E1}} \parallel \mathcal{L}_{\mathrm{E2}}) \cap \mathcal{L}_{\mathrm{io}}$, it holds that

$$\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} = \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\|} \parallel \mathcal{L}_{\mathrm{E}},$$

i.e. the error behaviour $\mathcal{L}_{\mathrm{err}}$ is avoided under the compound constraints. □

Proof Note that $\mathcal{L}_{\mathrm{PE}} = \mathcal{L}_{\|} \cup \mathcal{L}_{\mathrm{err}}$. Thus

$$\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} =^{Lem.A.1} (\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\|} \parallel \mathcal{L}_{\mathrm{E}}) \cup (\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{err}} \parallel \mathcal{L}_{\mathrm{E}}).$$

Note that, in the above synchronous compositions we have $\Sigma_{\mathrm{P}}, \Sigma_{\mathrm{E}} \subseteq \Sigma_{\mathrm{PE}}$, i.e. all events are shared events and hence

$$\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{err}} \parallel \mathcal{L}_{\mathrm{E}} = \mathcal{L}_{\mathrm{P}} \cap \mathcal{L}_{\mathrm{err}} \cap \mathcal{L}_{\mathrm{E}}.$$

As $\mathcal{L}_{\mathrm{P}} \subseteq \mathcal{L}_{\mathrm{io}}$ and $\mathcal{L}_{\mathrm{E}} \subseteq \mathcal{L}_{\mathrm{io}}$, we get

$$\mathcal{L}_{\mathrm{P}} \cap \mathcal{L}_{\mathrm{err}} \cap \mathcal{L}_{\mathrm{E}} = (\mathcal{L}_{\mathrm{P}} \cap \mathcal{L}_{\mathrm{io}}) \cap \mathcal{L}_{\mathrm{err}} \cap (\mathcal{L}_{\mathrm{E}} \cap \mathcal{L}_{\mathrm{io}}) = \mathcal{L}_{\mathrm{P}} \cap \mathcal{L}_{\mathrm{err}} \cap \mathcal{L}_{\mathrm{io}} \cap \mathcal{L}_{\mathrm{E}}.$$

Observe the language structure of $\mathcal{L}_{\mathrm{err}}$: $\mathcal{L}_{\mathrm{err}} \subseteq \Sigma^*_{\mathrm{PE}}(Y_{\mathrm{P1}}U_{\mathrm{P2}} + Y_{\mathrm{P2}}U_{\mathrm{P1}} + Y_{\mathrm{E1}}U_{\mathrm{E2}} + Y_{\mathrm{E2}}U_{\mathrm{E1}})$ to conclude that $\mathcal{L}_{\mathrm{err}} \cap \mathcal{L}_{\mathrm{io}} = \emptyset$. Thus,

$$\mathcal{L}_{\mathrm{P}} \cap \mathcal{L}_{\mathrm{err}} \cap \mathcal{L}_{\mathrm{io}} \cap \mathcal{L}_{\mathrm{E}} = \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{err}} \parallel \mathcal{L}_{\mathrm{E}} = \emptyset$$

and finally

$$(\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\parallel} \parallel \mathcal{L}_{\mathrm{E}}) \cup (\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{err}} \parallel \mathcal{L}_{\mathrm{E}}) = \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\parallel} \parallel \mathcal{L}_{\mathrm{E}}.$$

□

The following proposition states that the constraints of the individual plants indeed can be lifted to the compound plant by the (standard) shuffle product.

Proposition 5.3
Let $\mathcal{S}_{\mathrm{PE1}}$ and $\mathcal{S}_{\mathrm{PE2}}$ be I/O-plants, and let $\mathcal{L}_{\mathrm{Pi}},\ \mathcal{L}_{\mathrm{Ei}},\ i \in \{1,2\}$ be constraints. If $\mathcal{S}_{\mathrm{PE1}}$ is complete and Y_{P}-live w.r.t. $\mathcal{L}_{\mathrm{P1}}$ and $\mathcal{L}_{\mathrm{E1}}$, and if $\mathcal{S}_{\mathrm{PE2}}$ is complete and Y_{P}-live w.r.t. $\mathcal{L}_{\mathrm{P2}}$ and $\mathcal{L}_{\mathrm{E2}}$, then the I/O-shuffle $\mathcal{S}_{\mathrm{PE}} = (\Sigma_{\mathrm{PE}}, \mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}})$ is complete and Y_{P}-live w.r.t. $\mathcal{L}_{\mathrm{P}} := (\mathcal{L}_{\mathrm{P1}} \parallel \mathcal{L}_{\mathrm{P2}}) \cap \mathcal{L}_{\mathrm{io}}$ and $\mathcal{L}_{\mathrm{E}} := (\mathcal{L}_{\mathrm{E1}} \parallel \mathcal{L}_{\mathrm{E2}}) \cap \mathcal{L}_{\mathrm{io}}$. □

Proof
$\mathcal{S}_{\mathrm{PE}} = (\Sigma_{\mathrm{PE}}, \mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}})$ is complete w.r.t. $\mathcal{L}_{\mathrm{P}}$ and $\mathcal{L}_{\mathrm{E}}$:
Pick some arbitrary $s \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$. Regarding Proposition 5.2, $\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} = \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\parallel} \parallel \mathcal{L}_{\mathrm{E}}$. Thus, $p_{\mathrm{PE1}}(s) \in p_{\mathrm{PE1}}(\mathcal{L}_{\parallel}) = \mathcal{L}_{\mathrm{PE1}}$ and $p_{\mathrm{PE2}}(s) \in p_{\mathrm{PE2}}(\mathcal{L}_{\parallel}) = \mathcal{L}_{\mathrm{PE2}}$. As $\mathcal{L}_{\mathrm{P1}} \subseteq \mathcal{L}_{\mathrm{P}}$ and $\mathcal{L}_{\mathrm{E1}} \subseteq \mathcal{L}_{\mathrm{E}}$, $p_{\mathrm{PE1}}(s) \in \mathcal{L}_{\mathrm{P1}} \parallel \mathcal{L}_{\mathrm{PE1}} \parallel \mathcal{L}_{\mathrm{E1}}$ and $p_{\mathrm{PE2}}(s) \in \mathcal{L}_{\mathrm{P2}} \parallel \mathcal{L}_{\mathrm{PE2}} \parallel \mathcal{L}_{\mathrm{E2}}$. Consider the following two cases:

(i) $s \in \Sigma^*_{\mathrm{PE}} Y_{\mathrm{P1}}$: As $\mathcal{L}_{\mathrm{PE}}$ is an I/O-plant, $s\mu \in \mathcal{L}_{\mathrm{PE}}$ for all $\mu \in U_{\mathrm{P}}$. In particular, this holds for all $\mu \in U_{\mathrm{P1}}$. As $p_{\mathrm{PE1}}(s) \in \mathcal{L}_{\mathrm{PE1}}$ and $\mathcal{L}_{\mathrm{PE1}}$ is complete w.r.t. $\mathcal{L}_{\mathrm{P1}}$, there exists $\hat{\mu} \in U_{\mathrm{P1}}$ such that $p_{\mathrm{P1}}(p_{\mathrm{PE1}}(s)\hat{\mu}) \in \mathcal{L}_{\mathrm{P1}}$. As $s\hat{\mu} \in \mathcal{L}_{\mathrm{PE}}$, as $p_{\mathrm{E}}(s\mu) = p_{\mathrm{E}}(s) \in \mathcal{L}_{\mathrm{E}}$ and as $p_{\mathrm{P}}(s\mu) = p_{\mathrm{P}}(s)\mu \in \mathcal{L}_{\mathrm{P}}$ (because $\mathcal{L}_{\mathrm{P1}} \subseteq \mathcal{L}_{\mathrm{P}}$), it follows that $s\hat{\mu} \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$.

(ii) $s \in \Sigma^*_{\mathrm{PE}} Y_{\mathrm{E1}}$: analogous to case (i): As $\mathcal{L}_{\mathrm{PE}}$ is an I/O-plant, $s\mu \in \mathcal{L}_{\mathrm{PE}}$ for all $\mu \in U_{\mathrm{E}}$. In particular, this holds for all $\mu \in U_{\mathrm{E1}}$. As $p_{\mathrm{PE1}}(s) \in \mathcal{L}_{\mathrm{PE1}}$ and $\mathcal{L}_{\mathrm{PE1}}$ is complete w.r.t. $\mathcal{L}_{\mathrm{P1}}$, there exists $\hat{\mu} \in U_{\mathrm{E1}}$ such that $p_{\mathrm{E1}}(p_{\mathrm{PE1}}(s)\hat{\mu}) \in \mathcal{L}_{\mathrm{E1}}$. As $s\hat{\mu} \in \mathcal{L}_{\mathrm{PE}}$, as $p_{\mathrm{P}}(s\mu) = p_{\mathrm{P}}(s) \in \mathcal{L}_{\mathrm{P}}$ and as $p_{\mathrm{E}}(s\mu) = p_{\mathrm{E}}(s)\mu \in \mathcal{L}_{\mathrm{E}}$ (because $\mathcal{L}_{\mathrm{E1}} \subseteq \mathcal{L}_{\mathrm{E}}$), it follows that $s\hat{\mu} \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$.

(iii) $s \in \Sigma^*_{\mathrm{PE}}(U_{\mathrm{E1}} \cup U_{\mathrm{P1}})$: As $\mathcal{L}_{\mathrm{PE1}}$ is complete w.r.t. $\mathcal{L}_{\mathrm{P1}}$ and $\mathcal{L}_{\mathrm{E1}}$, there exists $\nu \in Y_{\mathrm{E1}} \cup Y_{\mathrm{P1}}$ such that $p_{\mathrm{PE1}}(s\nu) \in \mathcal{L}_{\mathrm{PE1}}$. As $\mathcal{L}_{\mathrm{PE1}}$ and $\mathcal{L}_{\mathrm{PE2}}$ do not share events, $p_{\mathrm{PE2}}(s\nu) = p_{\mathrm{PE2}}(s) \in \mathcal{L}_{\mathrm{PE2}}$ and thus $s\nu \in \mathcal{L}_{\mathrm{PE1}} \parallel \mathcal{L}_{\mathrm{PE2}}$. Note that also $s\nu \in \mathcal{L}_{\mathrm{io}}$. Thus, $s\nu \in \mathcal{L}_{\parallel} \subseteq \mathcal{L}_{\mathrm{PE}}$. As $\mathcal{L}_{\mathrm{P1}} \subseteq \mathcal{L}_{\mathrm{P}}$, $\mathcal{L}_{\mathrm{P2}} \subseteq \mathcal{L}_{\mathrm{P}}$, $\mathcal{L}_{\mathrm{E1}} \subseteq \mathcal{L}_{\mathrm{E}}$ and $\mathcal{L}_{\mathrm{E2}} \subseteq \mathcal{L}_{\mathrm{E}}$, it holds that $s\nu \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$.

(iv) The cases $s \in \Sigma^*_{\mathrm{PE}} Y_{\mathrm{P2}},\ s \in \Sigma^*_{\mathrm{PE}} Y_{\mathrm{E2}}$ and $s \in \Sigma^*_{\mathrm{PE}}(U_{\mathrm{E2}} \cup U_{\mathrm{P2}})$ are included due to symmetry reasons.

Hence, for all $s \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$, there exists $\sigma \in \Sigma_{\mathrm{PE}}$ such that $s\sigma \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$.

$\mathcal{S}_{\mathrm{PE}} = (\Sigma_{\mathrm{PE}}, \mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}})$ is Y_{P}-live w.r.t. $\mathcal{L}_{\mathrm{P}}$ and $\mathcal{L}_{\mathrm{E}}$:
Pick $w \in (\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}})^{\infty}$ and observe:

$$[\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}]^{\infty} =^{\text{Prop.5.2}} [\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\parallel} \parallel \mathcal{L}_{\mathrm{E}}]^{\infty} =$$
$$[\mathcal{L}_{\mathrm{P}} \parallel ((\mathcal{L}_{\mathrm{PE1}} \parallel \mathcal{L}_{\mathrm{PE2}}) \cap \mathcal{L}_{\mathrm{io}}) \parallel \mathcal{L}_{\mathrm{E}}]^{\infty} \subseteq [\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE1}} \parallel \mathcal{L}_{\mathrm{PE2}} \parallel \mathcal{L}_{\mathrm{E}}]^{\infty} =$$
$$[(\mathcal{L}_{\mathrm{P1}} \parallel \mathcal{L}_{\mathrm{PE1}} \parallel \mathcal{L}_{\mathrm{E1}}) \parallel (\mathcal{L}_{\mathrm{P2}} \parallel \mathcal{L}_{\mathrm{PE2}} \parallel \mathcal{L}_{\mathrm{E2}})]^{\infty}$$

According to Lemma A.5, it holds that $p_{\mathrm{PE1}}(w) \in (\mathcal{L}_{\mathrm{P1}} \parallel \mathcal{L}_{\mathrm{PE1}} \parallel \mathcal{L}_{\mathrm{E1}})^{\infty}$ and/or $p_{\mathrm{PE2}}(w) \in (\mathcal{L}_{\mathrm{P2}} \parallel \mathcal{L}_{\mathrm{PE2}} \parallel \mathcal{L}_{\mathrm{E2}})^{\infty}$. Hence, $p_{\mathrm{YP1}}(p_{\mathrm{PE1}}(w)) = p_{\mathrm{YP1}}(w) \in Y_{\mathrm{P1}}^{\omega}$ and/or $p_{\mathrm{YP2}}(p_{\mathrm{PE2}}(w)) = p_{\mathrm{YP2}}(w) \in Y_{\mathrm{P2}}^{\omega}$. In general, $p_{\mathrm{YP}}(w) \in Y_{\mathrm{P}}^{\omega}$. □

Hence, a compound I/O plant model for the independent behaviour of the two plant components is established, together with constraints that properly capture the conditions for liveness of the individual plant components.

Remark 5.1
In this thesis, we consider an I/O-shuffle that leads to a prefix-closed result in case of prefix-closed arguments. In practice, composed plants often feature the persistent guarantee of its components to (sooner or later) alternately *issue events – a property that usually cannot be enforced by control action but, instead, is naturally given by the composed plant. In particular in the case of control tasks that require the* alternate *operation of the involved plant components, a composed plant model based on the above I/O-shuffle can lead to over-restrictive results for the superposed controller. Hence, ongoing research includes an I/O-shuffle that formally expresses the ability of alternation by a non-prefix-closed result* $\mathcal{S}_{\mathrm{PE}} = \mathcal{S}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{S}_{\mathrm{PE2}}$ *such that*

$$w \in \mathcal{L}_{\mathrm{PE}}{}^{\infty} \Rightarrow p_{\mathrm{PE1}}(w) \in \mathcal{L}_{\mathrm{PE1}}{}^{\infty} \;\wedge\; p_{\mathrm{PE2}}(w) \in \mathcal{L}_{\mathrm{PE2}}{}^{\infty}$$

□

We proceed with modeling the interaction of the plants composed in the I/O-shuffle via a common environment model, called *I/O environment*.

5.1.2 I/O Environment

Technically, the I/O environment is a system, that is connected to an I/O plant via the port $(U_{\mathrm{E}}, Y_{\mathrm{E}})$, see Figure 5.2 b). Therefore, $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ has to be a controller-I/O port of the I/O environment. The I/O environment is used to describe two distinct kinds of interaction.

Internal interaction. The port $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ has to be a controller-I/O port of the I/O environment, as it is connected to the plant-I/O port $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ of the respective I/O plant. Via this port, the environment model can disable sequences of environment events that are not possible due to the concurrent behaviour of both plants, e.g. if both plants share resources among each other. Interacting discrete event systems often feature concurrent behaviour meaning that the liveness property of the individual plant components is lost in the compound behaviour due to conflicts in the interaction. In our framework, such situations (that have to be avoided by control) are captured by the I/O environment: seen from the I/O plant, the environment poses a constraint that is able and likely to violate the environment constraints $\mathcal{S}_{\mathrm{E}i}$ necessary for liveness of the plants $\mathcal{S}_{\mathrm{PE}i}$.

External interaction. Furthermore, the I/O environment forwards those sequences of environment events that concern the interaction of one or both plants with the *remaining* environment to the plant-I/O port $(Y_{\mathrm{L}}, U_{\mathrm{L}})$ that is connected with the external configuration. This is the case if e.g. the compound shares a resource with another group of plant components.

As a technical consequence of these considerations, we define the environment model to be of the same I/O structure as a controller.

Definition 5.2 (I/O environment)
An *I/O environment* is a tuple $\mathcal{S}_{\mathrm{EL}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{EL}})$, where:

(i) $(\Sigma_{\mathrm{EL}}, \mathcal{L}_{\mathrm{EL}})$ is a system with $\Sigma_{\mathrm{EL}} := U_{\mathrm{E}} \dot\cup Y_{\mathrm{E}} \dot\cup U_{\mathrm{L}} \dot\cup Y_{\mathrm{L}}$;

(ii) $(U_{\mathrm{E}}, Y_{\mathrm{E}})$ and $(U_{\mathrm{L}}, Y_{\mathrm{L}})$ are a controller- and a plant-I/O port, respectively;

(iii) $\mathcal{L}_{\mathrm{EL}} \subseteq \overline{((Y_{\mathrm{E}} U_{\mathrm{E}})^* (Y_{\mathrm{E}} Y_{\mathrm{L}} U_{\mathrm{L}} U_{\mathrm{E}})^*)^*}$;

(iv) $\mathcal{L}_{\mathrm{EL}}$ is complete.

□

Example 5.2
Transport Unit. Consider a chain of an arbitrary number of TU's, numbered alphabetically from left to right. To design a control hierarchy, we begin with compounding groups of two TU's, e.g. TU A and TU B. Note that the two plant models do not share events; the membership of each event to the respective component is indicated by the suffixes $\_A$ and $\_B$ in the event labels, e.g. $idle\_A$ and $idle\_B$. As indicated above, each locally controlled TU is abstracted by its specification (e.g. as in Figure 3.9), so first the I/O shuffle of the specifications of two transport units is computed.

The environment model $\mathcal{S}_{\mathrm{EL}} = (\Sigma_{\mathrm{EL}}, \mathcal{L}_{\mathrm{EL}})$ for the resulting module AB is designed in two steps, see Figure 5.3.

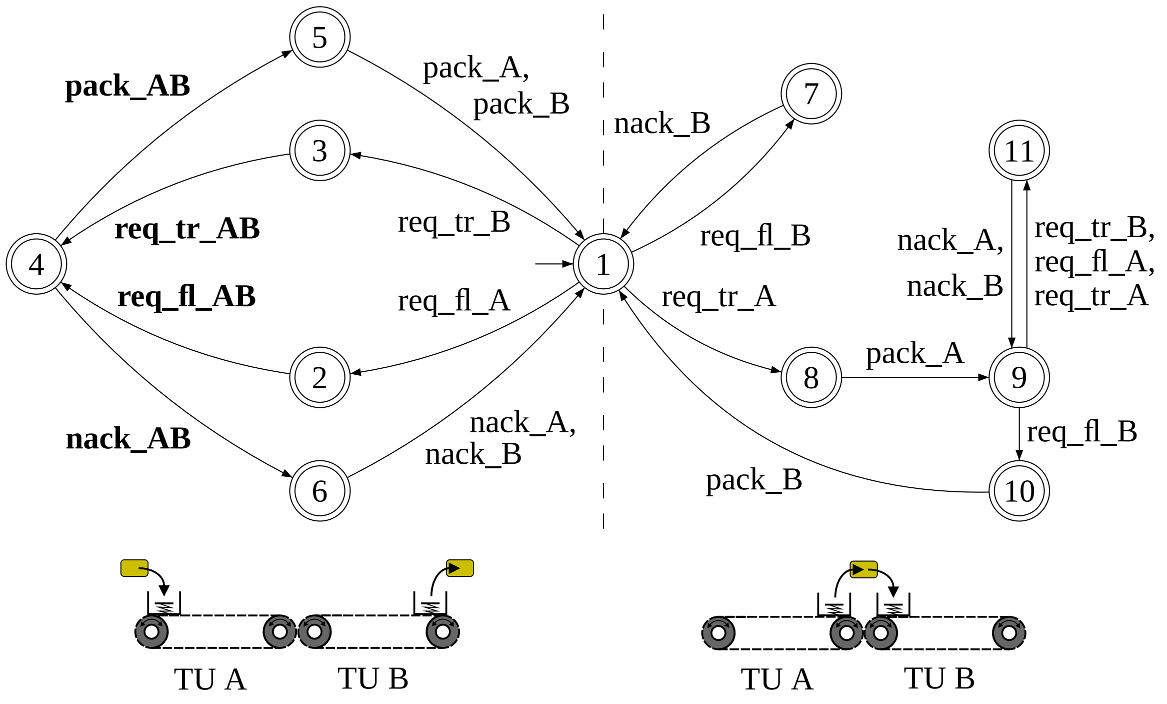


Figure 5.3: Environment model of two transport units TU A and B

First, we consider the *internal* interaction between TU A and B, namely the propagation of a workpiece from TU A to TU B, see right half of the automaton model in Figure 5.3. Initially, due to its I/O-controller structure, the environment has to accept all Y_{E}-events (all events labeled $req\_\ldots$) issued by TU A or TU B and may respond by the U_{E}-events $nack\_A$, $pack\_A$ or $nack\_B$, $pack\_B$, depending on the correct order of requests. The event req_fl_B is responded by $nack\_B$ (state 7) as TU A has not provided a workpiece yet. Instead, req_tr_A is followed by $pack\_A$, after which only the appropriate request req_fl_B leads to positive acknowledge (state 8), as TU B has to take over the workpiece provided by TU A.

The second step is the description of the *external* interaction (left part of Figure 5.3) of module AB with the remaining environment. To this end, we introduce the alphabets $Y_{\mathrm{L}} := \{req\_fl\_AB, req\_tr\_AB\}$ and $U_{\mathrm{L}} := \{nack\_AB, pack\_AB\}$ as the plant-I/O port of $\mathcal{S}_{\mathrm{EL}}$. As req_fl_A represents a request of the entire module AB, it is “translated” to the remaining environment by req_fl_AB (state 2). Now, the plant-I/O port of $\mathcal{S}_{\mathrm{EL}}$ has to accept all U_{L}-events. Both acknowledges from the remaining environment, $nack\_AB$ and $pack\_AB$ are reported to TU A by $nack\_A$ and $pack\_A$, respectively (states 4, 5 and 6). In the same way, the request req_tr_B is “translated” to the remaining environment (state 3).

Note that the environment constraints $\mathcal{S}_{\mathrm{E}i}$ as depicted in Figure 3.7 are violated in states 6, 7 and 11, because the shared resource is not provided as requested. Hence, in the compound of module AB and $\mathcal{S}_{\mathrm{EL}}$, the liveness of TU A and B is not preserved. □

Analogously to the I/O controller form, an I/O environment form can be defined for an automaton graph to represent an I/O environment.
The I/O-shuffle and the environment model are composed to one model of the interacting plants. Its external behaviour $\mathcal{S}_{\mathrm{PL}}$ (see Proposition 5.4) is an I/O plant: comparing the I/O structure of controller and environment, Proposition 3.3 carries over to the compound of plant and environment by uniform substitution as in Figure 5.4.

IO environment:		I/O controller:
$\mathcal{S}_{\mathrm{L}}$	$\leftrightarrow$	$\mathcal{S}_{\mathrm{C}}$
$\mathcal{S}_{\mathrm{EL}}$	$\leftrightarrow$	$\mathcal{S}_{\mathrm{CP}}$
I/O port $(Y_{\mathrm{E}}, U_{\mathrm{E}})$	$\leftrightarrow$	I/O port $(Y_{\mathrm{P}}, U_{\mathrm{P}})$
$\mathcal{S}_{\mathrm{PE}}$	$\leftrightarrow$	$\mathcal{S}_{\mathrm{PE}}$
I/O port $(Y_{\mathrm{P}}, U_{\mathrm{P}})$	$\leftrightarrow$	I/O port $(Y_{\mathrm{E}}, U_{\mathrm{E}})$
$\mathcal{S}_{\mathrm{P}}$	$\leftrightarrow$	$\mathcal{S}_{\mathrm{E}}$

Figure 5.4: Analogy between I/O environment and I/O controller

Proposition 5.4
Let $\mathcal{S}_{\mathrm{PE}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{E}}, Y_{\mathrm{E}}, \mathcal{L}_{\mathrm{PE}})$ be an I/O plant and let $\mathcal{S}_{\mathrm{EL}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{EL}})$ be an I/O environment. Then the external behaviour $\mathcal{S}_{\mathrm{PL}} := (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{PL}})$ with $\mathcal{L}_{\mathrm{PL}} := p_{\mathrm{PL}}(\mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{EL}})$ is an I/O plant. □

Proof
See proof of Proposition 3.3 with the analogies shown in Figure 5.4. □

In Proposition 5.3, suitable compound operator- and environment constraints describing the liveness of the individual plant components have been identified for the I/O shuffle. Now, the environment constraint $\mathcal{S}_{\mathrm{E}}$ is replaced by the I/O environment. Hence, suitable constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{L}}$ are required to enforce the original environment constraint in order to guarantee liveness of the compound plant. The following theorem characterizes such constraints. Typically, $\mathcal{S}_{\mathrm{L}}$ is given from an application context, and the below condition is solved for the variable $\mathcal{S}_{\mathrm{P}}$.

Theorem 5.1 (Compound Plant Model)
For $i \in \{1,2\}$, let $\mathcal{S}_{\mathrm{PE}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{PE}i})$ be an I/O plant, that is complete and $Y_{\mathrm{P}i}$-live w.r.t. the constraints $\mathcal{S}_{\mathrm{E}i} = (U_{\mathrm{E}i}, Y_{\mathrm{E}i}, \mathcal{L}_{\mathrm{Ei}})$ and $\mathcal{S}_{\mathrm{P}i} = (U_{\mathrm{P}i}, Y_{\mathrm{P}i}, \mathcal{L}_{\mathrm{Pi}})$. Let $\mathcal{S}_{\mathrm{EL}} = (U_{\mathrm{E}}, Y_{\mathrm{E}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{EL}})$ be an I/O environment and consider the compound system $\mathcal{S}_{\mathrm{PL}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{PL}})$, $\mathcal{L}_{\mathrm{PL}} = p_{\mathrm{PL}}((\mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}}) \parallel \mathcal{L}_{\mathrm{EL}})$. Let $\mathcal{S}_{\mathrm{P}} = (U_{\mathrm{P}}, Y_{\mathrm{P}}, \mathcal{L}_{\mathrm{P}})$ and $\mathcal{S}_{\mathrm{L}} = (U_{\mathrm{L}}, Y_{\mathrm{L}}, \mathcal{L}_{\mathrm{L}})$ be constraints with

$$p_{\mathrm{E}}(\mathcal{L}_{\mathrm{P}} \parallel (\mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}}) \parallel \mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}}) \subseteq (\mathcal{L}_{\mathrm{E1}} \parallel \mathcal{L}_{\mathrm{E2}}) \cap \mathcal{L}_{\mathrm{io}}\,, \tag{5.1}$$

$$p_{\mathrm{P}}(\mathcal{L}_{\mathrm{P}} \parallel (\mathcal{L}_{\mathrm{PE1}} \parallel_{\mathrm{io}} \mathcal{L}_{\mathrm{PE2}}) \parallel \mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}}) \subseteq (\mathcal{L}_{\mathrm{P1}} \parallel \mathcal{L}_{\mathrm{P2}}) \cap \mathcal{L}_{\mathrm{io}}\,. \tag{5.2}$$

with $\mathcal{L}_{\mathrm{io}}$ as defined in Definition 5.1. Then $\mathcal{S}_{\mathrm{PL}}$ is

(i) an I/O plant;

(ii) complete w.r.t. $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{L}}$,

(iii) Y_{P}-live w.r.t. $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{L}}$.

□

Proof

(I) $\mathcal{L}_{\mathrm{PL}}$ is an I/O-plant. Proof: this follows from Proposition 5.4.

(II) $\mathcal{L}_{\mathrm{PL}}$ is complete w.r.t. $\mathcal{L}_{\mathrm{P}}$ and $\mathcal{L}_{\mathrm{L}}$. Proof: Pick $s \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PL}} \parallel \mathcal{L}_{\mathrm{L}}$; hence there exists $r \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}}$ such that $p_{\mathrm{PL}}(r) = s$. As $\mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{EL}}$ is complete and Y_{P}-live w.r.t. $\mathcal{L}_{\mathrm{P}}$ and $\mathcal{L}_{\mathrm{L}}$, there exists $r'\nu,\ \nu \in Y_{\mathrm{P}}$ such that $rr'\nu \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}}$. As $p_{\mathrm{PL}}(r'\nu) = p_{\mathrm{PL}}(r')\nu \neq \epsilon$, $p_{\mathrm{PL}}(r')\nu = \sigma r''$ with some $\sigma \in \Sigma_{\mathrm{PL}}$ and $r \in \Sigma_{\mathrm{PL}}^{*}$.[1] Observing $p_{\mathrm{PL}}(rr'\nu) = p_{\mathrm{PL}}(r)p_{\mathrm{PL}}(r')\nu = s\sigma r''$, it holds that there exists $\sigma \in \Sigma_{\mathrm{PL}}$ such that $s\sigma \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PL}} \parallel \mathcal{L}_{\mathrm{L}}$.

(III) $\mathcal{L}_{\mathrm{PL}}$ is Y_{P}-live w.r.t. $\mathcal{L}_{\mathrm{P}}$ and $\mathcal{L}_{\mathrm{L}}$. Proof: Pick $w \in (\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}})^{\infty}$. As $\mathcal{S}_{\mathrm{PE}}$ is Y_{P}-live w.r.t. $\mathcal{L}_{\mathrm{P}}$ and $\mathcal{L}_{\mathrm{L}}$, it holds that $p_{\mathrm{YP}}(w) \in Y_{\mathrm{P}}^{\omega}$. Note that for all $w' \in (\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PL}} \parallel \mathcal{L}_{\mathrm{L}})^{\infty}$, it holds that $p_{\mathrm{YP}}(w') = p_{\mathrm{YP}}(p_{\mathrm{PL}}(w)) = p_{\mathrm{YP}}(w)$ for some $w \in (\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{EL}} \parallel \mathcal{L}_{\mathrm{L}})^{\infty}$. Hence $p_{\mathrm{YP}}(w') \in Y_{\mathrm{P}}^{\omega}$ for all $w' \in (\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PL}} \parallel \mathcal{L}_{\mathrm{L}})^{\infty}$.

□

Hence, liveness of the compound plant is achieved whenever the external constraints $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{L}}$ enforce the internal constraints $\mathcal{S}_{\mathrm{E}i}$ and $\mathcal{S}_{\mathrm{P}i}$. Then, we end up with an I/O plant as discussed in Chapter 4 and, hence, can approach the control problem accordingly. In particular, we can substitute the actual plant models $\mathcal{S}_{\mathrm{PE}i}$ by an abstraction: due to monotonicity of the applied language operations, this leads to an abstraction of the compound plant and to a conservative constraint $\mathcal{S}_{\mathrm{P}}$. As we are now in the position to design controllers also for groups of components, we can approach the design of a hierarchical control system for multi-component DES.

5.2 Stepwise Hierarchical System Design

In order to design a hierarchical control architecture for the composed system as illustrated in Figure 1.11, we suggest the following recurring sequence of steps:

[1]Note that not necessarily $\sigma = \nu$.

1. *Component-wise controller design.* For each component a local I/O controller can be designed as in Chapter 4 according to an individual specification $\mathcal{S}_{\mathrm{specCE}_i}$. By Theorem 3.1, the overall system is still given as a set of n I/O plants, where the I/O-plant model of each *controlled* component is given as the external closed loop of the uncontrolled I/O plant and its I/O controller.

2. *Abstraction step.* For the next hierarchical level, the original I/O plant components (uncontrolled plants or external closed loops) are replaced by an abstraction that captures only the behaviour that is relevant for superposed control action. For the controlled components, as mentioned earlier in this text, we propose to use the specifications $\mathcal{S}_{\mathrm{specCE}_i}$ as an abstraction for the external closed loop.

3. *Subsystem composition.* We suggest that groups of a comparatively small number of plant components shall be described by a compound model and equipped with control and measurement aggregation by one superposed I/O controller per group. At this point, the complexity of the compound model of each group (that is exponential in the number of components) is effectively reduced by the use of abstractions in the preceding step. We formally obtain a compound model of the group by a *shuffle product* composition and model the interaction of the plant components by an environment model represents the limited amount of resources available and thus, in general, does *not* meet the original environment constraints necessary for liveness of the individual plant components.

4. *Superposed control.* For each group and a specification for each group, we synthesize a superposed controller that respects Theorem 5.1 and thus meets the operator constraints and enforces the original environment constraints by only requesting resources when available. We end up with a new level of $\tilde{n} < n$ plant components, one per group. By replacement with the corresponding specifications, we proceed with step 2.

This procedure is iterated until one controller for the abstract overall plant model is designed. The exponential growth of complexity in the number of plant components observed in the monolithic approach is effectively avoided.

5.3 Complexity of the Transport Unit Example

For the subsystem composition step, the complexity of the resulting compound model is exponential in the number of subcomponents. However, the exponential growth has no effect on the next hierarchical level, as the controlled group is replaced by the specification model. As a result, assuming a fixed upper bound for the complexity of the specifications, the complexity of the overall control system is linear in the number of plant components. This deliberation is supported by the TU example.

Example 5.3
Transport Unit. Again, we consider the chain of an arbitrary number of TU's. The above sequence of hierarchical design steps is accomplished as follows.
Controller design: A local controller for each TU is designed for the specification $\mathcal{S}_{\text{specCE}}$ according to the previous sections. The local plant models and local controllers comprise 9 states each.
Abstraction step: As proposed above, the external closed loop $(\Sigma_{\text{CE}}, \mathcal{L}_{\text{CE}})$ of each TU is replaced by $\mathcal{S}_{\text{specCE}}$.
Subsystem composition: The abstractions of each two neighbored TU's are composed using the *I/O shuffle* composition. For each pair, the interaction of the two TU's among themselves and with the remaining environment is captured by a subordinate *I/O environment* model, which counts 14 states. The compound model of any two TU's is built of two specifications as in Figure 3.9 with 6 states each and the 14-state environment model. Hence, the composed result is of the order of $6 \times 6 \times 14 = 504$ states.
Design of superposed controllers: For the resulting compound models of two TU's, we require that the controlled module behaves as if it were one single transport unit. Accordingly, we keep up the specification in Fig. 3.9 also for the compounds of two TU's, by copy and paste and correct renaming of the events. The controller for two TU's and that specification counts 28 states. Now, the compound of any two controlled TU's is replaced by the specification with only 6 states - a considerable reduction compared to 504 states which we obtained above.
Overall hierarchy: Keeping up this specification for all levels, until a top-level controller for an abstract model of the whole chain of TU's is synthesized leads to a hierarchy of identical I/O controllers and I/O environments. Hence, the overall complexity can easily be predicted for a chain counting an arbitrary number of TU's.
Table 5.1 shows the sum of states for a chain of up to 16 TU's: both, the plant model hierarchy (comprising all I/O plants and the environment hierarchy) and the controller hierarchy feature linear complexity compared to the exponential growth of a monolithic plant model (see third column in Table 5.1. The according model is found in Appendix A.3).

Table 5.1: Transport Unit: Sum of States

No. of TU's	plant hierarchy	controller hierarchy	monolithic plant model
1	9	9	6
2	$2 \cdot 9 + 14 = 32$	$2 \cdot 9 + 28 = 46$	36
4	78	120	1296
8	170	278	7776
16	354	594	approx. $2,8 \cdot 10^{12}$

□

Chapter 6

Conclusions

In this contribution, we provide an input/output-based (I/O-based) system theoretic framework of hierarchical abstraction-based control system design for discrete event systems. The I/O-based description of discrete event models is adopted to formal languages from J.C. Willems' behavioural systems theory and is the key ingredient that allows for abstraction-based controller synthesis under preservation of safety- and liveness-properties.

First, a notational basis for the *concept of formal languages* is established in Chapter 2, including the graph-based representation by automata and the notion of ω-languages used to describe sequential behaviour.

With the formal language framework as a basis, an *I/O-based modeling framework for DES* is developed in Chapter 3. As a mathematical plant model, the I/O plant is proposed as an entity that interacts with an operator and an environment via well-defined I/O ports. The notion of liveness is reformulated in the context of inputs and outputs in form of a conditional liveness that depends on constraints on the external configuration of the plant. By its I/O structure, the corresponding I/O controller preserves controllability and basic liveness properties in the closed loop. As a main result, any controller that solves the control problem for an abstraction of the plant, is provably also a solution for the genuine control problem for the original plant.

In Chapter 4, an algorithmic *controller design procedure* is established that respects admissibility conditions and yields a solution to the controller design problem. Liveness of the closed loop is realized in form a an acyclic language that is algorithmically achieved via a cycle-free topology of the corresponding automata graph.

An extension of the results to a multi-layer *control hierarchy* is proposed in Chapter 5. First, a compound model for a group of plant components is developed by the I/O shuffle operation and by the notion of the I/O environment. The latter describes the interaction within the group of subplants and the interaction of the group with the remaining plant configuration. The presented

results show, that the resulting compound model readily serves as an I/O plant model for the next layer of superposed control, and constraints on liveness of the individual plant components can be passed on to be met by the superposed controller.
Next, a hierarchy of superposed controllers is developed, that is complemented by a hierarchy of environment models. At each layer of controller design, the plant models can be replaced by the specifications of the preceding design step due to the results on abstraction-based control. By repeated alternation of abstraction, subsystem composition with an environment model and superposed control, an overall control hierarchy is established that scales well in the number of plant components.

In parallel to this thesis, the I/O based approach has been implemented as the plug-in *HioSys* of the open-source C++ library libFAUDES (see [FAU, MSP08, MPS09]). The HioSys plug-in implements suitable data structures such as the class HioPlant, that extends the libFAUDES *vGenerator* class (which implements an automaton) by state- and event-attributes according to the I/O-plant form (Definition 3.5). Moreover, the plug-in offers a complete set of functions to step-by-step support the I/O-based design method as well as comprehensive routines such as *HioSynthHierarchical()*, which computes an I/O controller for a composed system. Via the libFAUDES interface to the scripting language LUA, the I/O-based design can be conducted by writing scripts that run without compiling. For a complete documentation, see [FAU].

The computational savings of the I/O based hierarchical approach compared to the monolithic approach to discrete event controller design are presented by application to the conceptional example of a chain of transport units that accompanies the thesis. Evaluated on this example, our approach features linear complexity in the number of plant components - having turned a complicated problem into a manageable one.

Appendix A

Proofs

This appendix provides some lemmas and proofs for statements made in the body of the thesis.

A.1 Languages and According Properties

Lemma A.1
Let $\mathcal{L}_\mathrm{a}$, $\mathcal{L}_\mathrm{b}$ and $\mathcal{L}_\mathrm{c}$ be languages. It holds that

$$(\mathcal{L}_\mathrm{a} \cup \mathcal{L}_\mathrm{b}) \parallel \mathcal{L}_\mathrm{c} = (\mathcal{L}_\mathrm{a} \parallel \mathcal{L}_\mathrm{c}) \cup (\mathcal{L}_\mathrm{b} \parallel \mathcal{L}_\mathrm{c})$$

□

Proof
(a) $(\mathcal{L}_\mathrm{a} \cup \mathcal{L}_\mathrm{b}) \parallel \mathcal{L}_\mathrm{c} \subseteq (\mathcal{L}_\mathrm{a} \parallel \mathcal{L}_\mathrm{c}) \cup (\mathcal{L}_\mathrm{b} \parallel \mathcal{L}_\mathrm{c})$. Proof: Pick $s \in (\mathcal{L}_\mathrm{a} \cup \mathcal{L}_\mathrm{b}) \parallel \mathcal{L}_\mathrm{c}$. Hence, $p_\mathrm{ab}(s) \in \mathcal{L}_\mathrm{a} \cup \mathcal{L}_\mathrm{b}$ and $p_\mathrm{c}(s) \in \mathcal{L}_\mathrm{c}$. W.l.o.g. assume $p_\mathrm{ab}(s) \in \mathcal{L}_\mathrm{a}$. Thus, $p_\mathrm{ab}(s) = p_\mathrm{a}(s)$. Since we have $p_\mathrm{a}(s) \in \mathcal{L}_\mathrm{a}$ and $p_\mathrm{c}(s) \in \mathcal{L}_\mathrm{c}$, it holds that $s \in \mathcal{L}_\mathrm{a} \parallel \mathcal{L}_\mathrm{c} \subseteq (\mathcal{L}_\mathrm{a} \parallel \mathcal{L}_\mathrm{c}) \cup (\mathcal{L}_\mathrm{b} \parallel \mathcal{L}_\mathrm{c})$.
(b) $(\mathcal{L}_\mathrm{a} \cup \mathcal{L}_\mathrm{b}) \parallel \mathcal{L}_\mathrm{c} \supseteq (\mathcal{L}_\mathrm{a} \parallel \mathcal{L}_\mathrm{c}) \cup (\mathcal{L}_\mathrm{b} \parallel \mathcal{L}_\mathrm{c})$. Proof: Note that it obviously holds that $\mathcal{L}_\mathrm{a} \parallel \mathcal{L}_\mathrm{c} \subseteq (\mathcal{L}_\mathrm{a} \cup \mathcal{L}_\mathrm{b}) \parallel \mathcal{L}_\mathrm{c}$ and $\mathcal{L}_\mathrm{b} \parallel \mathcal{L}_\mathrm{c} \subseteq (\mathcal{L}_\mathrm{a} \cup \mathcal{L}_\mathrm{b}) \parallel \mathcal{L}_\mathrm{c}$. Thus, $(\mathcal{L}_\mathrm{a} \parallel \mathcal{L}_\mathrm{c}) \cup (\mathcal{L}_\mathrm{b} \parallel \mathcal{L}_\mathrm{c}) \subseteq (\mathcal{L}_\mathrm{a} \cup \mathcal{L}_\mathrm{b}) \parallel \mathcal{L}_\mathrm{c}$. □

Lemma A.2
Let $\mathcal{L}_1 \subseteq \Sigma_1^*,\ \mathcal{L}_2 \subseteq \Sigma_2^*$ be languages over the alphabets $\Sigma_1,\ \Sigma_2$, and let $p_\mathrm{o} : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_\mathrm{o}^*$ be the natural projection to the alphabet $\Sigma_\mathrm{o} \subseteq \Sigma_1 \cap \Sigma_2$. Then,

$$p_\mathrm{o}(\mathcal{L}_1) \cap p_\mathrm{o}(\mathcal{L}_2) \supseteq p_\mathrm{o}(\mathcal{L}_1 \cap \mathcal{L}_2)$$

where equality does not hold, in general. □

Proof Pick an arbitrary string $s_o \in p_{\text{o}}(\mathcal{L}_1 \cap \mathcal{L}_2)$. Hence, $\exists s \in \mathcal{L}_1 \cap \mathcal{L}_2$ such that $p_{\text{o}}(s) = s_o$. Note that $s \in \mathcal{L}_1$ and $s \in \mathcal{L}_2$ and, consequently, $p_{\text{o}}(s) \in p_{\text{o}}(\mathcal{L}_1)$ and $p_{\text{o}}(s) \in p_{\text{o}}(\mathcal{L}_2)$. Hence, $p_{\text{o}}(s) = s \in p_{\text{o}}(\mathcal{L}_1) \cap p_{\text{o}}(\mathcal{L}_2)$. Thus we have $p_{\text{o}}(\mathcal{L}_1 \cap \mathcal{L}_2) \subseteq p_{\text{o}}(\mathcal{L}_1) \cap p_{\text{o}}(\mathcal{L}_2)$.
In general, equality does not hold - example: Let $\mathcal{L}_1 = \{ac\}$ over $\Sigma_1 = a, c$, $\mathcal{L}_2 = \{bc\}$ over $\Sigma_1 = b, c$ and $\Sigma_{\text{o}} = c$. Then

$$p_{\text{o}}(\mathcal{L}_1 \cap \mathcal{L}_2) = p_{\text{o}}(\varnothing) = \varnothing \subset p_{\text{o}}(\mathcal{L}_1) \cap p_{\text{o}}(\mathcal{L}_2) = \{c\}.$$

□

Lemma A.3
Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be prefix-closed languages. Then,

$$(\mathcal{L}_1)^\infty \parallel (\mathcal{L}_2)^\infty \subseteq (\mathcal{L}_1 \parallel \mathcal{L}_2)^\infty \tag{A.1}$$

□

Proof Pick $w \in (\mathcal{L}_1)^\infty \parallel (\mathcal{L}_2)^\infty$. Thus, $p_1(w) \in (\mathcal{L}_1)^\infty$ and $p_2(w) \in (\mathcal{L}_2)^\infty$. Consequently, $(p_1(w))^n \in \mathcal{L}_1$ and $(p_2(w))^n \in \mathcal{L}_2$ for all $n \in \mathbb{N}_0$, as $\mathcal{L}_1$ and $\mathcal{L}_2$ are prefix-closed. Obviously, there exists an infinite sequence $(k_i)_{i \in \mathbb{N}_0, k_{i+1} > k_i}$ such that $p_1(w^{k_i}) = (p_1(w))^{n_1} \in \mathcal{L}_1$ for each $n_1 \in \mathbb{N}_0$. For each k_i, with the length n_2 of $p_2(w^{k_i})$, we have $p_2(w^{k_i}) = (p_2(w))^{n_2} \in \mathcal{L}_2$. So, for all k_i, it holds that $p_1(w^{k_i}) \in \mathcal{L}_1$ and $p_2(w^{k_i}) \in \mathcal{L}_2$, i.e. $w^{k_i} \in \mathcal{L}_1 \parallel \mathcal{L}_2$ for all k_i of the infinite sequence (k_i). Hence, $w \in (\mathcal{L}_1 \parallel \mathcal{L}_2)^\infty$. As w was chosen arbitrarily, $(\mathcal{L}_1)^\infty \parallel (\mathcal{L}_2)^\infty \subseteq (\mathcal{L}_1 \parallel \mathcal{L}_2)^\infty$. □

Lemma A.4
For the subset relation A.1 (Lemma A.3), equality does not hold, in general. □

Proof Counterexample: Consider the prefix-closed languages $\mathcal{L}_1 = \{\epsilon, a\}$ over the alphabet $\Sigma_1 = \{a\}$ and $\mathcal{L}_2 = b^*$ over the alphabet $\Sigma_2 = \{b\}$. With $\mathcal{L}_1 \parallel \mathcal{L}_2 = \overline{b^*ab^*}$, we have $(\mathcal{L}_1 \parallel \mathcal{L}_2)^\infty = b^\omega + b^*ab^\omega$. On the other hand, with $(\mathcal{L}_1)^\infty = \varnothing$ and $(\mathcal{L}_2)^\infty = b^\omega$, we get $(\mathcal{L}_1)^\infty \parallel (\mathcal{L}_2)^\infty = \varnothing$ and thus $(\mathcal{L}_1)^\infty \parallel (\mathcal{L}_2)^\infty \subset (\mathcal{L}_1 \parallel \mathcal{L}_2)^\infty$. □

Lemma A.5
Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be regular languages over the alphabets Σ_1 and Σ_2, respectively. Then,

$$\forall w \in (\mathcal{L}_1 \parallel \mathcal{L}_2)^\infty : \ p_1(w) \in (\mathcal{L}_1)^\infty \text{ or } \ p_2(w) \in (\mathcal{L}_2)^\infty \text{ (or both).}$$

□

Proof Pick $w \in (\mathcal{L}_1 \parallel \mathcal{L}_2)^\infty$ and observe $w = \sigma_1\sigma_2\cdots$ with $\sigma_i \in \Sigma_1 \cup \Sigma_2$ *for all* $i \in \mathbb{N}$. Thus,

$$\sigma_i \in \Sigma_1 \text{ for } \textit{infinitely many } i \in \mathbb{N}\text{,or } \ \sigma_i \in \Sigma_2 \text{ for } \textit{infinitely many } i \in \mathbb{N} \text{ (or both). (*)}$$

Observe also that $\exists (n)_{n\in\mathbb{N}} : \; w^n \in \mathcal{L}_1 \parallel \mathcal{L}_2$, i.e. $p_1(w^n) \in \mathcal{L}_1$ and $p_2(w^n) \in \mathcal{L}_2$. Consequently, because of (*),

$$\exists (n_1)_{n_1\in\mathbb{N}} : \; (p_1(w))^{n_1} \in \mathcal{L}_1, \text{or } \; \exists (n_2)_{n_2\in\mathbb{N}} : \; (p_2(w))^{n_2} \in \mathcal{L}_2 \text{ (or both).}$$

As a consequence,

$$p_1(w) \in (\mathcal{L}_1)^\infty \text{ or } \; p_2(w) \in (\mathcal{L}_2)^\infty \text{ (or both).}$$

□

Lemma A.6
Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be regular languages. It holds that

$$(\mathcal{L}_1 \cup \mathcal{L}_2)^\infty = (\mathcal{L}_1)^\infty \cup (\mathcal{L}_2)^\infty$$

□

Proof

(i) $(\mathcal{L}_1 \cup \mathcal{L}_2)^\infty \supseteq (\mathcal{L}_1)^\infty \cup (\mathcal{L}_2)^\infty$. Proof: Pick $w \in (\mathcal{L}_1)^\infty \cup (\mathcal{L}_2)^\infty$ and w.l.o.g. assume $w \in (\mathcal{L}_1)^\infty$. Hence, there exists an infinite sequence $(n_i)_{i\in\mathbb{N}_0, n_{i+1}>n_i}$ such that $w|_{n_i} \in \mathcal{L}_1$ for all n_i, where $\mathcal{L}_1 \subseteq \mathcal{L}_1 \cup \mathcal{L}_2$. Thus, $w \in (\mathcal{L}_1 \cup \mathcal{L}_2)^\infty$.

(ii) $(\mathcal{L}_1 \cup \mathcal{L}_2)^\infty \subseteq (\mathcal{L}_1)^\infty \cup (\mathcal{L}_2)^\infty$. Proof: Pick $w \in (\mathcal{L}_1 \cup \mathcal{L}_2)^\infty$. Hence, there exists an infinite sequence $(n_i)_{i\in\mathbb{N}_0, n_{i+1}>n_i}$ such that $w|_{n_i} \in \mathcal{L}_1 \cup \mathcal{L}_2$ for all n_i. Thus, for all n_i, $w|_{n_i} \in \mathcal{L}_1$ or $w|_{n_i} \in \mathcal{L}_2$ (or both). As (n_i) is infinite, there exists an infinite sequence $(n_{i1})_{i1\in\mathbb{N}_0, n_{i1+1}>n_{i1}}$ such that $w|_{n_{i1}} \in \mathcal{L}_1$, or there exists an infinite sequence $(n_{i2})_{i2\in\mathbb{N}_0, n_{i2+1}>n_{i2}}$ such that $w|_{n_{i2}} \in \mathcal{L}_2$ (or both). Thus, $w \in (\mathcal{L}_1)^\infty$ or $w \in (\mathcal{L}_2)^\infty$ (or both), i.e. $w \in (\mathcal{L}_1)^\infty \cup (\mathcal{L}_2)^\infty$.

□

A.2 Input/Output-Based Results

Lemma A.7
Given two I/O plants $\mathcal{S}_{\text{PE}i} = (U_{\text{P}i}, Y_{\text{P}i}, U_{\text{E}i}, Y_{\text{E}i}, \mathcal{L}_{\text{PE}i})$ and their I/O shuffle $\mathcal{S}_{\text{PE}} = \mathcal{S}_{\text{PE1}} \parallel_{\text{io}} \mathcal{S}_{\text{PE2}}$, it holds that

$$\mathcal{L}_{\text{PE1}} \subseteq \mathcal{L}_{\text{PE}} \text{ and } \mathcal{L}_{\text{PE2}} \subseteq \mathcal{L}_{\text{PE}}$$

□

Proof For symmetry reasons, it is obviously sufficient to show the following relationship only.

$$\mathcal{L}_{\text{PE1}} \subseteq \mathcal{L}_{\parallel} \subseteq \mathcal{L}_{\text{PE}}$$

Proof: Note that $\mathcal{L}_{\parallel} := (\mathcal{L}_{\text{PE1}} \parallel \mathcal{L}_{\text{PE2}}) \cap \mathcal{L}_{\text{io}}$. $\mathcal{L}_{\text{PE1}}$ and $\mathcal{L}_{\text{PE2}}$ do not share events. Thus, it holds that $\mathcal{L}_{\text{PE1}} \subseteq \mathcal{L}_{\text{PE1}} \parallel \mathcal{L}_{\text{PE2}}$. Moreover,

$$\mathcal{L}_{\text{PE1}} \subseteq \overline{[(Y_{\text{P1}}U_{\text{P1}})^*(Y_{\text{E1}}U_{\text{E1}})^*]^*} \subseteq \mathcal{L}_{\text{io}}.$$

Hence, $\mathcal{L}_{\text{PE1}} \subseteq \mathcal{L}_{\parallel} \subseteq \mathcal{L}_{\parallel} \cup \mathcal{L}_{\text{err}} = \mathcal{L}_{\text{PE}}$. □

Proof Proof of Lemma 3.2
Preliminary note: Note that property (viii) in Definition 3.7 implies $q_0 \in Q_\mathrm{m}$. Hence, $\mathcal{L}_\mathrm{m}(G) \neq \emptyset$. We now prove that $(\Sigma, \mathcal{L}_\mathrm{m}(G))$ provides all Constraint properties.

(i) $(\Sigma, \mathcal{L}_\mathrm{m}(G))$ is a system with $\Sigma = U \dot{\cup} Y$: by definition, G recognizes the language $\mathcal{L}_\mathrm{m}(G)$ over Σ, and Property (i) requires $\Sigma = U \dot{\cup} Y$.

(ii) (U, Y) is a controller-I/O port of $(\Sigma, \mathcal{L}_\mathrm{m}(G))$. Proof: we show that (U, Y) provides all controller-I/O port properties.

(ii.i) From property (i) in Definition 3.7 we directly conclude $\Sigma = W \dot{\cup} U \dot{\cup} Y$ with $W = \emptyset$ and $U \neq \emptyset \neq Y$.

(ii.ii) $\mathcal{L}_\mathrm{m}(G) = \overline{(YU)^*}$. Proof: If $\mathcal{L}_\mathrm{m}(G) = \{\epsilon\}$, obviously $\mathcal{L}_\mathrm{m}(G) = \overline{(YU)^*}$. For $\mathcal{L}_\mathrm{m}(G) \supset \{\epsilon\}$, we continue with induction: Pick arbitrary $\sigma \in \mathcal{L}_\mathrm{m}(G) \cap \Sigma$. W.l.o.g. assume such σ exists (completeness is shown in the next item). Hence, $\delta(q_0, \sigma)!$. As property (iii) requires $q_0 \in Q_\mathrm{Y}$, property (iv) implies $\sigma \in Y$. Hence, $\sigma \in \overline{(YU)^*}$.
Now consider a nonempty string $s\sigma_{n+1} = \sigma_1\sigma_2 \ldots \sigma_n\sigma_{n+1},\ \sigma_i \in \Sigma,\ i = 1..n,\ n \in \mathbb{N}$ with $s\sigma_{n+1} \in \mathcal{L}_\mathrm{m}(G)$. Assume $s \in \overline{(YU)^*}$. We show that also $s\sigma_{n+1} \in \overline{(YU)^*}$. Note that there exists some $q \in Q$ such that $\delta(q, \sigma_n)!$ and $\delta(q, \sigma_n\sigma_{n+1})!$ and consider the following cases:

(a) $\sigma_n \in Y$. In this case, property (v) rules out $q \in Q_\mathrm{U}$. Because of property (ii), we can conclude $q \in Q_\mathrm{Y}$ and, with property (iv), $\delta(q, \sigma_n) \in Q_\mathrm{U}$. Consequently, property (v) implies $\sigma_{n+1} \in U$. Hence, $s\sigma_{n+1} \in \overline{(YU)^*}$.

(b) $\sigma_n \in U$. In this case, property (iv) rules out $q \in Q_\mathrm{Y}$. Because of property (ii), we can conclude $q \in Q_\mathrm{U}$ and, with property (v), $\delta(q, \sigma_n) \in Q_\mathrm{Y}$. Consequently, property (iv) implies $\sigma_{n+1} \in Y$. Hence, $s\sigma_{n+1} \in \overline{(YU)^*}$ whenever $s \in \overline{(YU)^*}$, which proves the induction step.

(ii.iii) $(\forall s \in \Sigma^*U \cup \{\epsilon\}, \nu \in Y)[s \in \mathcal{L}_\mathrm{m}(G) \Rightarrow s\nu \in \mathcal{L}_\mathrm{m}(G)]$. Proof:
First consider $s = \epsilon \in \mathcal{L}_\mathrm{m}(G)$ and observe $\delta(q_0, s) = q_0 \in Q_\mathrm{Y}$ by property (iii). Consequently, property (vi) implies that for all $\nu \in Y$ it holds that $\delta(q_0, s\nu)!$. Hence, if $s = \epsilon$, $s\nu \in \mathcal{L}_\mathrm{m}(G)$ for all $\nu \in Y$.
Now pick arbitrary $s\mu \in \mathcal{L}_\mathrm{m}(G),\ \mu \in U$.[1] Write $q := \delta(q_0, s)$ and observe $\delta(q, \mu)!$. As $\mu \notin Y$, property (iv) rules out $q \in Q_\mathrm{Y}$. Because of property (ii), $q \in Q_\mathrm{U}$. Thus, as $\mu \in U$, property (v) implies that $q' := \delta(q, \mu) \in Q_\mathrm{Y}$. Consequently, property (vi) implies that for all $\nu \in Y$ it holds that $\delta(q', \nu)!$. Hence, $s\mu\nu \in \mathcal{L}_\mathrm{m}(G)$ for all $\nu \in Y$.

(iii) $\mathcal{L}_\mathrm{m}(G)$ is complete. Proof: We have to show $(\forall s \in \mathcal{L}_\mathrm{m}(G) :\ \exists\, \sigma \in \Sigma)\ [s\sigma \in \overline{\mathcal{L}_\mathrm{m}(G)}]$. Note that due to property (viii), $\overline{\mathcal{L}_\mathrm{m}(G)} = \mathcal{L}_\mathrm{m}(G)$. Now pick arbitrary $s \in \mathcal{L}_\mathrm{m}(G)$. Hence there

[1]Note that due to the I/O structure proven in (ii.ii), $s \neq \epsilon$.

exists some $q \in Q$ such that $\delta(q_0, s) = q$. Because of property (vii) there exists $\sigma \in \Sigma,\ q' \in Q$ such that $q' = \delta(q, \sigma) = \delta(q_0, s\sigma)$. Property (viii) implies $q' \in Q_\mathrm{m}$. Thus, $s\sigma \in \mathcal{L}_\mathrm{m}(G)$.

Consequently, $(\Sigma, \mathcal{L}_\mathrm{m}(G))$ is an I/O constraint. □

Proof Proof of Lemma 3.4
Preliminary note: Note that property (x) in Definition 3.13 implies $q_0 \in Q_\mathrm{m}$. Hence, $\mathcal{L}_\mathrm{m}(G) \neq \varnothing$. We now prove that $(\Sigma, \mathcal{L}_\mathrm{m}(G))$ provides all I/O-controller properties.

(i) $(\Sigma, \mathcal{L}_\mathrm{m}(G))$ is a system: by definition, G recognizes the language $\mathcal{L}_\mathrm{m}(G)$ over Σ. Property (i) requires $\Sigma = U_\mathrm{C} \dot\cup Y_\mathrm{C} \dot\cup U_\mathrm{P} \dot\cup Y_\mathrm{P}$, and we identify $\Sigma_\mathrm{CP} = \Sigma_\mathrm{C} \dot\cup \Sigma_\mathrm{P} := \Sigma$ with $\Sigma_\mathrm{C} := U_\mathrm{C} \dot\cup Y_\mathrm{C}$ and $\Sigma_\mathrm{P} := U_\mathrm{P} \dot\cup Y_\mathrm{P}$.

(ii) $(U_\mathrm{C}, Y_\mathrm{C})$ and $(U_\mathrm{P}, Y_\mathrm{P})$ are a plant-I/O and a controller-I/O port of $(\Sigma, \mathcal{L}_\mathrm{m}(G))$. Proof: we first show that $(U_\mathrm{C}, Y_\mathrm{C})$ provides all plant-I/O port properties.

(ii.i) From property (i) in Definition 3.13 we directly conclude $\Sigma = W \dot\cup U_\mathrm{C} \dot\cup Y_\mathrm{C}$ (with $W = \Sigma - U_\mathrm{C} - Y_\mathrm{C} = U_\mathrm{P} \dot\cup Y_\mathrm{P}$) and $U_\mathrm{C} \neq \varnothing \neq Y_\mathrm{C}$.

(ii.ii) $\mathcal{L}_\mathrm{m}(G) = \overline{(W^*(Y_\mathrm{C}U_\mathrm{C})^*)^*}$ with $W^* = (Y_\mathrm{P}^* U_\mathrm{P}^*)^*$. Proof: We show $\mathcal{L}_\mathrm{m}(G) \subseteq \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*}$, which is a subset of $\overline{(W^*(Y_\mathrm{C}U_\mathrm{C})^*)^*}$. If $\mathcal{L}_\mathrm{m}(G) = \{\epsilon\}$, obviously $\mathcal{L}_\mathrm{m}(G) = \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*}$. For $\mathcal{L}_\mathrm{m}(G) \supset \{\epsilon\}$, we continue with induction: Pick arbitrary $\sigma \in \mathcal{L}_\mathrm{m}(G) \cap \Sigma$. Hence, $\delta(q_0, \sigma)!$. As property (iii) requires $q_0 \in Q_\mathrm{YP}$, property (iv) implies $\sigma \in Y_\mathrm{P}$. Hence, $\sigma \in \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*}$.
Now consider a nonempty string $s\sigma_{n+1} = \sigma_1\sigma_2 \ldots \sigma_n\sigma_{n+1},\ \sigma_i \in \Sigma,\ i = 1..n,\ n \in \mathbb{N}$ with $s\sigma_{n+1} \in \mathcal{L}_\mathrm{m}(G)$. Assume $s \in \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*}$. We show that $s\sigma_{n+1} \in \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*}$. Note that there exists some $q \in Q$ such that $\delta(q, \sigma_n)!$ and $\delta(q, \sigma_n\sigma_{n+1})!$ and consider the following cases:

(a) $\sigma_n \in Y_\mathrm{P}$. In this case, properties (v), (vi) and (vii) rule out $q \in Q_\mathrm{UC} \cup Q_\mathrm{YC,UP} \cup Q_\mathrm{UP}$. Because of property (ii), we can conclude $q \in Q_\mathrm{YP}$ and, with property (vi), $\delta(q, \sigma_n) \in Q_\mathrm{YC,UP}$. Consequently, property (vii) implies $\sigma_{n+1} \in Y_\mathrm{C} \cup U_\mathrm{P}$. Hence, $s\sigma_{n+1} \in \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*}$.

(b) $\sigma_n \in U_\mathrm{P}$. In this case, properties (iv) and (vii) rule out $q \in Q_\mathrm{UC} \cup Q_\mathrm{YP}$. Because of property (ii), we can conclude $q \in Q_\mathrm{YC,UP} \cup Q_\mathrm{UP}$ and, with properties (v) and (vi), $\delta(q, \sigma_n) \in Q_\mathrm{YP}$. Consequently, property (iv) implies $\sigma_{n+1} \in Y_\mathrm{P}$. Hence, $s\sigma_{n+1} \in \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*}$.

(c) $\sigma_n \in Y_\mathrm{C}$. In this case, properties (iv), (v) and (vii) rule out $q \in Q_\mathrm{UC} \cup Q_\mathrm{UP} \cup Q_\mathrm{YP}$. Because of property (ii), we can conclude $q \in Q_\mathrm{YC,UP}$ and, with property (vi), $\delta(q, \sigma_n) \in Q_\mathrm{UC}$. Consequently, property (vii) implies $\sigma_{n+1} \in U_\mathrm{C}$. Hence, $s\sigma_{n+1} \in \overline{((Y_\mathrm{P}U_\mathrm{P})^*(Y_\mathrm{P}Y_\mathrm{C}U_\mathrm{C}U_\mathrm{P})^*)^*}$.

(d) $\sigma_n \in U_{\rm C}$. In this case, properties (iv), (v) and (vi) rule out $q \in \cup Q_{\rm YC,UP} \cup Q_{\rm UP} \cup Q_{\rm YP}$. Because of property (ii), we can conclude $q \in Q_{\rm UC}$ and, with property (vii), $\delta(q,\sigma_n) \in Q_{\rm UP}$. Consequently, property (v) implies $\sigma_{n+1} \in U_{\rm P}$. Hence, $s\sigma_{n+1} \in \overline{((Y_{\rm P}U_{\rm P})^*(Y_{\rm P}Y_{\rm C}U_{\rm C}U_{\rm P})^*)^*}$.

Hence, $s\sigma_{n+1} \in \overline{(W^*(Y_{\rm C}U_{\rm C})^*)^*}$ whenever $s \in \overline{(W^*(Y_{\rm C}U_{\rm C})^*)^*}$, which proves the induction step.

(ii.iii) $(\forall s \in \Sigma^* Y_{\rm C}, \mu \in U_{\rm C})[s \in \mathcal{L}_{\rm m}(G) \Rightarrow s\mu \in \mathcal{L}_{\rm m}(G)]$. Proof:
Pick arbitrary $s\nu \in \mathcal{L}_{\rm m}(G),\ \nu \in Y_{\rm C}$. Write $q := \delta(q_0, s)$ and observe $\delta(q,\nu)!$. As $\nu \notin U_{\rm C} \cup U_{\rm P} \cup Y_{\rm P}$, properties (iv), (v) and (vii) rule out $q \in Q_{\rm UC} \cup Q_{\rm UP} \cup Q_{\rm YP}$. Because of property (ii), $q \in Q_{\rm YC,UP}$. Thus, as $\nu \in Y_{\rm C}$, property (vi) implies that $q' := \delta(q,\nu) \in Q_{\rm UC}$. Consequently, property (viii) implies that for all $\mu \in U_{\rm C}$ it holds that $\delta(q',\mu)!$. Hence, $s\nu\mu \in \mathcal{L}_{\rm m}(G)$ for all $\mu \in U_{\rm C}$.

Thus, $(U_{\rm C}, Y_{\rm C})$ is a plant-I/O port of $(\Sigma, \mathcal{L}_{\rm m}(G))$. We now show that $(U_{\rm P}, Y_{\rm P})$ provides all controller-I/O port properties.

(ii.iv) From property (i) in Definition 3.13 we directly conclude $\Sigma = W'\dot\cup U_{\rm P}\dot\cup Y_{\rm P}$ (with $W' = \Sigma - U_{\rm P} - Y_{\rm P} = U_{\rm C}\dot\cup Y_{\rm C}$) and $U_{\rm P} \neq \emptyset \neq Y_{\rm P}$.

(ii.v) $\mathcal{L}_{\rm m}(G) = \overline{(Y_{\rm P}W'^*U_{\rm P})^*}$ with $W'^* = (Y_{\rm C}^* U_{\rm C}^*)^*$. Proof: with item (ii.ii), we have shown $\mathcal{L}_{\rm m}(G) \subseteq \overline{((Y_{\rm P}U_{\rm P})^*(Y_{\rm P}Y_{\rm C}U_{\rm C}U_{\rm P})^*)^*}$, which is a subset of $\overline{(Y_{\rm P}W'^*U_{\rm P})^*}$.

(ii.vi) $(\forall s \in \Sigma^* U_{\rm P} \cup \{\epsilon\}, \nu \in Y_{\rm P})[s \in \mathcal{L}_{\rm m}(G) \Rightarrow s\nu \in \mathcal{L}_{\rm m}(G)]$. Proof:
First consider $s = \epsilon \in \mathcal{L}_{\rm m}(G)$ and observe $\delta(q_0, s) = q_0 \in Q_{\rm YP}$ by property (iii). Consequently, property (ix) implies that for all $\nu \in Y_{\rm P}$ it holds that $\delta(q_0, s\mu)!$. Hence, if $s = \epsilon$, $s\mu \in \mathcal{L}_{\rm m}(G)$ for all $\mu \in U_{\rm C}$.
Now pick arbitrary $s\mu \in \mathcal{L}_{\rm m}(G),\ \mu \in U_{\rm P}$.[2] Write $q := \delta(q_0, s)$ and observe $\delta(q,\mu)!$. As $\mu \notin U_{\rm C} \cup Y_{\rm P}$, properties (iv) and (vii) rule out $q \in Q_{\rm UC} \cup Q_{\rm YP}$. Because of property (ii), $q \in Q_{\rm YC,UP} \cup Q_{\rm UP}$. Thus, as $\nu \in U_{\rm P}$, properties (v) and (vi) imply that $q' := \delta(q,\nu) \in Q_{\rm YP}$. Consequently, property (ix) implies that for all $\mu \in U_{\rm C}$ it holds that $\delta(q',\mu)!$. Hence, $s\nu\mu \in \mathcal{L}_{\rm m}(G)$ for all $\mu \in U_{\rm C}$.

(iii) As shown in item (ii.ii), $\mathcal{L}_{\rm m}(G) \subseteq \overline{((Y_{\rm P}U_{\rm P})^*(Y_{\rm P}Y_{\rm C}U_{\rm C}U_{\rm P})^*)^*}$.

(iv) $\mathcal{L}_{\rm m}(G)$ is complete. Proof: We have to show $(\forall s \in \mathcal{L}_{\rm m}(G) :\ \exists\ \sigma \in \Sigma)\ [s\sigma \in \overline{\mathcal{L}_{\rm m}(G)}]$. Note that due to property (x), $\overline{\mathcal{L}_{\rm m}(G)} = \mathcal{L}_{\rm m}(G)$. Now pick arbitrary $s \in \mathcal{L}_{\rm m}(G)$. Hence there exists some $q \in Q$ such that $\delta(q_0, s) = q$. Because of property (xi) there exist $\sigma \in \Sigma,\ q' \in Q$ such that $q' = \delta(q,\sigma) = \delta(q_0, s\sigma)$. Property (x) implies $q' \in Q_{\rm m}$. Thus, $s\sigma \in \mathcal{L}_{\rm m}(G)$.

Consequently, $(\Sigma, \mathcal{L}_{\rm m}(G))$ is an I/O controller. □

[2]Note that due to the I/O structure proven in (ii.ii), $s \neq \epsilon$.

Proof Proof of Proposition 3.4

(i) If $\mathcal{S}_{\mathrm{PE}}$ is complete w.r.t. $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$, then $\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ is complete. Proof:
Pick an arbitrary $s \in \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$. We will show that there always exists σ such that $p_{\mathrm{PE}}(s\sigma) \in \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ and $p_{\mathrm{CP}}(s\sigma) \in \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}}$ (or $p_{\mathrm{C}}(s\sigma) \in \mathcal{L}_{\mathrm{C}}$ and $p_{\mathrm{CP}}(s\sigma) \in \mathcal{L}_{\mathrm{CP}}$), i.e. $s\sigma \in \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$. Observe that, since $\mathcal{S}_{\mathrm{CP}}$ fulfills condition (i) of Definition 3.11, $p_{\mathrm{PE}}(s) \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$.

First consider $s = \epsilon$. Since $\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}$ and $\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ is complete, we can pick $\sigma \in Y_{\mathrm{P}} \cup Y_{\mathrm{E}}$ with $p_{\mathrm{PE}}(s\sigma) = \sigma \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$. In particular, $p_{\mathrm{PE}}(s\sigma) \in \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$. Consider the following two cases: (a) If $\sigma \in Y_{\mathrm{P}}$, we have $p_{\mathrm{CP}}(s\sigma) = \sigma \in \mathcal{L}_{\mathrm{CP}}$, as $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{P}}Y_{\mathrm{C}}U_{\mathrm{C}}U_{\mathrm{P}})^*)^*}$ and Y_{P} is free in $\mathcal{S}_{\mathrm{CP}}$. Furthermore, $p_{\mathrm{C}}(s\sigma) = \epsilon \in \mathcal{L}_{\mathrm{C}}$. (b) If $\sigma \in Y_{\mathrm{E}}$, we have $p_{\mathrm{CP}}(s\sigma) = \epsilon \in \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{C}}$.

For the case $s \neq \epsilon$, write $s = r\sigma$ for some $(r \in \Sigma^*_{\mathrm{CPE}}, \sigma \in \Sigma_{\mathrm{CPE}})$. Thus, we have $p_{\mathrm{C}}(r\sigma) \in \mathcal{L}_{\mathrm{C}}$, $p_{\mathrm{CP}}(r\sigma) \in \mathcal{L}_{\mathrm{CP}}$, $p_{\mathrm{PE}}(r\sigma) \in \mathcal{L}_{\mathrm{PE}}$ and $p_{\mathrm{E}}(r\sigma) \in \mathcal{L}_{\mathrm{E}}$. Now, we need to establish the existence of $\hat{\sigma} \in \Sigma_{\mathrm{CPE}}$ such that $p_{\mathrm{PE}}(r\sigma\hat{\sigma}) \in \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$, $p_{\mathrm{CP}}(r\sigma\hat{\sigma}) \in \mathcal{L}_{\mathrm{CP}}$ and $p_{\mathrm{C}}(r\sigma\hat{\sigma}) \in \mathcal{L}_{\mathrm{C}}$. We distinguish the following cases.

(a) $\sigma \in Y_{\mathrm{C}}$: since $\mathcal{S}_{\mathrm{C}}$ is complete and since $\mathcal{L}_{\mathrm{C}} \subseteq \overline{(Y_{\mathrm{C}}U_{\mathrm{C}})^*}$, we can pick $\mu_{\mathrm{C}} \in U_{\mathrm{C}}$ with $p_{\mathrm{C}}(r\sigma\mu_{\mathrm{C}}) \in \mathcal{L}_{\mathrm{C}}$. As U_{C} is free in $\mathcal{S}_{\mathrm{CP}}$, $p_{\mathrm{CP}}(r\sigma\mu_{\mathrm{C}}) \in \mathcal{L}_{\mathrm{CP}}$. Obviously, $p_{\mathrm{PE}}(r\sigma\mu_{\mathrm{C}}) = p_{\mathrm{PE}}(r\sigma) \in \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$.

(b) $\sigma \in Y_{\mathrm{E}}$: since $\mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}$ is complete, we can pick $\mu_{\mathrm{E}} \in U_{\mathrm{E}}$ with $p_{\mathrm{PE}}(r\sigma\mu_{\mathrm{E}}) \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} \subseteq \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$. Obviously, $p_{\mathrm{CP}}(r\sigma\mu_{\mathrm{E}}) = p_{\mathrm{CP}}(r\sigma) \in \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}}$.

(c) $\sigma \in U_{\mathrm{C}}$: since $\mathcal{S}_{\mathrm{CP}}$ is complete and since $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{P}}Y_{\mathrm{C}}U_{\mathrm{C}}U_{\mathrm{P}})^*)^*}$, we can pick $\mu_{\mathrm{P}} \in U_{\mathrm{P}}$ with $p_{\mathrm{CP}}(r\sigma\mu_{\mathrm{P}}) \in \mathcal{L}_{\mathrm{CP}}$. Observe $r\sigma = t\nu_{\mathrm{P}}v$ with $t \in \Sigma^*_{\mathrm{CPE}}$, $\nu_{\mathrm{P}} \in Y_{\mathrm{P}}$ and $v \in \Sigma^*_{\mathrm{C}}$. In particular, $p_{\mathrm{PE}}(r\sigma) = p_{\mathrm{PE}}(t)\nu_{\mathrm{P}} \in \mathcal{L}_{\mathrm{PE}}$. As $\mathcal{L}_{\mathrm{PE}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}$ and U_{P} is free in $\mathcal{S}_{\mathrm{PE}}$, we have $p_{\mathrm{PE}}(r\sigma)\mu_{\mathrm{P}} = p_{\mathrm{PE}}(t)\nu_{\mathrm{P}}\mu_{\mathrm{P}} \in \mathcal{L}_{\mathrm{PE}}$. Furthermore, $p_{\mathrm{C}}(r\sigma\mu_{\mathrm{P}}) = p_{\mathrm{C}}(r\sigma) \in \mathcal{L}_{\mathrm{C}}$ and $p_{\mathrm{E}}(r\sigma\mu_{\mathrm{P}}) = p_{\mathrm{E}}(r\sigma) \in \mathcal{L}_{\mathrm{E}}$.

(d) $\sigma \in U_{\mathrm{E}} \cup U_{\mathrm{P}}$: as $\mathcal{L}_{\mathrm{PE}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*}$ and as $\mathcal{S}_{\mathrm{PE}}$ is complete w.r.t. $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$, there exists some $\hat{\sigma} \in Y_{\mathrm{P}} \cup Y_{\mathrm{E}}$ with $p_{\mathrm{PE}}(r\sigma\hat{\sigma}) \in \mathcal{L}_{\mathrm{P}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$. In particular, $p_{\mathrm{PE}}(r\sigma\hat{\sigma}) \in \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$. If $\hat{\sigma} \in Y_{\mathrm{E}}$, we have $p_{\mathrm{CP}}(r\sigma\hat{\sigma}) = p_{\mathrm{CP}}(r\sigma) \in \mathcal{L}_{\mathrm{CP}}$ and $p_{\mathrm{C}}(r\sigma\hat{\sigma}) = p_{\mathrm{C}}(r\sigma) \in \mathcal{L}_{\mathrm{C}}$.
Else $\hat{\sigma} \in Y_{\mathrm{P}}$. Observe that either (a) $r\sigma \in \Sigma^*_{\mathrm{E}}$ or b) $r\sigma = t\mu_{\mathrm{P}}v$ with $t \in \Sigma_{\mathrm{CPE}}$, $\mu_{\mathrm{P}} \in U_{\mathrm{P}}$ and $v \in \Sigma^*_{\mathrm{E}}$. In particular, (a) $p_{\mathrm{CP}}(r\sigma) = \epsilon$ or (b) $p_{\mathrm{CP}}(r\sigma) = p_{\mathrm{CP}}(t)\mu_{\mathrm{P}}$. As $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{P}}Y_{\mathrm{C}}U_{\mathrm{C}}U_{\mathrm{P}})^*)^*}$ and Y_{P} free in $\mathcal{S}_{\mathrm{CP}}$, both cases imply $p_{\mathrm{CP}}(r\sigma)\hat{\sigma} \in \mathcal{L}_{\mathrm{CP}}$. Moreover, $p_{\mathrm{C}}(r\sigma\hat{\sigma}) = p_{\mathrm{C}}(r\sigma) \in \mathcal{L}_{\mathrm{C}}$.

e) $\sigma \in Y_{\mathrm{P}}$: since $\mathcal{S}_{\mathrm{CP}}$ is complete, we can pick $\hat{\sigma} \in Y_{\mathrm{C}} \cup U_{\mathrm{P}}$ with $p_{\mathrm{CP}}(r\sigma\hat{\sigma}) \in \mathcal{L}_{\mathrm{CP}}$; if $\hat{\sigma} \in Y_{\mathrm{C}}$ we have $p_{\mathrm{C}}(r\sigma\hat{\sigma}) = p_{\mathrm{C}}(r\sigma)\hat{\sigma} \in \mathcal{L}_{\mathrm{C}}$ as Y_{C} is free in $\mathcal{S}_{\mathrm{C}}$. Furthermore, $p_{\mathrm{PE}}(r\sigma\hat{\sigma}) = p_{\mathrm{PE}}(r\sigma) \in \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$;
else $\hat{\sigma} \in U_{\mathrm{P}}$. Then, $p_{\mathrm{PE}}(r\sigma\hat{\sigma}) = p_{\mathrm{PE}}(r\sigma)\hat{\sigma} \in \mathcal{L}_{\mathrm{PE}}$, as U_{P} is a free input of $\mathcal{S}_{\mathrm{PE}}$. Furthermore, $p_{\mathrm{E}}(r\sigma\hat{\sigma}) = p_{\mathrm{E}}(r\sigma) \in \mathcal{L}_{\mathrm{E}}$ and $p_{\mathrm{C}}(r\sigma\hat{\sigma}) = p_{\mathrm{C}}(r\sigma) \in \mathcal{L}_{\mathrm{C}}$;

(ii) If in addition $\mathcal{S}_{\mathrm{CP}}$ is admissible w.r.t. $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$ then $\mathcal{L}_{\mathrm{C}} \parallel p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}) \parallel \mathcal{L}_{\mathrm{E}}$ is complete. Proof: Pick $s \in p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}})$; hence there exists $r \in \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$ such that $p_{\mathrm{CE}}(r) = s$. As $\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$ is complete and Y_{C}-live w.r.t. $\mathcal{L}_{\mathrm{C}}$ and $\mathcal{L}_{\mathrm{E}}$, there exists $r'\nu,\ \nu \in Y_{\mathrm{C}}$ such that $rr'\nu \in \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$. As $p_{\mathrm{CE}}(r'\nu) = p_{\mathrm{CE}}(r')\nu \neq \epsilon$, it holds that $p_{\mathrm{CE}}(r')\nu = \sigma r''$ with some $\sigma \in \Sigma_{\mathrm{CE}}$ and $r \in \Sigma_{\mathrm{CE}}^*$.[3] Observing $p_{\mathrm{CE}}(rr'\nu) = p_{\mathrm{CE}}(r)p_{\mathrm{CE}}(r')\nu = s\sigma r''$, it holds that there exists $\sigma \in \Sigma_{\mathrm{CE}}$ such that $s\sigma \in \mathcal{L}_{\mathrm{CE}}$.

□

Proof Proof of Theorem 4.1
IF. To show: G is free of Y_{C}-less SCC's $\Rightarrow \mathcal{L}(G)$ is Y_{C}-live. Proof by contradiction: Assume G is free of Y_{C}-less SCC's, but $\mathcal{L}(G)$ is not Y_{C}-live. Hence, according to Proposition 4.1, there exists a string $st \in \mathcal{L}(G)$, $t \neq \epsilon$, with $st \equiv_{\mathcal{L}(G)} s$ and $p_{\mathrm{YC}}(t) = \epsilon$. Note that as s can be extended by t to the equivalent string st, also st can be extended to the equivalent string $stt \in \mathcal{L}(G)$, and so on, i.e. s can be extended by an arbitrary repetition of t. Hence, $st^* \subseteq \mathcal{L}(G)$. Let n denote the finite number of states of G. Thus, only a maximum number of $m \leq n$ elements of st^* can be represented by different states. All remaining elements of st^* are represented by some of the same m states. Accordingly, we can find two strings st^{n1} and $st^{n1}t^{n2}$, $n_2 \neq 0$, that are represented by the same state: $\delta(q_0, st^{n1}) = \delta(q_0, st^{n1}t^{n2}) := q_1$. Note that each state q_2 on the path t^{n2} from q_1 to q_1 belongs to the same equivalence class $Q_i \subseteq Q$ according to Definition 4.2, as q_2 is equivalent to q_1 (acc. to the equivalence defined in Def. 4.2): from each state q_2 on this path, there exists a path τ_b to state q_1 as well as there exists a path τ_a from q_1 to q_2, with $p_{\mathrm{YC}}(\tau_a) = p_{\mathrm{YC}}(\tau_b) = \epsilon$, as $\tau_a\tau_b = t^{n2}$ and $p_{\mathrm{YC}}(t^{n2}) = \epsilon$. Consider two possible cases:
a) $|Q_i| > 1$, then Q_i is a Y_{C}-less SCC.
b) $|Q_i| = 1$. Consequently, $t^{n2} = t = \sigma$ for some $\sigma \in \Sigma - Y_{\mathrm{C}}$, as $t \neq \epsilon$ and $p_{\mathrm{YC}}(t) = \epsilon$. Thus, Q_i is a Y_{C}-less SCC.
Hence, Q_i is a Y_{C}-less SCC, and we have the contradiction.
ONLY IF. To show: $\mathcal{L}(G)$ is Y_{C}-live $\Rightarrow G$ is free of Y_{C}-less SCC's. Proof by contradiction: Assume $\mathcal{L}(G)$ is Y_{C}-live but G is not free of Y_{C}-less SCC's, i.e. there exists at least one Y_{C}-less SCC that shall be denoted Q_i. Consider two possible cases:
a) $|Q_i| = 1$. Then, Q_i consists of one element q_i. Denote s the path from the initial state to q_i, i.e. $s \in \mathcal{L}(G)$ and $\delta(q_0, s) = q_i$. According to Definition 4.2, $\delta(q_j, \sigma) = q_j$ for some $\sigma \in \Sigma - Y_{\mathrm{C}}$. Hence, $s\sigma \equiv_{\mathcal{L}(G)} s$, and also $s\sigma$ can be extended by σ, i.e. $s\sigma\sigma \in \mathcal{L}(G)$, and so on. We get $s\sigma^* \subseteq \mathcal{L}(G)$. For the limit of $\mathcal{L}(G)$, we can conclude $w := s\sigma^\omega \in \mathcal{L}(G)^\infty$. As $\sigma \in \Sigma - Y_{\mathrm{C}}$, it holds

[3]Note that not necessarily $\sigma = \nu$.

that $p_{Y\mathrm{C}}(w) = \epsilon \notin Y_{\mathrm{C}}^{\omega}$. Thus, in this case, $\mathcal{L}(G)$ is not Y_{C}-live.
b) $|Q_i| > 1$. We choose two states $q_1 \neq q_2$ that are element of Q_i. Denote s the path from the initial state to q_1, i.e. $s \in \mathcal{L}(G)$ and $\delta(q_0, s) = q_1$. As Q_i is a Y_{C}-less SCC, there exists a path t_1 from q_1 to q_2, i.e. $\delta(q_1, t_1) = \delta(q_0, st_1) = q_2$ and $st_1 \in \mathcal{L}(G)$. Likewise, there exists a path t_2 from q_2 to q_1, i.e. $\delta(q_2, t_2) = \delta(q_0, st_1t_2) = q_1$ and $st_1t_2 \in \mathcal{L}(G)$. Note that, as $\delta(q_0, s) = \delta(q_0, st_1t_2)$, it holds that $s \equiv_{\mathcal{L}(G)} st_1t_2$. As s can be extended by t_1t_2, also the nerode-equivalent string st_1t_2 can be extended by t_1t_2, i.e. $st_1t_2t_1t_2 \in \mathcal{L}(G)$, and so on. We get $s(t_1t_2)^* \subseteq \mathcal{L}(G)$. For the limit of $\mathcal{L}(G)$, we can conclude $w := s(t_1t_2)^{\omega} \in \mathcal{L}(G)^{\infty}$. As, according to Definition 4.2, $p_{Y\mathrm{C}}(t_1) = p_{Y\mathrm{C}}(t_2) = \epsilon$, it holds that $p_{Y\mathrm{C}}(w) = \epsilon \notin Y_{\mathrm{C}}^{\omega}$. Thus, also in this case, $\mathcal{L}(G)$ is not Y_{C}-live.
Hence, $\mathcal{L}(G)$ is not Y_{C}-live, and we have the contradiction. □

Proof Proof of Lemma 4.1
Observe

$$\begin{aligned}\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c} = \quad & (\mathcal{K}_{\mathrm{CP}} \cup \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \cup \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}) \parallel \mathcal{L}_{\mathrm{PE}c} \quad =^{\text{Lemma A.1}} \\ = \; & (\mathcal{K}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c}) \cup (\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{PE}c}) \cup (\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{PE}c}).\end{aligned}$$

We show $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{PE}c} = \varnothing = \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{PE}c}$. As $\mathcal{L}_{\mathrm{PE}c}$ is prefix-closed, for any string $s \in \Sigma_{\mathrm{CPE}}^*$ it holds that $s \notin \mathcal{L}_{\mathrm{PE}c} \Rightarrow st \notin \mathcal{L}_{\mathrm{PE}c}\ \forall t \in \Sigma_{\mathrm{CPE}}^*$. This means that it is sufficient to show $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \parallel \mathcal{L}_{\mathrm{PE}c} = \varnothing = \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}} \parallel \mathcal{L}_{\mathrm{PE}c}$, which can be done by showing:

$$\forall s \in \Sigma_{\mathrm{CPE}}^* : \; p_{\mathrm{CP}}(s) \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \cup \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}} \;\Rightarrow\; s \notin \mathcal{L}_{\mathrm{PE}c}$$

Pick arbitrary $s \in \mathcal{L}_{\mathrm{PE}c}$.
First, consider $p_{\mathrm{CP}}(s) \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$ and observe from Definition 4.5 of $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$:

(1) $p_{\mathrm{CP}}(s) \in \Sigma_{\mathrm{CP}}^* Y_{\mathrm{P}}$. Hence, $s = r_{\mathrm{CPE}}\nu_{\mathrm{P}}t_E$ with $r_{\mathrm{CPE}} \in \Sigma_{\mathrm{CPE}}^*,\ \nu_{\mathrm{P}} \in Y_{\mathrm{P}},\ t_E \in \Sigma_{\mathrm{E}}^*$,
(2) $p_{\mathrm{CP}}(s) = r_{\mathrm{CP}}\nu_{\mathrm{P}} \notin \mathcal{K}_{\mathrm{CP}}$ (with $r_{\mathrm{CP}} = p_{\mathrm{CP}}(r_{\mathrm{CPE}})$),
(3) $r_{\mathrm{CP}}\nu'_{\mathrm{P}} \in \mathcal{K}_{\mathrm{CP}}$ for some $\nu'_{\mathrm{P}} \in Y_{\mathrm{P}}$.

Proof by contradiction: we show that $s \in \mathcal{L}_{\mathrm{PE}c}$ is a contradiction to item (2) above.
So, assume $s \in \mathcal{L}_{\mathrm{PE}c}$. As $\mathcal{L}_{\mathrm{PE}c} = \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} \subseteq \overline{[(Y_{\mathrm{P}}(Y_{\mathrm{C}}U_{\mathrm{C}})^*U_{\mathrm{P}})^*\Sigma_{\mathrm{E}}^*]^*}$ [4], it holds that every Y_{P}-event is followed by a U_{P}- or Y_{C}- event. Comparing this to $s = r_{\mathrm{CPE}}\nu_{\mathrm{P}}t_E$, this means $t_{\mathrm{E}} = \epsilon$ and $s = r_{\mathrm{CPE}}\nu_{\mathrm{P}}$.
From item (3), we know $p_{\mathrm{CP}}(r_{\mathrm{CPE}}\nu'_{\mathrm{P}}) = r_{\mathrm{CP}}\nu'_{\mathrm{P}} \in \mathcal{K}_{\mathrm{CP}}$. As $\mathcal{K}_{\mathrm{CP}} = p_{\mathrm{CP}}(\mathcal{K}_2)$, there exists $r'_{\mathrm{CPE}} \in \Sigma_{\mathrm{CPE}}$ such that $r'_{\mathrm{CPE}}\nu'_{\mathrm{P}} \in \mathcal{K}_2$.
Note that $\mathcal{K}_2 \subseteq \mathcal{L}_{\mathrm{PE}c}$, i.e. $r'_{\mathrm{CPE}}\nu'_{\mathrm{P}} \in \mathcal{L}_{\mathrm{PE}c}$. In particular, as both, $\mathcal{K}_2$ and $\mathcal{L}_{\mathrm{PE}c}$ are prefix-closed, $r'_{\mathrm{CPE}} \in \mathcal{K}_2$ and $r'_{\mathrm{CPE}} \in \mathcal{L}_{\mathrm{PE}c}$.

[4]This language structure results from the I/O plant language format of $\mathcal{L}_{\mathrm{PE}}$ synchronized with the language format of $\mathcal{L}_{\mathrm{C}}$ induced by the controller-I/O port $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ of $\mathcal{S}_{\mathrm{C}}$.

Now, we use the normality property of $\mathcal{K}_2$: Compute $p_{\mathrm{CP}}(r'_{\mathrm{CPE}}) = r_{\mathrm{CP}}$. As also $p_{\mathrm{CP}}(r_{\mathrm{CPE}}) = r_{\mathrm{CP}}$, we have

$$r_{\mathrm{CPE}} \in p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(r'_{\mathrm{CPE}})) \subseteq p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(\mathcal{K}_2)).$$

As $r_{\mathrm{CPE}} \in \mathcal{L}_{\mathrm{PE}c}$, we have $r_{\mathrm{CPE}} \in p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(\mathcal{K}_2)) \cap \mathcal{L}_{\mathrm{PE}c}$. As $\mathcal{K}_2$ is normal w.r.t. $\mathcal{L}_{\mathrm{PE}c}$ and Σ_{CP}, we receive $r_{\mathrm{CPE}} \in \mathcal{K}_2$. Hence, as according to the assumption $r_{\mathrm{CPE}}\nu_{\mathrm{P}} \in \mathcal{L}_{\mathrm{PE}c}$, and as $\mathcal{K}_2$ is controllable w.r.t. $\mathcal{L}_{\mathrm{PE}c}$ and Y_{P}, it holds that $r_{\mathrm{CPE}}\nu_{\mathrm{P}} \in \mathcal{K}_2$.
Thus, $p_{\mathrm{CP}}(r_{\mathrm{CPE}}\nu_{\mathrm{P}}) = r_{\mathrm{CP}}\nu_{\mathrm{P}} \in \mathcal{K}_{\mathrm{CP}}$, which contradicts the above item (2)!

Second, consider $p_{\mathrm{CP}}(s) \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$ and observe from Definition 4.5 of $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$:

(1) $p_{\mathrm{CP}}(s) \in \Sigma_{\mathrm{CP}}^* U_{\mathrm{C}}$. Hence, $s = r_{\mathrm{CPE}}\mu_{\mathrm{C}}t_E$ with $r_{\mathrm{CPE}} \in \Sigma_{\mathrm{CPE}}^*,\ \mu_{\mathrm{C}} \in U_{\mathrm{C}},\ t_E \in \Sigma_{\mathrm{E}}^*$,
(2) $p_{\mathrm{CP}}(s) = r_{\mathrm{CP}}\mu_{\mathrm{C}} \notin \mathcal{K}_{\mathrm{CP}}$ (with $r_{\mathrm{CP}} = p_{\mathrm{CP}}(r_{\mathrm{CPE}})$),
(3) $r_{\mathrm{CP}}\mu'_{\mathrm{C}} \in \mathcal{K}_{\mathrm{CP}}$ for some $\mu'_{\mathrm{C}} \in U_{\mathrm{C}}$.

Proof by contradiction: we show that $s \in \mathcal{L}_{\mathrm{PE}c}$ is a contradiction to item (2) above.
So, assume $s \in \mathcal{L}_{\mathrm{PE}c}$. As $\mathcal{L}_{\mathrm{PE}c} = \mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} \subseteq \overline{[(Y_{\mathrm{P}}(Y_{\mathrm{C}}U_{\mathrm{C}})^*U_{\mathrm{P}})^*\Sigma_{\mathrm{E}}^*]^*}$, it holds that every U_{C}-event is followed by a U_{P}- or Y_{C}- event. Comparing this to $s = r_{\mathrm{CPE}}\mu_{\mathrm{C}}t_E$, this means $t_{\mathrm{E}} = \epsilon$ and $s = r_{\mathrm{CPE}}\mu_{\mathrm{C}}$.
From item (3), we know $p_{\mathrm{CP}}(r_{\mathrm{CPE}}\mu'_{\mathrm{C}}) = r_{\mathrm{CP}}\mu'_{\mathrm{C}} \in \mathcal{K}_{\mathrm{CP}}$. As $\mathcal{K}_{\mathrm{CP}} = p_{\mathrm{CP}}(\mathcal{K}_2)$, there exists $r'_{\mathrm{CPE}} \in \Sigma_{\mathrm{CPE}}$ such that $r'_{\mathrm{CPE}}\mu'_{\mathrm{C}} \in \mathcal{K}_2$.
Note that $\mathcal{K}_2 \subseteq \mathcal{L}_{\mathrm{PE}c}$, i.e. $r'_{\mathrm{CPE}}\mu'_{\mathrm{C}} \in \mathcal{L}_{\mathrm{PE}c}$. In particular, as both, $\mathcal{K}_2$ and $\mathcal{L}_{\mathrm{PE}c}$ are prefix-closed, $r'_{\mathrm{CPE}} \in \mathcal{K}_2$ and $r'_{\mathrm{CPE}} \in \mathcal{L}_{\mathrm{PE}c}$.
Now, we use the normality property of $\mathcal{K}_2$: Compute $p_{\mathrm{CP}}(r'_{\mathrm{CPE}}) = r_{\mathrm{CP}}$. As also $p_{\mathrm{CP}}(r_{\mathrm{CPE}}) = r_{\mathrm{CP}}$, we have

$$r_{\mathrm{CPE}} \in p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(r'_{\mathrm{CPE}})) \subseteq p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(\mathcal{K}_2)).$$

As $r_{\mathrm{CPE}} \in \mathcal{L}_{\mathrm{PE}c}$, we have $r_{\mathrm{CPE}} \in p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(\mathcal{K}_2)) \cap \mathcal{L}_{\mathrm{PE}c}$. As $\mathcal{K}_2$ is normal w.r.t. $\mathcal{L}_{\mathrm{PE}c}$ and Σ_{CP}, we receive $r_{\mathrm{CPE}} \in \mathcal{K}_2$. Hence, as according to the assumption $r_{\mathrm{CPE}}\mu_{\mathrm{C}} \in \mathcal{L}_{\mathrm{PE}c}$, and as $\mathcal{K}_2$ is controllable w.r.t. $\mathcal{L}_{\mathrm{PE}c}$ and U_{C}, it holds that $r_{\mathrm{CPE}}\mu_{\mathrm{C}} \in \mathcal{K}_2$.
Thus, $p_{\mathrm{CP}}(r_{\mathrm{CPE}}\mu_{\mathrm{C}}) = r_{\mathrm{CP}}\mu_{\mathrm{C}} \in \mathcal{K}_{\mathrm{CP}}$, which contradicts the above item (2)!

Consequently, the assumption $s \in \mathcal{L}_{\mathrm{PE}c}$ is wrong in both cases. I.e. for arbitrary $s \in \Sigma_{\mathrm{CPE}}^*$ with $p_{\mathrm{CP}}(s) \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \cup \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$ we get $s \notin \mathcal{L}_{\mathrm{PE}c}$. I.e. $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \parallel \mathcal{L}_{\mathrm{PE}c} = \emptyset = \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}} \parallel \mathcal{L}_{\mathrm{PE}c}$ and thus, as $\mathcal{L}_{\mathrm{PE}c}$ is prefix-closed, $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{PE}c} = \emptyset = \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{PE}c}$. □

Proof (Theorem 4.2)
Observe that for the languages constructed in the I/O Controller Synthesis Algorithm it holds that $\mathcal{K}_2 \subseteq \mathcal{K}_1 \subseteq \mathcal{K}_0$.
Note that, technically, $(\Sigma_{\mathrm{CP}}, \emptyset)$ is a solution. Now, consider $\mathcal{L}_{\mathrm{CP}} \supset \emptyset$. We have to show the following items:

1) $\mathcal{S}_{\mathrm{CP}}$ is an I/O controller:

(i) $\mathcal{S}_{\mathrm{CP}}$ is a system with $\Sigma_{\mathrm{CP}} = \Sigma_{\mathrm{C}} \dot\cup \Sigma_{\mathrm{P}}$, $\Sigma_{\mathrm{C}} := U_{\mathrm{C}} \dot\cup Y_{\mathrm{C}}$, $\Sigma_{\mathrm{P}} := U_{\mathrm{P}} \dot\cup Y_{\mathrm{P}}$;

(ii) $(U_{\mathrm{C}}, Y_{\mathrm{C}})$ and $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ are a plant- and a controller-I/O port for $\mathcal{S}_{\mathrm{CP}}$, respectively;

(iii) $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{P}}Y_{\mathrm{C}}U_{\mathrm{C}}U_{\mathrm{P}})^*)^*}$;

(iv) $\mathcal{L}_{\mathrm{CP}}$ is complete.

2) $\mathcal{S}_{\mathrm{CP}}$ is admissible to $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$:

(i) $p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{P}}$

(ii) $\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$ is Y_{C}-live w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$

3) $\mathcal{S}_{\mathrm{CP}}$ enforces $\mathcal{S}_{\mathrm{specCE}}$ on $\mathcal{S}_{\mathrm{PE}c}$.

Proof:

1)

i) $\mathcal{S}_{\mathrm{CP}}$ is a system, as Σ_{CP} is provided by Π and $\mathcal{L}_{\mathrm{CP}}$ is a regular language over Σ_{CP}.

ii) a) $(U_{\mathrm{C}}, Y_{\mathrm{C}})$ is a plant-I/O port for $\mathcal{S}_{\mathrm{CP}}$ and b) $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ is a controller-I/O port for $\mathcal{S}_{\mathrm{CP}}$.

a) We prove all plant-I/O port properties for $(U_{\mathrm{C}}, Y_{\mathrm{C}})$ given in Definition 3.2:

I) $\Sigma_{\mathrm{CP}} = W \dot\cup U_{\mathrm{C}} \dot\cup Y_{\mathrm{C}}$ with $W = \Sigma_{\mathrm{P}}$, $U_{\mathrm{C}} \neq \emptyset \neq Y_{\mathrm{C}}$ given by Π.

II) Note that $\mathcal{L}_{\mathrm{CP}} = \mathcal{K}_{\mathrm{CP}} \cup \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \cup \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$.
First, consider $\mathcal{K}_{\mathrm{CP}}$ and observe
$\mathcal{K}_{\mathrm{CP}} = p_{\mathrm{CP}}(\mathcal{K}_2) \subseteq p_{\mathrm{CP}}(\mathcal{K}_0) \subseteq \overline{(Y_{\mathrm{P}}(\epsilon + Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*} \subseteq \overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$.
Second, consider $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ and pick arbitrary $s \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$. Then, according to Definition (4.5), $s = s'\nu_{\mathrm{P}}$ for some $\nu_{\mathrm{P}} \in Y_{\mathrm{P}}$ and $\exists \nu'_{\mathrm{P}} \in Y_{\mathrm{P}}$ such that $s'\nu'_{\mathrm{P}} \in \mathcal{K}_{\mathrm{CP}}$, where $\mathcal{K}_{\mathrm{CP}} \subseteq \overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$. Consequently, as ν_{P} and ν'_{P} are from the same alphabet Y_{P}, also $s = s'\nu_{\mathrm{P}} \in \overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$. As s was chosen arbitrarily, it holds that $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \subseteq \overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$. Moreover, as $s = s'\nu_{\mathrm{P}}$, it holds that also the extension to $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ meets the language format $\overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$.
Third (analogous to $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$), consider $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ and pick arbitrary $s \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$. Then, according to Definition (4.5), $s = s'\mu_{\mathrm{C}}$ for some $\mu_{\mathrm{C}} \in U_{\mathrm{C}}$ and $\exists \mu'_{\mathrm{C}} \in U_{\mathrm{C}}$ such that $s'\mu'_{\mathrm{C}} \in \mathcal{K}_{\mathrm{CP}}$, where $\mathcal{K}_{\mathrm{CP}} \subseteq \overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$. Consequently, as μ_{C} and μ'_{C} are from the same alphabet U_{C}, also $s = s'\mu_{\mathrm{C}} \in \overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$. As s was chosen arbitrarily, it holds that $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}} \subseteq \overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$. Moreover, as $s = s'\mu_{\mathrm{C}}$, it holds that also the extension to $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ meets the language format $\overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$.
Summing up, we get $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$.

III) To show: U_{C} is free in $\mathcal{S}_{\mathrm{CP}}$, i.e.: $\forall s \in \Sigma_{\mathrm{CP}}^* Y_{\mathrm{C}},\ \mu \in U_{\mathrm{C}} :\ s \in \mathcal{L}_{\mathrm{CP}} \Rightarrow s\mu \in \mathcal{L}_{\mathrm{CP}}$.

Proof: pick an arbitrary $s \in \mathcal{L}_{\mathrm{CP}} \cap \Sigma_{\mathrm{CP}}^* Y_{\mathrm{C}}$ and arbitrary $\mu \in U_{\mathrm{C}}$. Observe $\mathcal{L}_{\mathrm{CP}} = \mathcal{K}_{\mathrm{CP}} \cup \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \cup \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$.

First, consider $s \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. As $s \in \Sigma_{\mathrm{CP}}^* Y_{\mathrm{C}}$, but $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \in \Sigma_{\mathrm{CP}}^* Y_{\mathrm{P}}$, $s \notin \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$. This means, s is element of the extension of $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$ to $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. Hence, as $s \in \Sigma_{\mathrm{CP}}^* Y_{\mathrm{C}}$, and as in $\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ all U_{C}-events are enabled after a Y_{C}-event, it holds that $s\mu \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \subseteq \mathcal{L}_{\mathrm{CP}}$.

Second, consider $s \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. As $s \in \Sigma_{\mathrm{CP}}^* Y_{\mathrm{C}}$, but $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}} \in \Sigma_{\mathrm{CP}}^* U_{\mathrm{C}}$, $s \notin \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$. This means, s is element of the extension of $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$ to $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. Hence, as $s \in \Sigma_{\mathrm{CP}}^* Y_{\mathrm{C}}$, and as in $\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ all U_{C}-events are enabled after a Y_{C}-event, it holds that $s\mu \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \subseteq \mathcal{L}_{\mathrm{CP}}$.

Third, consider $s \in \mathcal{K}_{\mathrm{CP}}$. As $\mathcal{K}_{\mathrm{CP}} = p_{\mathrm{CP}}(\mathcal{K}_2)$ and as $s \in \Sigma_{\mathrm{CP}}^* Y_{\mathrm{C}}$, $\exists s' \in \mathcal{K}_2$ such that $p_{\mathrm{CP}}(s') = s$ and $s' = s''\nu_{\mathrm{C}}$ with $s'' \in \Sigma_{\mathrm{CPE}}^*$, $\nu_{\mathrm{C}} \in Y_{\mathrm{C}}$. As $\mathcal{K}_2$ is complete, $\exists \sigma \in \Sigma_{\mathrm{CPE}}$ such that $s'\sigma \in \mathcal{K}_2$. Considering the language format of $\mathcal{K}_2$, it holds that

$$\mathcal{K}_2 \subseteq \mathcal{K}_0 \subseteq \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{specCE}} \subseteq \overline{(\Sigma_{\mathrm{P}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*} \parallel \overline{((Y_{\mathrm{C}}U_{\mathrm{C}})^*(Y_{\mathrm{E}}U_{\mathrm{E}})^*)^*} \subset \overline{(\Sigma_{\mathrm{PE}}^*(Y_{\mathrm{C}}U_{\mathrm{C}})^*)^*}$$

This means that Y_{C}- and U_{C}-events strictly alternate in $\mathcal{K}_2$. Hence, as $s' = s''\nu_{\mathrm{C}}$ we conclude $\sigma = \mu_{\mathrm{C}} \in U_{\mathrm{C}}$. I.e. $s\mu_{\mathrm{C}} \in \mathcal{K}_{\mathrm{CP}} \subseteq \mathcal{L}_{\mathrm{CP}}$ for some $\mu_{\mathrm{C}} \in U_{\mathrm{C}}$. Now, pick arbitrary $\mu'_{\mathrm{C}} \in U_{\mathrm{C}} - \mu_{\mathrm{C}}$. Then, either $s\mu'_{\mathrm{C}} \in \mathcal{K}_{\mathrm{CP}}$ or, if $s\mu'_{\mathrm{C}} \notin \mathcal{K}_{\mathrm{CP}}$, then $s\mu'_{\mathrm{C}} \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}} \subseteq \mathcal{L}_{\mathrm{CP}}$ (see Definition 4.5 of $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$).

Summing up, we have: $s \in \mathcal{L}_{\mathrm{CP}} \cap \Sigma_{\mathrm{CP}}^* Y_{\mathrm{C}} \Rightarrow s\mu \in \mathcal{L}_{\mathrm{CP}}$ for arbitrary $\mu \in U_{\mathrm{C}}$.

Hence, $(U_{\mathrm{C}}, Y_{\mathrm{C}})$ is plant-I/O port of $\mathcal{S}_{\mathrm{CP}}$.

b) $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ is a controller-I/O port for $\mathcal{S}_{\mathrm{CP}}$. Proof: we prove all controller-I/O port properties for $\mathcal{S}_{\mathrm{CP}}$ given in Definition 3.3:

I) $\Sigma_{\mathrm{CP}} = W \dot\cup U_{\mathrm{P}} \dot\cup Y_{\mathrm{P}}$ with $W = \Sigma_{\mathrm{C}}$, $U_{\mathrm{P}} \neq \emptyset \neq Y_{\mathrm{P}}$ given by Π.

II) $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$. Proof:

First, consider $\mathcal{K}_{\mathrm{CP}}$ and observe $\mathcal{K}_{\mathrm{CP}} = p_{\mathrm{CP}}(\mathcal{K}_2) \subseteq p_{\mathrm{CP}}(\mathcal{K}_0) \subseteq p_{\mathrm{CP}}(\overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}) \subseteq \overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$.

Second, consider $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ and pick arbitrary $s \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$. Then, according to Definition (4.5), $s = s'\nu_{\mathrm{P}}$ for some $\nu_{\mathrm{P}} \in Y_{\mathrm{P}}$ and $\exists \nu'_{\mathrm{P}} \in Y_{\mathrm{P}}$ such that $s'\nu'_{\mathrm{P}} \in \mathcal{K}_{\mathrm{CP}}$, where $\mathcal{K}_{\mathrm{CP}} \subseteq \overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$. Consequently, as ν_{P} and ν'_{P} are from the same alphabet Y_{P}, also $s = s'\nu_{\mathrm{P}} \in \overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$. As s was chosen arbitrarily, it holds that $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \subseteq \overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$. Moreover, as $s = s'\nu_{\mathrm{P}}$, it holds that also the extension to $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ meets the language format $\overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$.

Third (analogous to $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$), consider $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ and pick arbitrary $s \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$. Then, according to Definition (4.5), $s = s'\mu_{\mathrm{C}}$ for some $\mu_{\mathrm{C}} \in U_{\mathrm{C}}$ and $\exists \mu'_{\mathrm{C}} \in U_{\mathrm{C}}$ such that $s'\mu'_{\mathrm{C}} \in \mathcal{K}_{\mathrm{CP}}$, where $\mathcal{K}_{\mathrm{CP}} \subseteq \overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$. Consequently, as μ_{C} and μ'_{C} are from the same alphabet U_{C}, also $s = s'\mu_{\mathrm{C}} \in \overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$. As s was chosen arbitrarily, it holds that $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}} \subseteq \overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$. Moreover, as $s = s'\mu_{\mathrm{C}}$, it holds that also the extension to $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ meets the language format $\overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$.

Hence, $\mathcal{L}_{\mathrm{CP}} = \mathcal{K}_{\mathrm{CP}} \cup \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \cup \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \subseteq \overline{(Y_{\mathrm{P}}\Sigma_{\mathrm{C}}^*U_{\mathrm{P}})^*}$.

III) $(\forall s \in \Sigma_{\mathrm{CP}}^*U_{\mathrm{P}} \cup \{\epsilon\}, \nu \in Y_{\mathrm{P}})[s \in \mathcal{L}_{\mathrm{CP}} \Rightarrow s\nu \in \mathcal{L}_{\mathrm{CP}}]$. Proof:

Pick arbitrary $s \in (\Sigma_{\mathrm{CP}}^*U_{\mathrm{P}} \cup \{\epsilon\}) \cap \mathcal{L}_{\mathrm{CP}}$. We show $s\nu \in \mathcal{L}_{\mathrm{CP}}$ for arbitrary $\nu \in Y_{\mathrm{P}}$. As $s \in \mathcal{L}_{\mathrm{CP}}$, either $s \in \mathcal{K}_{\mathrm{CP}}$ or $s \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ or $s \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$.

First, consider $s \in \mathcal{K}_{\mathrm{CP}}$. Note that (only) this case includes $s = \epsilon$, as $\mathcal{K}_{\mathrm{CP}}$ is prefix-closed and as $\epsilon \notin \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \cup \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$. As $\mathcal{K}_{\mathrm{CP}} = p_{\mathrm{CP}}(\mathcal{K}_2)$, $\exists s' \in \mathcal{K}_2 \subseteq \mathcal{K}_1 \subseteq \mathcal{K}_0$ with $p_{\mathrm{CP}}(s') = s$. Note that, as $\mathcal{K}_2$ is complete, $\exists \sigma_1 \in \Sigma_{\mathrm{CPE}}$ such that $s'\sigma_1 \in \mathcal{K}_2$, and $\exists \sigma_2 \in \Sigma_{\mathrm{CPE}}$ such that $s'\sigma_1\sigma_2 \in \mathcal{K}_2$ and so on. Repeating this procedure infinitely often, we receive: $\exists w \in \Sigma_{\mathrm{CPE}}^{\omega}$ with $w = \sigma_1\sigma_2\ldots$ and $sw \in \mathcal{K}_2^{\infty}$.
As $\mathcal{K}_2 \subseteq \mathcal{K}_1$, it holds that $\mathcal{K}_2$ is Y_{C}-live (see Proposition 4.2). Thus, $p_{\mathrm{YC}}(s'w) \in Y_{\mathrm{C}}^{\omega}$, and $\exists n \in \mathbb{N}$ such that $s'w^n = s't\nu_{\mathrm{C}},\ t \in \Sigma_{\mathrm{CPE}}^*,\ \nu_{\mathrm{C}} \in Y_{\mathrm{C}}$ and $s't\nu_{\mathrm{C}} \in \mathcal{K}_2 \subseteq \mathcal{K}_0$.
As $p_{\mathrm{CP}}(\mathcal{K}_0) \subseteq \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$, and as either $s = \epsilon$ or $s = s''\mu,\ \mu \in U_{\mathrm{P}}$, it holds that $p_{\mathrm{CP}}(t) = \nu_{\mathrm{P}}t',\ t' \in \Sigma_{\mathrm{CP}}^*,\ \nu_{\mathrm{P}} \in Y_{\mathrm{P}}$. Hence, $p_{\mathrm{CP}}(s't) = s\nu_{\mathrm{P}}t' \in \mathcal{K}_{\mathrm{CP}}$, i.e. $s\nu_{\mathrm{P}} \in \mathcal{K}_{\mathrm{CP}} \subseteq \mathcal{L}_{\mathrm{CP}}$.
Now, pick arbitrary $\nu_{\mathrm{P}}' \in Y_{\mathrm{P}} - \nu_{\mathrm{P}}$. Then, either $s\nu_{\mathrm{P}}' \in \mathcal{K}_{\mathrm{CP}}$ or, if $s\nu_{\mathrm{P}}' \notin \mathcal{K}_{\mathrm{CP}}$, then $s\nu_{\mathrm{P}}' \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \subseteq \mathcal{L}_{\mathrm{CP}}$ (see Definition 4.5 of $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$).

Second, consider $s \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. Note that, as $s \in \Sigma_{\mathrm{CP}}^*U_{\mathrm{P}} \cup \{\epsilon\}$ but $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \in \Sigma_{\mathrm{CP}}^*Y_{\mathrm{P}}$, it holds that $\epsilon < s \notin \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$. I.e. $s = t\mu_{\mathrm{P}},\ t \in \Sigma_{\mathrm{CP}}^*,\ \mu_{\mathrm{P}} \in U_{\mathrm{P}}$, and s is element of the extension of $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$ to $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. As in the extension $\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ all Y_{P}-events are possible after some U_{P}-event, it holds that $s\nu_{\mathrm{P}} \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \subseteq \mathcal{L}_{\mathrm{CP}}$ for arbitrary $\nu_{\mathrm{P}} \in Y_{\mathrm{P}}$.

Third, consider $s \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. Note that, as $s \in \Sigma_{\mathrm{CP}}^*U_{\mathrm{P}} \cup \{\epsilon\}$ but $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}} \in \Sigma_{\mathrm{CP}}^*U_{\mathrm{C}}$, it holds that $\epsilon < s \notin \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$. I.e. $s = t\mu_{\mathrm{P}},\ t \in \Sigma_{\mathrm{CP}}^*,\ \mu_{\mathrm{P}} \in U_{\mathrm{P}}$, and s is element of the extension of $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$ to $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. As in the extension $\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ all Y_{P}-events are possible after some U_{P}-event, it holds that $s\nu_{\mathrm{P}} \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \subseteq \mathcal{L}_{\mathrm{CP}}$ for arbitrary $\nu_{\mathrm{P}} \in Y_{\mathrm{P}}$.

Summing up, we receive $s\nu \in \mathcal{L}_{\mathrm{CP}}$ for arbitrary $\nu \in Y_{\mathrm{P}}$.

Hence, $(U_{\mathrm{P}}, Y_{\mathrm{P}})$ is controller-I/O port of $\mathcal{S}_{\mathrm{CP}}$.
iii) $\mathcal{L}_{\mathrm{CP}} \subseteq \overline{((Y_{\mathrm{P}}U_{\mathrm{P}})^*(Y_{\mathrm{P}}Y_{\mathrm{C}}U_{\mathrm{C}}U_{\mathrm{P}})^*)^*} = \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$. Proof:
First, observe $\mathcal{K}_{\mathrm{CP}} = p_{\mathrm{CP}}(\mathcal{K}_2) \subseteq p_{\mathrm{CP}}(\mathcal{K}_0) \subseteq \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$.
Second, consider $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ and pick arbitrary $s \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$. Then, according to Definition (4.5), $s = s'\nu_{\mathrm{P}}$ for some $\nu_{\mathrm{P}} \in Y_{\mathrm{P}}$ and $\exists \nu_{\mathrm{P}}' \in Y_{\mathrm{P}}$ such that $s'\nu_{\mathrm{P}}' \in \mathcal{K}_{\mathrm{CP}}$, where $\mathcal{K}_{\mathrm{CP}} \subseteq \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$. Consequently, as ν_{P} and ν_{P}' are from the same alphabet Y_{P}, also $s = s'\nu_{\mathrm{P}} \in \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$. As s was chosen arbitrarily, it holds that $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}} \subseteq \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$. Moreover, as $s = s'\nu_{\mathrm{P}}$, it holds that also the extension to $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ meets the language format $\overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$.
Third (analogous to $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}$), consider $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ and pick arbitrary $s \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}$. Then, according to Definition (4.5), $s = s'\mu_{\mathrm{C}}$ for some $\mu_{\mathrm{C}} \in U_{\mathrm{C}}$ and $\exists \mu_{\mathrm{C}}' \in U_{\mathrm{C}}$ such that $s'\mu_{\mathrm{C}}' \in \mathcal{K}_{\mathrm{CP}}$, where $\mathcal{K}_{\mathrm{CP}} \subseteq \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$. Consequently, as μ_{C} and μ_{C}' are from the same alphabet U_{C}, also $s = s'\mu_{\mathrm{C}} \in \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$. As s was chosen arbitrarily, it holds that $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}} \subseteq \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$. Moreover, as $s = s'\mu_{\mathrm{C}}$, it holds that also the extension to

$\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ meets the language format $\overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$.

Hence, $\mathcal{L}_{\mathrm{CP}} = \mathcal{K}_{\mathrm{CP}} \cup \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \subseteq \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*}$.
iv) $\mathcal{L}_{\mathrm{CP}}$ is complete. Proof: pick arbitrary $s \in \mathcal{L}_{\mathrm{CP}}$.

First, consider $s \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. Observe that $\mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ is complete by construction and thus $\exists \sigma \in \Sigma_{\mathrm{CP}}$ such that $s\sigma \in \mathcal{K}_{\mathrm{CP}}^{Y_{\mathrm{P}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \subseteq \mathcal{L}_{\mathrm{CP}}$.

Second, consider $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$. Observe that $\mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*}$ is complete by construction and thus $\exists \sigma \in \Sigma_{\mathrm{CP}}$ such that $s\sigma \in \mathcal{K}_{\mathrm{CP}}^{U_{\mathrm{C}}}\overline{(U_{\mathrm{P}}Y_{\mathrm{P}})^*} \subseteq \mathcal{L}_{\mathrm{CP}}$.

Third, consider $s \in \mathcal{K}_{\mathrm{CP}}$. Then, $\exists s' \in \mathcal{K}_2$ such that $p_{\mathrm{CP}}(s') = s$. Note that $\mathcal{K}_2$ is complete. Thus, $\exists \sigma_1 \in \Sigma_{\mathrm{CPE}}$ such that $s'\sigma \in \mathcal{K}_2$. Analogously, $\exists \sigma_2 \in \Sigma_{\mathrm{CPE}}$ such that $s'\sigma_1\sigma_2 \in \mathcal{K}_2$. Repeating this procedure infinitely often, we receive $w \in \Sigma_{\mathrm{CPE}}^{\omega}$ with $s'w = s'\sigma_1\sigma_2\cdots \in \mathcal{K}_2^{\infty}$. As $\mathcal{K}_2 \subseteq \mathcal{K}_1$ is Y_{C}-live, $p_{\mathrm{YC}}(s') \in Y_{\mathrm{C}}^{\omega}$. Hence, $\exists n \in \mathbb{N}$ such that $w^n = t\nu_{\mathrm{C}} \in \mathcal{K}_2$. Note that $p_{\mathrm{CP}}(t\nu_{\mathrm{C}}) = p_{\mathrm{CP}}(t)\nu_{\mathrm{C}} \neq \epsilon$. Consequently, for $p_{\mathrm{CP}}(s') = s$, $\exists \sigma \in \Sigma_{\mathrm{CP}}$ with $\sigma \leq p_{\mathrm{CP}}(t\nu_{\mathrm{C}})$ and $s\sigma \in \mathcal{K}_{\mathrm{CP}} \subseteq \mathcal{L}_{\mathrm{CP}}$.

Summing up, for arbitrary $s \in \mathcal{L}_{\mathrm{CP}}$, $\exists \sigma \in \Sigma_{\mathrm{CP}}$ such that $s\sigma \in \mathcal{L}_{\mathrm{CP}}$. Hence, $\mathcal{L}_{\mathrm{CP}}$ is complete.

Consequently, $\mathcal{S}_{\mathrm{CP}}$ is an I/O controller.

2) $\mathcal{S}_{\mathrm{CP}}$ is admissible to $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$, $\mathcal{S}_{\mathrm{P}}$ and $\mathcal{S}_{\mathrm{E}}$. Proof: We have to show (i) $p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{P}}$ and (ii) $\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}}$ is Y_{C}-live w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$.
(i) $p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{P}}$. Proof:
Observe

$$\begin{aligned} p_{\mathrm{P}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) = p_{\mathrm{P}}(\mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c}) =_{\text{Lemma 4.1}} p_{\mathrm{P}}(\mathcal{K}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c}) \subseteq \\ p_{\mathrm{P}}(\mathcal{K}_{\mathrm{CP}}) = p_{\mathrm{P}}(p_{\mathrm{CP}}(\mathcal{K}_2)) = p_{\mathrm{P}}(\mathcal{K}_2) \subseteq p_{\mathrm{P}}(\mathcal{K}_0) = \\ p_{\mathrm{P}}(\mathcal{L}_{\mathrm{PE}c} \parallel \mathcal{L}_{\mathrm{P}} \parallel \overline{(Y_{\mathrm{P}}(\epsilon \vee Y_{\mathrm{C}}U_{\mathrm{C}})U_{\mathrm{P}})^*} \parallel \mathcal{L}_{\mathrm{specCE}}) \subseteq \mathcal{L}_{\mathrm{P}} \end{aligned}$$

(ii) to show: $\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}$ is Y_{C}-live. Proof:
Observe
$\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} = \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c} =_{\text{Lemma 4.1}} \mathcal{K}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c} = p_{\mathrm{CP}}(\mathcal{K}_2) \parallel \mathcal{L}_{\mathrm{PE}c} = p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(\mathcal{K}_2)) \cap \mathcal{L}_{\mathrm{PE}c}$. Note that, by definition, $\mathcal{K}_2$ is normal w.r.t. Σ_{CP} and $\mathcal{L}_{\mathrm{PE}c}$. Thus $p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(\mathcal{K}_2)) \cap \mathcal{L}_{\mathrm{PE}c} = \mathcal{K}_2$. Summing up, we get $\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}} = \mathcal{K}_2$, and $\mathcal{K}_2 \subseteq \mathcal{K}_1$ is Y_{C}-live.

3) $\mathcal{S}_{\mathrm{CP}}$ enforces $\mathcal{S}_{\mathrm{specCE}}$ on $\mathcal{S}_{\mathrm{PE}}$ w.r.t. $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{E}}$. Proof:

Observe

$$p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) =_{\text{Lemma 4.1}} p_{\mathrm{CE}}(\mathcal{K}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}c}) =$$
$$p_{\mathrm{CE}}(p_{\mathrm{CP}}(\mathcal{K}_2) \parallel \mathcal{L}_{\mathrm{PE}c}) = p_{\mathrm{CE}}(p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(\mathcal{K}_2)) \cap \mathcal{L}_{\mathrm{PE}c})$$

Note that, by definition, $\mathcal{K}_2$ is normal w.r.t. Σ_{CP} and $\mathcal{L}_{\mathrm{PE}c}$. Thus $p_{\mathrm{CE}}(p_{\mathrm{CP}}^{-1}(p_{\mathrm{CP}}(\mathcal{K}_2)) \cap \mathcal{L}_{\mathrm{PE}c}) = p_{\mathrm{CE}}(\mathcal{K}_2)$. Note that $p_{\mathrm{CE}}(\mathcal{K}_2) \subseteq p_{\mathrm{CE}}(\mathcal{K}_0) \subseteq \mathcal{L}_{\mathrm{specCE}}$. Summing up, $p_{\mathrm{CE}}(\mathcal{L}_{\mathrm{C}} \parallel \mathcal{L}_{\mathrm{CP}} \parallel \mathcal{L}_{\mathrm{PE}} \parallel \mathcal{L}_{\mathrm{E}}) \subseteq \mathcal{L}_{\mathrm{specCE}}$.

From items 1) to 3), we conclude: $\mathcal{S}_{\mathrm{CP}}$ is a solution for Π. □

A.3 Chain of Transport Units: Monolithic Plant Model

A standard shared event model of a TU B that lies between a TU A on its left and a TU C on its right is shown in the subsequent figure. The events are interpreted as follows. The shared events (denoted by bold labels) are $A2B$ and $B2C$ and describe the propagation of a workpiece from TU A to TU B and from TU B to TU C, respectively. $Bfull$ / $Bempty$ is issued by the sensor when a workpiece arrives in / leaves the box. A workpiece that was received from TU A is transported to the right border of the TU by $Bdel2r$. $Btakefl$ moves the box to the left border.

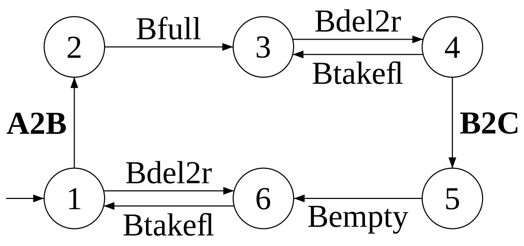


Figure 1.1: Transport Unit: simple shared-event model

The composition of TU B with the analogous model of TU A results in a state minimal automaton with 36 states, i.e. the worst case of exponential growth is found for this example. For a chain of up to 16 TU's, we obtain the numbers in the third column of table 5.1.

References

[AHU75] A.V. AHO, J.E. HOPCROFT, AND J.D. ULLMAN. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1975.

[Bal94] S. BALEMI. Input/Output Discrete Event Processes and Communication Delays. *Discrete Event Dynamic Systems: Theory and Applications*, 4(1):41–85, February 1994.

[BGK$^+$90] R. D. BRANDT, V. GARG, R. KUMAR, F. LIN, S. I. MARCUS, AND W. M. WONHAM. Formulas for Calculating Supremal Controllable and Normal Sublanguages. *System and Control Letters*, 15(2):111–117, 1990.

[BHP$^+$93] S. BALEMI, G.J. HOFFMANN, P.GYUGYI, H.WONG-TOI, AND G.F. FRANKLIN. Supervisory Control of a Rapid Thermal Multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059, July 1993.

[BW94] B.A. BRANDIN AND W.M. WONHAM. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39:329–342, 1994.

[CDFV88] R. CIESLAK, C. DESCLAUX, A. FAWAZ, AND P. VARAIYA. Supervisory Control of Discrete Event Processes with Partial Observation. *IEEE Transactions on Automatic Control*, 33(3):249–260, 1988.

[CKN98] J.E.R. CURY, B.A. KROGH, AND T. NIINOMI. Synthesis of supervisory controllers for hybrid systems based on approximating automata. *IEEE Transactions on Automatic Control, Special issue on hybrid systems*, 43:564–568, 1998.

[CL08] C.G. CASSANDRAS AND S. LAFORTUNE. *Introduction to Discrete Event Systems*. Springer, New York, USA, 2nd edition, 2008.

[dCCK02] A.E.C. DA CUNHA, J.E.R. CURY, AND B.H. KROGH. An Assume Guarantee Reasoning for Hierarchical Coordination of Discrete Event Systems. *WODES*, 2002.

[dQC00] M.H. DE QUERIOZ AND J.E.R. CURY. Modular Control of Composed Systems. *American Control Conference*, 2000.

[FAU] Friedrich-Alexander University Discrete Event Systems library (libFAUDES), as of July 2009. http://www.rt.eei.uni-erlangen.de/FGdes/faudes.

[FM06] H. FLORDAL AND R. MALIK. Modular Nonblocking Verification Using Conflict Equivalence. In *IEEE Proc. WODES'06 - 8th International Workshop on Discrete Event Systems, Ann Arbor, USA*, pages 100–106, 2006.

[GM04] B. GAUDIN AND H. MARCHAND. Modular Supervisory Control of a Class of Concurrent Discrete Event Systems. *Workshop on Discrete Event Systems*, 2004.

[GM05] B. GAUDIN AND H. MARCHAND. Efficient Computation of Supervisors for loosely synchronous Discrete Event Systems: A State-Based Approach. *IFAC World Congress*, 2005.

[HC02] P. HUBBARD AND P.E. CAINES. Dynamical Consistency in Hierarchical Supervisory Control. *IEEE Transactions on Automatic Control*, 47(1):37–52, 2002.

[HU79] J.E. HOPCROFT AND J.D. ULLMAN. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, 1979.

[JMRT08] T. JERON, H. MARCHAND, V. RUSU, AND V. TSCHAEN. Ensuring the Conformance of Reactive Discrete-Event Systems Using Supervisory Control. In *IEEE Proceedings, 42nd Conference on Decision and Control*, volume 3, pages 2692–2697, December 2008.

[KASL00] X. KOUTSOUKOS, P.J. ANTSAKLIS, J.A. STIVER, AND M.D. LEMMON. Supervisory Control of Hybrid Systems. *Proceedings of the IEEE*, 88:1026–1049, July 2000.

[KGM92] R. KUMAR, V. GARG, AND S.I. MARCUS. On supervisory control of sequential behaviors. *IEEE Transactions on Automatic Control*, 37:1978–1985, 1992.

[KGM95] R. KUMAR, V.K. GARG, AND S.I. MARCUS. Finite buffer realization of input-output discrete-event systems. *IEEE Transactions on Automatic Control*, 40(6):1042–1053, 1995.

[KvS04] J. KOMENDA AND J. H. VAN SCHUPPEN. Supremal Normal Sublanguages of Large Distributed Discrete-Event Systems. *Workshop on Discrete Event Systems*, 2004.

[Led96] R.J. LEDUC. PLC Implementation of a DES Supervisor for a Manufacturing System Testbed: an Implementation Perspective. *Master's Thesis, Department of Computer and Electrical Engineering, University of Toronto*, 1996.

[Led02] R.J. LEDUC. *Hierarchical Interface Based Supervisory Control*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, 2002.

[LT89] N. Lynch and M. Tuttle. An introduction to Input/Output automata. *CWI-Quarterly*, 2(3):219–246, 1989.

[LW90] F. Lin and W.M. Wonham. Decentralized Control and Coordination of Discrete-Event Systems with Partial Observation. *IEEE Transactions on Automatic Control*, 35(12):1330–1337, 1990.

[LW91] F. Lin and W.M. Wonham. Verification of Nonblocking in Decentralized Supervision. *Control-Theory and Advanced Technology*, 7(1):19–29, 1991.

[LW97] S.-H. Lee and K.C. Wong. Decentralised control of concurrent discrete-event systems with non-prefix closed local specification. In *IEEE Proc. of the 36th International Conference on Decision and Control*, pages 2958–2963, 1997.

[LW02] S-H. Lee and K.C. Wong. Structural Decentralised Control of Concurrent DES. *European Journal of Control*, 35:1125–1134, October 2002.

[Ma04] C. Ma. Nonblocking Supervisory Control of State Tree Structures. *Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Toronto*, 2004.

[MF08] R. Malik and H. Flordal. Yet Another Approach to Compositional Synthesis of Discrete Event Systems. In *IEEE Proc. WODES'08 - 9th International Workshop on Discrete Event Systems, Göteborg, Sweden*, pages 16–21, 2008.

[MPS09] Th. Moor, S. Perk, and K. Schmidt. Extending the Discrete Event Systems Library libFAUDES. In *Methoden und Anwendungen der Regelungstechnik – Erlangen-Münchener Workshops 2007 und 2008*, pages 121–134. Shaker-Verlag, 2009.

[MR99] Th. Moor and J. Raisch. Supervisory control of hybrid systems within a behavioural framework. *Systems and Control Letters*, 38:157–166, 1999.

[MR05] Th. Moor and J. Raisch. Hierarchical Hybrid Control of a Multiproduct Batch Blant. In *Proc. 16th IFAC World Congress*, Prague, 2005.

[MRD03] Th. Moor, J. Raisch, and J.M. Davoren. Admissibility criteria for a hierarchical design of hybrid control systems. In *Proc. IFAC Conference on the Analysis and Design of Hybrid Systems (ADHS'03)*, pages 389–394, 2003.

[MSP08] Th. Moor, K. Schmidt, and S. Perk. libFAUDES – An Open Source C++ Library for Discrete Event Systems. In *IEEE Proc. WODES'08 - 9th International Workshop on Discrete Event Systems, Göteborg, Sweden*, pages 125–130, 2008.

[Ner58] A. Nerode. Linear Automaton Transformations. *Proceedings of the American Mathematical Society*, 9:541–544, 1958.

[OW90] J. S. OSTROFF AND W. M. WONHAM. A Framework for Real-Time Discrete Event Control. *IEEE Transactions on Automatic Control*, 35, April 1990.

[Per04] S. PERK. Hierarchical Design of Discrete Event Controllers: An Automated Manufacturing System Case Study. 2004. Diploma Thesis, Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg.

[PMS06] S. PERK, TH. MOOR, AND K. SCHMIDT. Hierarchical Discrete Event Systems with Inputs and Outputs. In *IEEE Proc. WODES'06 - 8th International Workshop on Discrete Event Systems, Ann Arbor, USA*, pages 427–432, 2006.

[PMS07a] S. PERK, TH. MOOR, AND K. SCHMIDT. Model Based Design of Hierarchical Control Systems with Input/Output Structure. In *Methoden und Anwendungen der Regelungstechnik – Erlangen-Münchener Workshops 2005 und 2006*, pages 63–75. Shaker-Verlag, 2007.

[PMS07b] S. PERK, TH. MOOR, AND K. SCHMIDT. Regelungstheorie für ereignisdiskrete Systeme zur modellbasierten Berechnung von SPS-Programmen. In *3. Rexroth Doktoranden Kolloquium*, Lohr am Main, Germany, 2007.

[PMS08] S. PERK, TH. MOOR, AND K. SCHMIDT. Controller Synthesis for an I/O-Based Hierarchical System Architecture. In *IEEE Proc. WODES'08 - 9th International Workshop on Discrete Event Systems, Göteborg, Sweden*, pages 474–479, 2008.

[QC00] M.H.DE QUERIOZ AND J.E.R. CURY. Modular Supervisory Control of Large Scale Discrete Event Systems. *Workshop on Discrete Event Systems*, 2000.

[RO98] J. RAISCH AND S.D. O'YOUNG. Discrete approximation and supervisory control of continuous systems. *IEEE Transactions on Automatic Control, Special issue on hybrid systems*, 43:569–573, 1998.

[RW87a] P.J. RAMADGE AND W.M. WONHAM. Modular Feedback Logic for Discrete Event Systems. *SIAM Journal of Control and Optimization*, 25:1202–1218, 1987.

[RW87b] P.J. RAMADGE AND W.M. WONHAM. Supervisory control of a class of discrete event systems. *SIAM J. Control and Optimization*, 25:206–230, 1987.

[RW89] P.J. RAMADGE AND W.M. WONHAM. The control of discrete event systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[SB08] K. SCHMIDT AND C. BREINDL. On Maximal Permissiveness of Hierarchical and Modular Supervisory Control Approaches for Discrete Event Systems. In *IEEE Proc. WODES'08 - 9th International Workshop on Discrete Event Systems, Göteborg, Sweden*, pages 462–467, 2008.

[Sch05] K. SCHMIDT. *Hierarchical Control of Decentralized Discrete Event Systems: Theory and Application*. PhD thesis, Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg, 2005. Download: http://www.rt.eei.uni-erlangen.de/FGdes/publications.html.

[SMG06] K. SCHMIDT, H. MARCHAND, AND B. GAUDIN. Modular and decentralized supervisory control of concurrent discrete event systems using reduced system models. In *IEEE Proc. WODES'08 - 9th International Workshop on Discrete Event Systems, Göteborg, Sweden*, pages 149–154, 2006.

[SMP08] K. SCHMIDT, TH. MOOR, AND S. PERK. Nonblocking Hierarchical Control of Decentralized Discrete Event Systems. *IEEE Transactions on Automatic Control*, 53(10):2252–2265, Nov. 2008.

[SSZ07] K. SCHMIDT, E. SCHMIDT, AND J. ZADDACH. A Shared-Medium Communication Architecture For Distributed Discrete Event Systems. *Mediterranean Conference on Control and Automation*, 2007.

[TW94a] J.G. THISTLE AND W.M. WONHAM. Supervision of infinite behavior of discrete-event systems. *SIAM J. Control and Optimization*, 32:1098–1113, 1994.

[TW94b] J.G. THISTLE AND W.M. WONHAM. Supervision of infinite behavior of finite automata. *SIAM J. Control and Optimization*, 32:1075–1097, 1994.

[Wen06] F. WENCK. *Modellbildung, Analyse und Steuerungsentwurf für gekoppelte ereignisdiskrete Systeme*. PhD thesis, Lehrstuhl für Automatisierungstechnik und Prozessinformatik, Ruhr-Universität Bochum, 2006.

[WH91] Y. WILLNER AND M. HEYMANN. Supervisory Control of Concurrent Discrete-Event Systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[Wil91] J.C. WILLEMS. Paradigms and puzzles in the theory of dynamic systems. *IEEE Transactions on Automatic Control*, 36:258–294, 1991.

[Won08] W.M. WONHAM. Supervisory control of discrete event systems. Technical report, Department of Electrical & Computer Engineering, University of Toronto, 2008. Download: http://www.control.toronto.edu/DES/.

[WR88] W.M. WONHAM AND P.J. RAMADGE. Modular Supervisory Control of Discrete Event Systems. *Mathematics of Control, Signals and Systems*, 1(1):13–30, 1988.

[WW96] K.C. WONG AND W.M. WONHAM. Hierarchical Control of Discrete-Event Systems. *Discrete Event Dynamic Systems: Theory and Applications*, 6(3):241–273, 1996.

[ZW90] H. ZHONG AND W.M. WONHAM. On the Consistency of Hierarchical Supervision in Discrete-Event Systems. *IEEE Transactions on Automatic Control*, 35:1125–1134, October 1990.

Lebenslauf

Zur Person:

Sebastian Perk
geboren am 03. 09. 1978 in Gräfelfing
verheiratet, 1 Kind

Schulbildung:

1985–1989	Grundschule in Greifenberg
1989–1998	Rhabanus-Maurus-Gymnasium St. Ottilien
Juni 1998	Allgemeine Hochschulreife

Wehrdienst:

1998–1999	Grundwehrdienst bei der Drohnenbatterie 200 in München

Studium:

1999	Praktikum bei Infineon Technologies in Regensburg
1999–2004	Studium der Elektrotechnik, Elektronik und Informationstechnik an der Friedrich-Alexander-Universität Erlangen-Nürnberg
2000, 2002	Studentische Hilfskraft am Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik der Universität Erlangen-Nürnberg,
2002	Werkstudent bei Framatome ANP (nunmehr AREVA) in Erlangen
2003	3-monatiger USA-Aufenthalt im Rahmen der Diplomarbeit am Department of Electrical and Computer Engineering der Carnegie Mellon University Pittsburgh
2003-2004	Studentische Hilfskraft am Lehrstuhl für Regelungstechnik der Universität Erlangen-Nürnberg
Dez. 2004	Studienabschluss Dipl.-Ing.

Hochschultätigkeit:

seit 2005	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Regelungstechnik der Universität Erlangen-Nürnberg