Hierarchical Control of Decentralized Discrete Event Systems

Theory and Application

Der Technischen Fakultät der

Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR-INGENIEUR

vorgelegt von

Klaus Schmidt

Erlangen 2005

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

Tag der Einreichung:	23. 05. 2005
Tag der Promotion:	11. 07. 2005
Dekan:	Prof. Dr. rer. nat. A. Winnacker
Berichterstatter:	Prof. Dr.-Ing. T. Moor
	Prof. Bruce Krogh

Acknowledgement

At first I would like to thank my thesis advisor Prof. Dr.-Ing. Thomas Moor for his interest in my work and for his helpful and constructive comments and ideas. Further on I want to highlight the hospitality of Prof. Bruce Krogh who gave me the opportunity to participate in his research activites on various topics at the Carnegie Mellon University in Pittsburgh.

I am grateful to Prof. Dr.-Ing. Günter Roppenecker, the chairman of my Ph.D. committee, for supporting my work at the Lehrstuhl für Regelungstechnik and for encouraging my studies and research abroad. Also I want to thank Prof. Dr.-Ing. Christoph Plaum for his participation in the Ph.D. committee as the external faculty member.

It was very important for my scientific career to work with my colleague Dr. Ing. Johann Reger who introduced me to the area of discrete event systems. Along the same lines I would like to thank my former student and colleague Dipl.-Ing. Sebastian Perk for his collaboration, and my colleagues Dipl.-Ing. Felix Antritter, Dr.-Ing. Joachim Deutscher and Dipl.-Ing. Gerald Drenckhahn for their friendship.

In particular I would like to express my gratitude to my family for their constant love, availability and support, and I would like to thank my parents-in-law for their encouragment and trust. Finally my love and gratefulness goes to my wife for sharing my academic interests as well as my life and for her love and care during the final time of my thesis work.

Erlangen, July 2005 Klaus Schmidt

Table of Contents

Zusammenfassung

Gegenstand dieser Arbeit ist der hierarchische und dezentrale Steuerungsentwurf für ereignisdiskrete Systeme (DES). Ereignisdiskrete Systeme besitzen einen diskreten Zustandsraum und auch die Zeitwerte sind als diskrete Zeitpunkte aufzufassen. Das dynamische Verhalten von DES ist dabei ereignisgetrieben, das heißt diskrete Zustandsänderungen werden durch das Auftreten asynchroner Ereignisse verursacht.

Mitte der 80er Jahre wurde von P.J. Ramadge und W.M. Wonham eine Steuerungstheorie (RW-Steuerungstheorie) für ereignisdiskrete Systeme entwickelt. Ereignisdiskrete Systeme werden durch formale Sprachen über dem Alphabet der Systemereignisse charakterisiert, wobei die Sprache selbst die Systemtrajektorien beschreibt. Durch Verhindern sogenannter “steuerbarer“ Ereignisse kann das Systemverhalten eingeschränkt werden, um einem spezifizierten Verhalten zu genügen.

Da ereignisdiskrete Systeme sehr viele Zustände haben können, ist die RW-Steuerungstheorie für große zusammengesetzte Systeme nicht direkt anwendbar. Aufgrund dessen werden in der Literatur verschiedene Ansätze — modulare, dezentrale und hierarchische Ansätze — untersucht, um durch die Ausnutzung der Systemstruktur die theoretischen Ergebnisse der RW-Steuerungstheorie umzusetzen.

Unser Ansatz verbindet dabei dezentrale und hierarchische Verfahren, um die RW-Steuerungstheorie für große ereignisdiskrete Systeme zu erweitern. Es wird ausgenutzt, dass große DES aus vielen Systemkomponenten zusammengesetzt sind, welche interagieren. Die einzelnen Komponenten werden lokal gesteuert und dann auf das für die Interaktion relevante Verhalten abstrahiert. Die abstrahierten dezentralen Systemmodelle werden in einem höheren Level der Hierarchie zusammengesetzt und dort gesteuert. Diese mit geringerem Rechenaufwand ermittelte Steuerung wird dann in dezentrale Steuerungen im unteren Level der Hierarchie übersetzt. Unser Verfahren stellt dabei strukturelle Bedingungen an die Komponentenmodelle sowie ihre Abstraktionen. Sind diese Bedingungen erfüllt, so garantiert unsere Methode hierarchisch konsistentes sowie blockierungsfreies Verhalten des gesteuerten Systems, d.h. im oberen Level entworfene Steuerungen können im unteren Level implementiert werden und das gesteuerte System bleibt nicht in einem unerwünschten Systemzustand hängen. Außerdem läßt sich unser Ansatz für eine beliebige Anzahl von hierarchischen Ebenen anwenden.

Die Funktionsfähigkeit unseres Verfahrens wird anhand eines umfassend ausgearbeiteten Laborbeispiels mit einer Zustandsanzahl in der Größenordnung von 10^{24} illustriert. Dabei wird ein Fischertechnik Laborexperiment[1] einer Fertigungsanlage modelliert und eine Spezifikation für das Gesamtmodell in einer 4-Level Hierarchie implementiert.

[1]Die Fischertechnikanlage befindet sich am Lehrstuhl für Regelungstechnik der Universität Erlangen-Nürnberg.

Chapter 1

Introduction

Various technical systems and processes, such as manufacturing systems, telecommunication networks, traffic systems, logistics, to name just a few, can be described as discrete event systems (DES). These systems exhibit the common characteristic feature that they are discrete in both state space and in time. The dynamic behavior of a DES is event driven, that is changes of the discrete system state are triggered by the occurrence of asynchronous events.

In the mid 80s, a framework for the control of DES was established by P.J. Ramadge and W.M. Wonham. In [RW87b], the concept of a feedback controller is employed to achieve that the system behaves as specified by the system designer. This feedback controller is denoted supervisor, as it observes the events occurring in the system and disables events according to its control strategy, whereby the strategy depends on past event sequences. The feedback loop is shown in Figure 1.1.

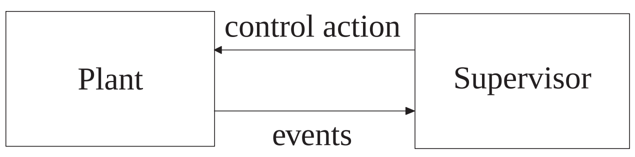


Figure 1.1: Feedback loop with a DES plant and a supervisor

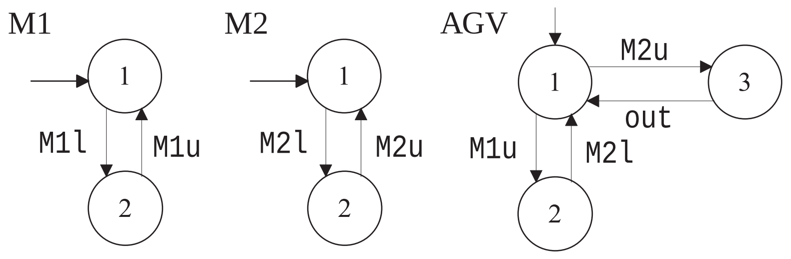
In the Ramadge/Wonham (RW) framework, DES are formally modeled as recognizers of formal languages where the alphabet of the language consists of the system events. The language itself describes the set of trajectories of the system. Some of the events (controllable events) can be directly influenced (disabled) by the supervisor, whereas there is no immediate effect on other events (uncontrollable events). The task of the supervisor is to disable controllable events such that a specified system behavior is obtained. In the RW framework, specifications are given as formal languages and there are algorithms, which compute supervisors that are provably correct. On the one hand, a supervisor has to be able to fulfill the specification by applying its control action to the DES. On the other hand, it has to be nonblocking, that is, it must not lead the system to configurations where no more operation is possible.

In industrial practice, DES are controlled by Programmable Logic Controllers (PLC). A PLC receives output signals from the DES plant and computes the respective input signals in a cyclic fashion. Informally, the program running on the PLC is nothing but a realization of a supervisor. In each program cycle, the output signals of the DES (events) are read and a list of instruction is executed, determining the input signals (enabled events) of the plant for the next cycle.

The main challenge in computing a supervisor in the RW framework is the combinatorial explosion of the state space for large-scale systems. This is due to the fact that a composite system is based on the cartesian product of its subsystems. For large-scale systems, both supervisor computation and PLC implementation become impractical, which is illustrated with the following example.

Limitations of monolithic supervisory control

An example for a discrete event system is the small production cell with two machines (M1 and M2) and one automated guided vehicle (AGV) (see [Won04]). The discrete event models are represented as automata, where nodes (circles) and arrows indicate states and state-transitions, respectively. The two machines can load parts (`M1l` and `M2l`) and unload again (`M1u` and `M2u`). Initially both machines are empty (in state 1). The AGV can accept one part from either machine (`M1u` and `M2u`) and also load the second machine (`M2l`) or remove parts from the production cell (`out`). Automata for the system components and sample PLC code for M1 are shown in Figure 1.2.[1]



```
//*******************
AN "m1unload"//if the machine does not unload
A   "m1l"        //and the event "m1l" happens
S   "m1load"   //the machine is loading
//*******************
AN "m1load"   //if the machine does not load
A   "m1u"        //and the event "m1u" happens
S   "m1unload"//the machine is unloading
```

Figure 1.2: Components of the small production cell

As can be seen from Figure 1.2, the AGV receives parts from machine M1 and M2 (via the shared events `M1u`, `M2l` and `M2u`). Thus, its action is synchronized with the other components of the system, i.e. the events `M1ul`, `M2l` and `M2u` have to occur at the same time. Formally, the system interaction is represented by the synchronous product of the components M1, M2 and AGV, as depicted in Figure 1.3.

[1]The PLC code checks which events occur in the system and the corresponding action is written to the system input (S "m1load" and S "m1unload"). The "A" and "AN" commands stand for "AND" and "AND NOT", respectively.

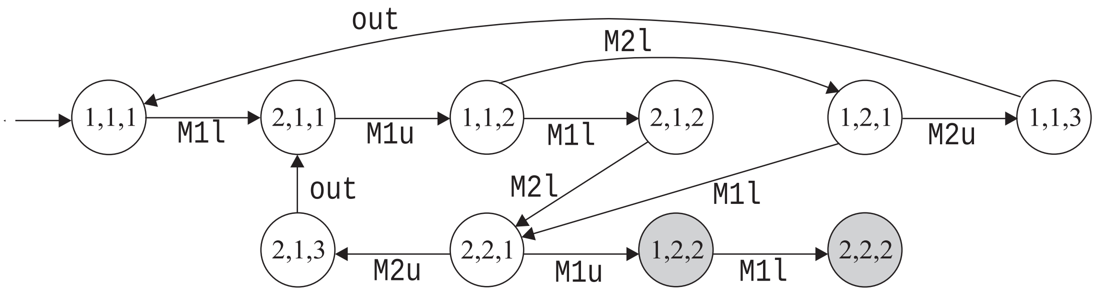


Figure 1.3: Synchronized components of the small production cell

Each state in the synchronous product represents the states which the respective component is in. For example, in state (1,1,2), M1 is in state 1, M2 is in state 1 and AGV is in state 2.

The system in Figure 1.3 is blocking. If the state (2,2,2) (shaded in Figure 1.3) is reached, no further event can occur, that is, the system is stuck. Intuitively, in the blocking state M2 contains a part and is waiting for the AGV to unload. At the same time M1 has already been unloaded, such that AGV is full and cannot receive the part from M2. Formally, there are no transitions possible in the blocking state (2,2,2) and hence it is a "bad" state for the supervisor computation, if it is specified that the supervised system must not contain any blocking. In this case, the supervisor must prevent that the states (1,2,2) and (2,2,2) are reached[2]. Assuming that the event M1u is controllable, it has to disable this event, if the synchronized production cell is in state (2,2,1). This shows that a DES supervisor can prevent blocking with its control action. In the RW framework, such supervisor is computed by efficient algorithms.

The RW framework reaches its computational limit, if systems with many components are considered. As can be seen for the small example, the number of states of a synchronized system grows with the product of the number of states of its components. Although it is possible to handle systems with millions of states, already relatively small manufacturing systems exceed this order of magnitude by far, which makes the application of the monolithic approach infeasible (for example, a composite system with 10 components with 10 states each would have 10^{10} states). Thus, the benefit of computing supervisors which guarantee the specified behavior is paid at the price of dealing with large state spaces.

Yet, experienced programmers manage to write PLC code, and thus implicitly implement DES supervisors for large-scale systems. With their expert knowledge about the system, they apply a "divide and conquer" strategy, paying attention to one system component at a time.

We want to formalize the idea of exploiting the system structure for supervisor synthesis. To this end, we combine decentralized and hierarchical supervisory control approaches.

[2]From (1,2,2) only the "bad state" (2,2,2) can be reached.

Modular and Decentralized Control

The decentralized control architecture is illustrated in Figure 1.4.

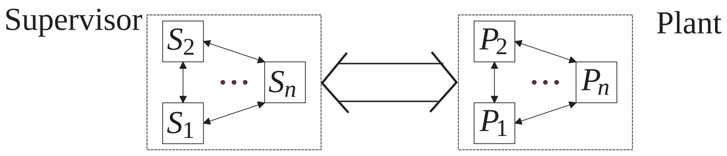


Figure 1.4: Decentralized control of discrete event systems

A first approach to reducing the complexity of supervisor synthesis is *modular control* as elaborated in [RW87a, WR88, RW89]. Monolithic supervisors for different specifications are designed and implemented together. Although controllability can be verified easily, checking if the modular supervisors are nonconflicting, that is, if their joint action is nonblocking, is computationally expensive (see also [RL02]). An improvement of this technique is given in [dQC00, QC00, dQ00], where the plant is considered as a composite system. Controllability and nonconflicting behavior only need to be checked for system components with specified behavior. [GM04] elaborates the modular computation of controllable sublanguages of a specification language using abstractions of the composite plant, to avoid the composition of the system components.

In contrast to the modular approach, *decentralized control* approaches focus on the computation of distributed interacting supervisors as investigated in [CDFV88, LW90, BGK$^+$90]. The decentralized supervisors only have partial observations of the plant events, and to guarantee that specifications can be fulfilled, a property, called co-observability, is needed. There are several ways how to fuse the control actions of the supervisors, such as *conjunctive* or *disjunctive* architectures [Bar99, YL00, Yoo02, YL02]. Recent work also includes communication between the supervisors. The decentralized approach is formulated in a co-algebra setting in [KvS03]. Extensions of the method to nondeterministic systems and to concurrent systems with modular specifications are given in [KS97] and [JK02, JCK01], respectively. Unfortunately, for composite systems, the decentralized method still needs the computation of the overall system model. Thus, its computational effort equals the monolithic approach. Complexity results are provided in [RW95, YL02].

A different view on the decentralized control of concurrent (composite) discrete event systems is taken in [WH91, LW97, LW02, KvS04]. Purely structural conditions of the subsystems of the discrete event plant are used, instead of the conditions on the specification with respect to the system. The overall system does not have to be computed, and thus, after verifying the required system properties, supervisor synthesis can be performed with a smaller computational effort than for the monolithic approach. Because of this reason, the idea of identifying structural system properties is adopted in our work.

Hierarchical Control

The basic idea of hierarchical control is presented in Figure 1.5.

The hierarchical control approach employs system models with different degrees of detail. On the one hand, hierarchical system models can be constructed "bottom-up", that is, the low-level model is abstracted to higher levels by aggregation of information. (see [ZW90, Zho92, WW96, Pu00, dCCK02, HC02, MG02, Led02, MRD03, SRM04, SPM05, SMP05, MS05]. On the other hand, hierarchies can also be built "top-down" as in [BH93, Wan95, Goh03, Ma04, GM05b, GM05a]. Then, a high-level model (for example an automaton) of the system is generated first, and the structural components of the high-level model are filled with more detailed information (for example states of the high-level automaton represent a whole set of states on the lower level). All these approaches have in common that supervisors are designed on the high level and then translated to the low level for implementation.

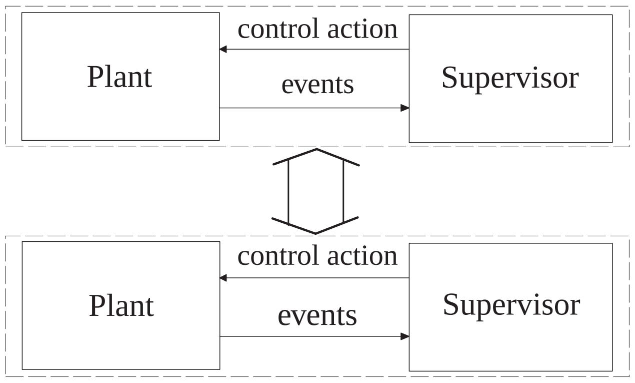


Figure 1.5: Hierarchical Control of discrete event systems

The development of bottom-up hierarchical control techniques was initiated by the work in [ZW90, Zho92]. *Output control consistency* is used as a structural condition to construct the high-level model in the RW framework. The low-level system model is aggregated, and the high-level controllability properties are determined based on local low-level behavior. One drawback of the approach is the fact that marking is not considered and thus low-level supervisors can be blocking. This problem is solved in [WW96] in an algebraic setting by introducing *control structures*. Control structures denote a generalization of the RW framework for representing controllability properties on the high level. *Causal maps* are employed for hierarchical abstraction, and the *observer* property helps guaranteeing nonblocking system behavior. The results in [Pu00] adopt the idea of control structures and observers to present a theory generalizing the RW framework, combined with the corresponding algorithms. In addition to that, decentralized systems without any shared events are investigated. *Consistent* and *reliable* abstractions are required for decentralized supervisor synthesis. The idea of control structures is further elaborated in [dCC02]. On the low level of the hierarchy, the RW framework is used. On the high level, the concept of control structures, equipped with a flexible marking function (see also [CTdC01a, CTdC01b]), is employed. Based on an *assume guarantee reasoning*, this method is extended to decentralized systems without shared events in [dCCK02]. The work in [MRD03, MS05] presents a hierarchical design method for systems with an input/output structure in the behavioral framework, where it is possible to use a high-level specification as the system abstraction.

The approaches in [TC02, HC02] perform an aggregation (partition) of the state space to get an abstraction of the low-level system. Elaborating on the connectivity of the high-level states, nonblocking supervisors are designed.

A method which explicitly uses structural information of the discrete event plant is presented in [LWL01, LLW01, Led02]. It is based on the definition of interfaces, which indicate how the hierarchical levels can interact. Both the serial (monolithic) and the parallel (decentralized) cases are investigated and the computation of nonblocking supervisors is elaborated for large-scale systems. In this approach, the use of structural information is indispensable for handling large systems.

Opposed to the bottom-up methods, the top-down approaches build the hierarchy starting from the high-level system model. Unifying the work in [BH93, Wan95, Goh98, Ma99], the technique in [Ma04] constructs a hierarchy based on state tree structures (STS). Incorporating (AND) and (OR) superstates, the hierarchical model also accounts for large-scale composite systems. The method in [GM05b, GM05a] employs a fixed-point computation to compute nonblocking controllers for hierarchical state machines.

Contribution and Outline of the Thesis

In our work, a bottom-up approach is elaborated for hierarchical control. We use a particular causal map, the *natural projection*, for system abstraction. Different from [ZW90, Zho92, WW96, Pu00, dCCK02, HC02, MG02], the high-level event set is a subset of the low-level event set, and the low-level model is projected on the high-level events. This abstraction method makes it possible to carry over the controllability properties of events from the low level to the high level. Thus, the RW framework can also be employed for the high-level model. As a consequence, the method qualifies for a multi-level hierarchy.

The *consistent implementation* is defined for the low-level realization of high-level supervisors. It guarantees *hierarchical consistency* of the hierarchical architecture by default, that is the low-level closed-loop system behaves as expected in the high level. Moreover, the consistent implementation does not involve an extensive evaluation of controllability results for local behavior like in [Zho92, Pu00, dCCK02, HC02]. For being able to apply this supervisor implementation for nonblocking low-level control, structural conditions are required. The abstraction, which is the natural projection, has to be an observer. In addition to that, we introduce *marked string acceptance* as a relation between the marking on the high level and the low level. Our hierarchical architecture is nonblocking and hierarchically consistent for systems with the above properties.

Our approach is further extended to decentralized systems, for making use of the structure of composite systems. In this setting, we require that all events which are shared by different subsystems have to be carried over to the high level. This makes it possible first to abstract the decentralized system components and then to compose them to a high-level composite system with a low computational effort. As the synchronized behavior of the subsystems is captured in the high-level, the problem of conflicting behavior, which is a main issue in modular approaches, does not occur. Defining a decentralized version of the consistent implementation and requiring the same system properties as above, our hierarchical and decentralized multi-level architecture is nonblocking and hierarchically consistent.

The thesis is organized as follows. In Chapter 2, the basic notation used throughout the thesis is presented. Different from [ZW90, Zho92, WW96, Pu00, dCCK02, HC02, MG02, Led02, SRM04, SPM05, SMP05], which are based on a finite automata formulation, our theoretical results are elaborated in a language framework as defined in Section 2.3. The equivalent automata representation is also given at the end of the chapter. For later use in the hierarchical and decentralized framework, our purely hierarchical method is presented in Chapter 3. The main theorem of this chapter establishes nonblocking hierarchical control if the system is locally nonblocking and marked string accepting. In Chapter 4, we extend the hierarchical architecture to decentralized systems, and we prove that it is nonblocking and hierarchically consistent. In addition to the theoretical results, algorithms for both the verification of system properties and supervisor synthesis are developed along with the respective complexity results. The applicability of the approach to large-scale composite systems is demonstrated with a laboratory case study in Chapter 5, and the performance of the method is evaluated.

Chapter 2

Basics of RW Supervisory Control Theory

Discrete event systems are systems which are discrete in both time and state space. Also changes in the system state occur asynchronously and driven by events rather than by a clock. Examples for discrete event systems are manufacturing systems, networks, digital circuits, communication protocols, etc.

A natural framework for describing such systems are *formal languages* [HU79], where sequences of events form so-called *strings* and there are distinguished strings — words — which represent the event sequences accepted by the discrete event system. *Regular languages* are of particular importance as these languages are recognized by finite automata which can model systems with a finite number of discrete states.

Starting from this, a framework for the control of discrete event systems has been elaborated in [RW87a]. It is called supervisor control theory and its main goal is to synthesize controllers — supervisors — which restrict the possible behavior of the system to some desired behavior without causing blocking, i.e. without the closed loop getting stuck.

It is possible to represent discrete event systems as a set of regular languages or model them as finite automata. For sake of clarity, this work provides a clear separation of both concepts. Theoretical considerations are stated in the regular language framework. As both approaches are equivalent, the formulation of the theoretical results is also given in the automata framework. The automata representation is then used for the algorithmic implementation of the theoretical result.

The chapter is organized as follows. Section 2.1 recalls formal language definitions and states various results on regular languages which will be relevant in this thesis. Automata are introduced in Section 2.2, and the relation between finite automata and regular languages is established. In Section 2.3, a language-based framework for the control of discrete event systems is developed. The computability of the results is outlined by working out an automata representation of the language-based theoretical results presented before.

2.1 Languages

Formal languages are used in different areas of computer science, such as compiler generation, pattern recognition, search algorithms, parser development, etc. In this thesis, formal languages represent the behavior of discrete event systems. Section 2.1.1 gives a short review of the basic notions. Mainly, it refers to the comprehensive introduction to automata and formal languages in [HU79]. A thorough description of the control theoretic ideas is given in [Won04, CL99]. We also construct examples for illustrating the theoretical concepts.

2.1.1 General Definitions

Let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$, $m \in \mathbb{N}$ be a finite set of distinct symbols. Σ is denoted an *alphabet* and arbitrary concatenations $s = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_k}$ of symbols $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_k} \in \Sigma$ with $i_1, \ldots, i_k \in \{1, \ldots, m\}$, are called *strings*, where $k \geq 1$ is the length of the string s. The set of all strings with elements from Σ is written as Σ^+ and the empty sequence (sequence with no symbols) is ε, where $\varepsilon \notin \Sigma$. ε is also called the *empty string*. Together with Σ^+, $\Sigma^* := \Sigma^+ \cup \{\varepsilon\}$ is the *Kleene closure* of Σ.

Using the terms from above, the concept of a *language* over an alphabet Σ can be introduced.

Definition 2.1 (Language [HU79])
Let Σ be an alphabet. A language L over Σ is a set $L \subseteq \Sigma^*$. $\square$

Note that both the empty language $\emptyset$ and the Kleene closure Σ^* are included in this definition. Also observe that there is a distinction between $\emptyset$ (the language with no strings) and ε (the string with no symbols).

As languages are sets, the common set operations such as union, intersection and difference are applicable. Further important operations are the *concatenation*, the *prefix-closure* and the *Kleene-closure* ([HU79]). Let s and t be two strings with $s, t \in \Sigma^*$. The concatenation of s and t is written st, and it holds that $st \in \Sigma^*$. The string s is called a *prefix*, and t is called a *suffix* of st. A language which includes all prefixes of its strings is called *prefix-closed*. For an arbitrary language L, the *prefix-closure* operation yields a language which contains all prefixes of strings in L. The *set of active symbols* $\Sigma(s)$ describes the set of symbols which can extend the string $s \in L$ such that the resulting string is still in L, i.e. $\Sigma(s) := \{\sigma \in \Sigma | s\sigma \in L\}$[1].

The operations defined above for strings are generalized to languages as shown in Definition 2.2.

Definition 2.2 (Language Operations [HU79])
Let $L, K \subseteq \Sigma^*$. The following operations on languages are defined:

[1]Later this set will be referred to as the *active event set*

(i) the concatenation LK of L and K: $LK := \{s \in \Sigma^* | s = uv \text{ with } u \in L \text{ and } v \in K\}$.

(ii) the prefix-closure $\overline{L}$ of L: $\overline{L} := \{s \in \Sigma^* | \exists u \in \Sigma^* \text{ s.t. } su \in L\}$.

(iii) the Kleene-closure of L: $L^* := \{\varepsilon\} \cup L \cup LL \cup LLL \cdots$.

□

The following example illustrates the concept of formal languages and the operations defined above.

Example 2.1
Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ be an alphabet. $s = \mathtt{ab}$ and $t = \mathtt{a}$ are strings over Σ and $st = \mathtt{aba}$ is the concatenation of s and t. The Kleene-closure of Σ can be represented as $\Sigma^* = \{\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{aa}, \mathtt{ab}, \mathtt{ac}, \mathtt{ba}, \mathtt{bb}, \mathtt{bc}, \mathtt{ca}, \mathtt{cb}, \mathtt{cc}, \mathtt{aaa}, \ldots\}$. An example for a language over the alphabet Σ is $L = \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$ with the words (strings) `ab`, `aba`, `aca` and `abca`. L is not a prefix-closed language as the prefix `a` of `ab` is not contained in L. The prefix-closure of L is $\overline{L} = \{\varepsilon, \mathtt{a}, \mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$. The concept of a language is illustrated in Figure 2.1. Strings are represented as lines, and ticks symbolize the symbols which are concatenated to form a string. Strings are always read from left to right. The line enclosing a set of strings represents the corresponding language. □

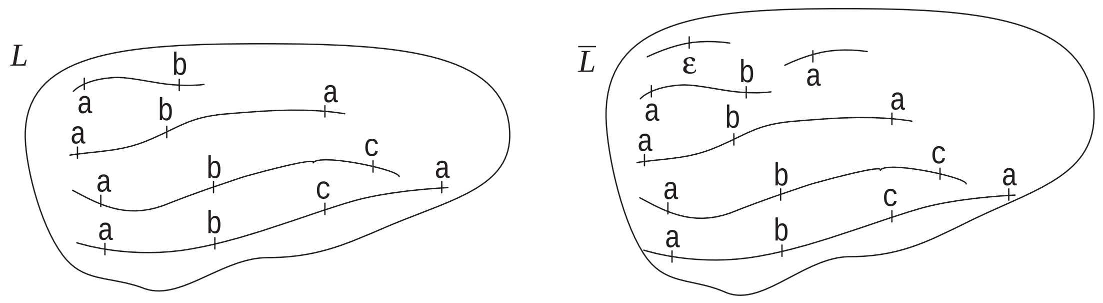


Figure 2.1: Illustration of a formal language L and the prefix closure $\overline{L}$.

The *natural projection* from Σ^* to Σ_0^* for two alphabets Σ and Σ_0 with $\Sigma_0 \subseteq \Sigma$ is defined as follows.

Definition 2.3 (Natural Projection [Won04])
Let $\Sigma_0 \subseteq \Sigma$. The natural projection $p_0 : \Sigma^* \to \Sigma_0^*$ is defined recursively.

$$\begin{aligned} p_0(\varepsilon) &:= \varepsilon \\ p_0(\sigma) &:= \begin{cases} \sigma & \text{if } \sigma \in \Sigma_0 \\ \varepsilon & \text{else} \end{cases} \\ p_0(s\sigma) &:= p_0(s)p_0(\sigma) \end{aligned}$$

for $s \in \Sigma^*$ and $\sigma \in \Sigma$. □

This means that the natural projection erases symbols in a string with elements from the larger symbol set Σ if they do not belong to the smaller symbol set Σ_0. Note that the projection operation is an increasing monotonic function on sets.[2] There is also an inverse map corresponding to the natural projection.

Definition 2.4 (Inverse Projection [Won04])
Let $\Sigma_0 \subseteq \Sigma$. The inverse projection $(p_0)^{-1} : \Sigma_0^* \to 2^{\Sigma^*}$ is

$$(p_0)^{-1}(t) := \{s \in \Sigma^* | p_0(s) = t\}$$

for $t \in \Sigma_0^*$. $\Box$

The natural projection p_0 and the inverse projection $(p_0)^{-1}$ can be generalized to languages $L \in \Sigma^*$ and $L_0 \in \Sigma_0^*$, respectively, by applying them to all strings of the given language:

$$\begin{aligned} p_0(L) &:= \{t \in \Sigma_0^* | \exists s \in L \text{ s.t. } p_0(s) = t\}, \\ (p_0)^{-1}(L_0) &:= \{s \in \Sigma^* | \exists t \in L_0 \text{ s.t. } p_0(s) = t\}. \end{aligned}$$

Note that $L \subseteq (p_0)^{-1}(p_0(L))$ and it is often the case that $L \subset (p_0)^{-1}(p_0(L))$. The computation of the natural projection and its inverse is shown in the next example.

Example 2.2
Let $L = \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$ be the language from the previous example. The alphabet is $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, and the alphabet $\Sigma_0 = \{\mathtt{a}, \mathtt{b}\}$ is chosen as output alphabet for the natural projection $p_0 : \Sigma^* \to \Sigma_0^*$. Considering all strings in L, the corresponding projected strings are $p_0(\mathtt{ab}) = p_0(\mathtt{a})p_0(\mathtt{b}) = \mathtt{ab}$, $p_0(\mathtt{aba}) = p_0(\mathtt{ab})p_0(\mathtt{a}) = \mathtt{aba}$, $p_0(\mathtt{abc}) = p_0(\mathtt{ab})p_0(\mathtt{c}) = \mathtt{ab}$ and $p_0(\mathtt{abca}) = p_0(\mathtt{abc})p_0(\mathtt{a}) = \mathtt{aba}$. Thus, the projected language is $p_0(L) = \{\mathtt{ab}, \mathtt{aba}\}$. The projection is illustrated in Figure 2.2. The convention of Example 2.1 is used except for symbols from the set Σ_0. These symbols are represented by crosses. The dashed lines indicate strings in L which are projected to the respective string in $p_0(L)$.

For describing the inverse projection, $(p_0)^{-1}(p_0(L))$ shall be evaluated. As an example, $(p_0)^{-1}(\mathtt{ab}) = \{\mathtt{c}^*\mathtt{a}\,\mathtt{c}^*\mathtt{b}\,\mathtt{c}^*\}$ and by analogous computations, $(p_0)^{-1}(p_0(L)) = \{\mathtt{c}^*\mathtt{a}\,\mathtt{c}^*\mathtt{b}\,\mathtt{c}^*, \mathtt{c}^*\mathtt{a}\,\mathtt{c}^*\mathtt{b}\,\mathtt{c}^*\mathtt{a}\,\mathtt{c}^*\}$. This result indicates that the inverse projection of a projected language indeed includes the original language. $\Box$

[2] The partial order on the input and output sets is the set inclusion, i.e. if $L_1 \subseteq L_2 \subseteq \Sigma^*$, then $p_0(L_1) \subseteq p_0(L_2) \subseteq \Sigma_0^*$.

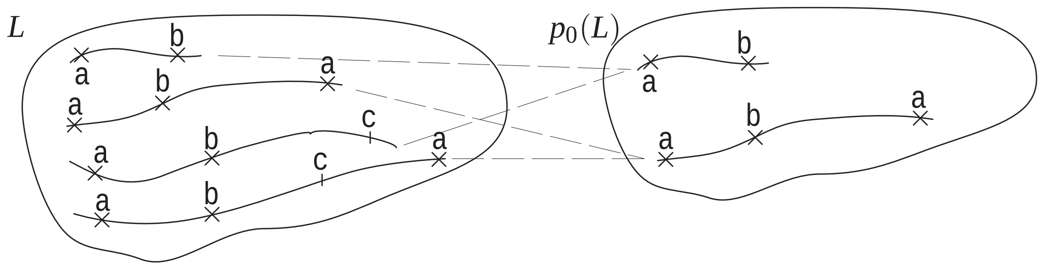


Figure 2.2: Projection of languages.

In the next Lemma a useful property of the natural projection is given. The projection of the prefix-closure of a language equals the prefix-closure of the projected language.

Lemma 2.1 (Prefix-Closure of the Natural Projection [dQ00])
Let $L \subseteq \Sigma^*$ be a language and let $p_0 : \Sigma^* \to \Sigma_0^*$ be the natural projection with $\Sigma_0 \subseteq \Sigma$. Then $p_0(\overline{L}) = \overline{p_0(L)}$. □

An important operation for languages is the *synchronous product* which can be introduced using the inverse projection.

Definition 2.5 (Synchronous Product [Won04, CL99])
Given $\Sigma = \Sigma_1 \cup \Sigma_2$ and the natural projections $p_1 : \Sigma^* \to \Sigma_1^*$, $p_2 : \Sigma^* \to \Sigma_2^*$, the synchronous product of two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ is:

$$L_1||L_2 := p_1^{-1}(L_1) \cap p_2^{-1}(L_2).$$

□

The intersection of the languages $p_1^{-1}(L_1)$ and $p_2^{-1}(L_2)$ in Definition 2.5 ensures that the projection $p_1(s)$ is an element of L_1 and the projection $p_2(s)$ is an element of L_2 for any string s in $L_1||L_2$.

The concept of the synchronous product is explained in the subsequent example.

Example 2.3
Let $L_1 = \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$ and $L_2 = \{\mathtt{ad}\}$ with alphabets $\Sigma_1 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $\Sigma_2 = \{\mathtt{a}, \mathtt{d}\}$, respectively. Then the overall alphabet is $\Sigma = \Sigma_1 \cup \Sigma_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$. Using the natural projections defined in Definition 2.5 results in $(p_1)^{-1}(L_1) = \{\mathtt{d}^*\mathtt{a}\mathtt{d}^*\mathtt{b}\,\mathtt{d}^*, \mathtt{d}^*\mathtt{a}\,\mathtt{d}^*\mathtt{b}\,\mathtt{d}^*\mathtt{a}\,\mathtt{d}^*, \mathtt{d}^*\mathtt{a}\,\mathtt{d}^*\mathtt{b}\,\mathtt{d}^*\mathtt{c}\,\mathtt{d}^*, \mathtt{d}^*\mathtt{a}\,\mathtt{d}^*\mathtt{b}\,\mathtt{d}^*\mathtt{c}\,\mathtt{d}^*\mathtt{a}\,\mathtt{d}^*\}$, and $(p_2)^{-1}(L_2) = \{\{\mathtt{b}, \mathtt{c}\}^*\mathtt{a}\{\mathtt{b}, \mathtt{c}\}^*\mathtt{d}\{\mathtt{b}, \mathtt{c}\}^*\}$. Thus, the synchronous product of the two languages is $L_1||L_2 = (p_1)^{-1}(L_1) \cap (p_2)^{-1}(L_2) = \{\mathtt{abd}, \mathtt{adb}, \mathtt{abcd}, \mathtt{abdc}, \mathtt{adbc}\}$. □

2.1.2 Regular Languages

The theory of formal languages distinguishes four classes of languages which are also known as the Chomsky Hierarchy [HU79]. In our work, the class of *regular languages* is considered in detail. Regular languages can be represented by *regular expressions*.

Definition 2.6 (Regular Expression and Regular Language [HU79, CL99])
Let Σ be an alphabet. The regular expressions over Σ and the *regular languages* they describe are defined recursively.

(i) $\emptyset$ is a regular expression.

(ii) ε is a regular expression, and it denotes the language $\{\varepsilon\}$.

(iii) for any $\sigma \in \Sigma$, σ is a regular expression, representing the language $\{\sigma\}$.

(iv) If r and s are regular expressions, characterizing the languages R and S, respectively, then the operations $(r+s)$, rs and r^* are regular expressions denoting the languages $R \cup S$, RS and R^*.

(v) there are no regular expressions other that those constructed by applying rules (i) to (iv) a finite number of times.

The regular language represented by a regular expression is the set of strings which is expressed by the regular expression. $\square$

In the previous section, different operations on languages were introduced. The operations which conserve regularity are enumerated in the following lemma.

Lemma 2.2 (Operations on Regular Languages [HU79])
Let Σ be an alphabet and R and S be regular languages over Σ. Then $R \cup S$, $R \cap S$, R^* and $\Sigma^* - R$ are regular languages. $\square$

A proof of this lemma is given in [HU79].

Regular languages play a crucial role in describing the behavior of dynamical systems in this work. Before relating the concept of regular languages to an automata representation in the next section, the following lemma states a well-known result on the projection of regular languages [Won97].

Lemma 2.3 (Projection of a Regular Language)
Let $L \subseteq \Sigma^*$ be a regular language, and let $p_0 : \Sigma^* \to \Sigma_0^*$ with $\Sigma_0 \subseteq \Sigma$ be the natural projection. Then $p_0(L)$ is regular. $\square$

An outline of the proof of this result is provided in Appendix A.

In the following example, the regular languages presented in the previous examples will be generated by regular expressions.[3]

Example 2.4
The language $L = \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$ from Example 2.1 is a regular language which can be expressed by the regular expression $l = \mathtt{ab}(\varepsilon + \mathtt{a} + \mathtt{c} + \mathtt{ca})$. As L is regular, its projection on the symbol set $\{\mathtt{a}, \mathtt{b}\}$ according to Example 2.2 must also be regular. The regular expression for $p_0(L) = \{\mathtt{ab}, \mathtt{aba}\}$ is $l_p = \mathtt{ab}(\varepsilon + \mathtt{a})$. □

2.2 Automata

In the previous section, regular languages were introduced by using regular expressions. Another very important tool for representing regular languages are finite automata. In the sequel, the general notion of an automaton is introduced, and the relation between finite automata and regular languages is established.

Definition 2.7 (Automaton [HU79])
An *automaton* is a 5-tuple $G := (X, \Sigma, \delta, X_0, X_\mathrm{m})$. X is the set of states and Σ is the finite set of symbols which is also referred to as the *alphabet*. The transition function $\delta : X \times \Sigma \to 2^X$ is a partial function, i.e. it is only defined for a subset of Σ in any state $x \in X$. The initial state set of the automaton is $X_0 \subseteq X$, and $X_\mathrm{m} \subseteq X$ is the set of marked states, that is a set of distinguished states[4]. The automaton is denoted *finite state* if the number of states is finite. If the initial state set consists of one state, i.e. $X_0 = x_0$ and if the transition function is unique, i.e. $\delta : X \times \Sigma \to X$, then the automaton is called *deterministic*. □

The following example illustrates the concept of a finite state automaton.

Example 2.5
Let $G = (\Sigma, X, \delta, x_0, X_\mathrm{m})$ be a deterministic finite state automaton. A graphical representation of G is shown in Figure 2.3. Nodes in the graph represent states of the automaton and arrows between the states denote transitions between states according to the transition function. The alphabet of G is $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, the state set is $X = \{1, 2, 3, 4, 5\}$, the initial state is $x_0 = 1$ and the marked state set is $X_\mathrm{m} = \{3, 4, 5\}$. The transition function δ states that $\delta(1, \mathtt{a}) = 2$ and $\delta(2, \mathtt{b}) = 3$ for example. □

[3] Note that there are many ways of representing the same regular expression.

[4] For example, a marked state can represent the termination of a task in a discrete event system model.

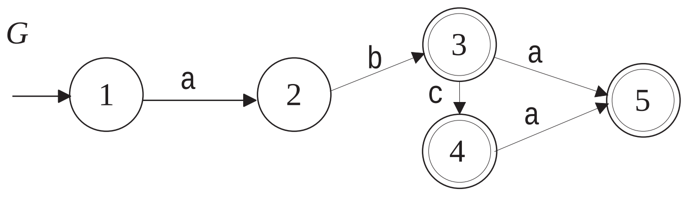


Figure 2.3: Automaton graph of G

For convenience, $\delta(x,\sigma)!$ expresses that δ is defined for σ at state x, and the *set of active symbols* is defined as $\Lambda(x) := \{\sigma \in \Sigma | \delta(x,\sigma)!\}$ for states $x \in X$. The transition function δ can be extended to a partial function on $2^X \times \Sigma^*$. Recursively let $\delta(X,\varepsilon) := X$ and define $\delta(X,s\sigma) := \{x' \in X | x' \in \delta(x'',\sigma) \text{ for } x'' \in \delta(x,s)\}$. For deterministic automata, this definition simplifies to a partial function on $X \times \Sigma^*$ with $\delta(x,\varepsilon) = x$ and $\delta(x,s\sigma) = \delta(\delta(x,s),\sigma)$, whenever both $x' = \delta(x,s)$ and $\delta(x',\sigma)!$. This means that for deterministic automata, a string s which is defined in a state $x \in X$ always leads to a unique successor state $x' = \delta(x,s) \in X$.

The definition of languages of an automaton is based on the directed paths which can be followed in an automaton.

Definition 2.8 (Generated and Marked Language [Won04])
Let $G = (X,\Sigma,\delta,x_0,X_\mathrm{m})$ be an automaton. The *language generated* by G is

$$L(G) := \{s \in \Sigma^* | \delta(x_0,s) \text{ is defined for some } x_0 \in X_0\}.$$

The *language marked* by G is

$$L_\mathrm{m}(G) := \{s \in \Sigma^* | \delta(x_0,s) \in X_\mathrm{m} \text{ for some } x_0 \in X_0\}.$$

□

That is, the generated language includes all sequences of symbols which can be followed in the automaton starting from an initial state. The marked language contains all sequences of symbols which lead from an initial state to a marked state. In particular, it is readily observed that $L_\mathrm{m}(G) \subseteq L(G)$. An automaton G *generates* the language $L(G)$ and *recognizes* the language $L_\mathrm{m}(G)$.

Example 2.6
The marked language of the automaton G in Figure 2.3 is $L_\mathrm{m}(G) = \{\mathtt{ab},\mathtt{aba},\mathtt{abc},\mathtt{abca}\}$, and the prefix-closed language is $L(G) = \{\varepsilon,\mathtt{a},\mathtt{ab},\mathtt{aba},\mathtt{abc},\mathtt{abca}\}$. □

Having defined the language marked by an automaton, it is interesting to ask which type of languages can be spoken by a finite automaton. To this end, the *Nerode equivalence* on languages which defines an equivalence relation on strings is recalled.

Definition 2.9 (Nerode Equivalence [Ner58])
The Nerode equivalence relation on Σ^* with respect to $L \subseteq \Sigma^*$ (or $\bmod L$) is defined as follows. For $s,t \in \Sigma^*$,

$$s \equiv_L t \text{ or } s \equiv t \bmod L \quad \text{iff} \quad \forall u \in \Sigma^* : su \in L \text{ iff } tu \in L.$$

□

This means that two strings $s,t \in \Sigma^*$ are in the same equivalence class of the Nerode equivalence iff they can be continued to a word in L in exactly the same way. The cardinality or *index* of the Nerode equivalence relation is denoted $||L||$. In case that $||L|| < \infty$, there is a finite number of equivalence classes. In this case, the Myhill-Nerode Theorem shows that it is possible to represent L by a *finite automaton* which recognizes L.

Theorem 2.1 (Myhill-Nerode Theorem [Ner58])
The following statements are equivalent

(i) The set $L \in \Sigma^*$ is recognized by a finite automaton

(ii) L is the union of some equivalence classes of a right-invariant equivalence relation with finite index

(iii) Let the equivalence relation $\equiv_L$ be defined as in Definition 2.9. Then $\equiv_L$ is of finite index.

□

Thus, item (i) and item (iii) show that if the Nerode equivalence relation for a language L yields a finite set of equivalence classes, then L can be represented by a finite automaton. In the proof of Theorem 2.1, [HU79] provides a procedure for constructing such automaton. It is interesting to note that if the language L is recognized by a finite automaton, then there exists a minimal automaton[5] recognizing L [HU79].

Theorem 2.2 (Minimal Automaton [HU79])
The minimal automaton recognizing L is unique except for an isomorphism. □

There are algorithms for computing the minimal automaton for a given language L (see also [HU79, Hop71]), and the resulting automaton is referred to as the *canonical recognizer* of the language L. In the canonical recognizer, each state represents an equivalence class of the Nerode equivalence on Σ^* w.r.t L.

In addition to relating finite automata to Nerode equivalence classes of languages with finite index, a very useful property of deterministic finite automata is stated in the following theorem.

[5]In this context "minimal" means "minimum state".

Theorem 2.3 (Finite Automata and Regular Languages [HU79])
If the language $L \in \Sigma^*$ is recognized by a finite automaton, then L is regular. Also if L is a regular language, then there exists a finite automaton which recognizes L □

The second statement does not say whether the resulting automaton is deterministic or nondeterministic. Yet, the following result states that there is always a deterministic automaton which recognizes the language that is marked by a nondeterministic automaton.

Lemma 2.4 (Deterministic Automaton for a Nondeterministic Automaton [HU79])
Let L be a language which is recognized by a nondeterministic finite automaton G_{nd}. Then there exists a deterministic finite automaton G_{d} which recognizes L, i.e. $L_{\text{m}}(G_{\text{d}}) = L_{\text{m}}(G_{\text{nd}})$. □

With Lemma 2.4 a useful corollary can be established.

Corollary 2.1 (Regular Languages and Deterministic Finite Automata)
If $L \in \Sigma^*$ is a regular language, then there exists a deterministic finite automaton G with $L_{\text{m}}(G) = L$. □

Proof: Corollary 2.1 follows with the second statement in Theorem 2.3 and Lemma 2.4. □

Consequently, a regular language can be represented by a deterministic finite automaton. Analogously to Definition 2.5 for languages, there is a *synchronous composition* operation for automata.[6] It computes an automaton representing the common behavior of two given automata. [7]

Definition 2.10 (Synchronous Composition)
The synchronous product of two deterministic automata $G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1}, X_{m,1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2}, X_{m,2})$ is

$$G_1||G_2 := (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \delta_{1||2}, (x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$$

with

$$\delta_{1||2}((x_1,x_2),\sigma) := \begin{cases} (\delta_1(x_1,\sigma), \delta_2(x_2,\sigma)) & \text{if} \quad \sigma \in \Lambda_1(x_1) \cap \Lambda_2(x_2) \\ (\delta_1(x_1,\sigma), x_2) & \text{if} \quad \sigma \in \Lambda_1(x_1) - \Sigma_2 \\ (x_1, \delta_2(x_2,\sigma)) & \text{if} \quad \sigma \in \Lambda_2(x_2) - \Sigma_1 \\ \text{undefined} & \text{else} \end{cases}$$

□

[6]This operation is also referred to as the *parallel composition*.

[7]Here, the synchronous product is defined for deterministic automata as a more general definition for nondeterministic automata is not required in this thesis.

This means that a *shared symbol* $\sigma \in \Sigma_1 \cap \Sigma_2$ can occur at a state of the composed automaton $G_1||G_2$ only if it is in the set of active symbols of both of the respective states of G_1 and G_2 (synchronization), while the rest of the symbols can occur whenever they are generated by G_1 or G_2. A state of the resulting automaton is marked only if both respective states of G_1 and G_2 are marked. Also note that not all states in the canonical products $X_1 \times X_2$ and $X_{m,1} \times X_{m,2}$ need to be reachable from the initial state.

Definition 2.10 is closely related to Definition 2.5. It is a well known fact from the literature ([Won04, CL99]) that the language generated by the synchronous composition of two automata $L(G_1||G_2)$ equals the synchronous product of the generated languages $L(G_1)||L(G_2)$.

Lemma 2.5
Let $G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1}, X_{m,1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2}, X_{m,2})$ be deterministic automata. Then it holds that

$$L(G_1)||L(G_2) = L(G_1||G_2) \text{ and } L_\mathrm{m}(G_1)||L_\mathrm{m}(G_2) = L_\mathrm{m}(G_1||G_2).$$

□

The synchronous composition of automata is explained in the subsequent example.

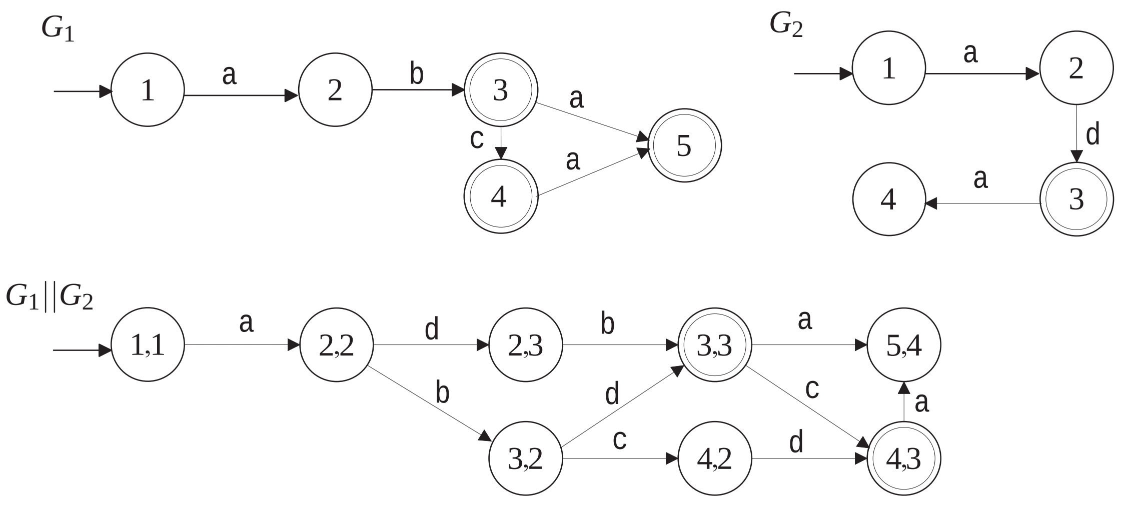


Figure 2.4: Synchronous composition of G_1 and G_2

Example 2.7
The synchronous composition of the automata G_1 and G_2, recognizing the languages $L_1 = \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$ and $L_2 = \{\mathtt{ad}\}$ in Example 2.3 is computed. Note that $\overline{L_\mathrm{m}(G_2)} \subset L(G_2)$ in this case, i.e. the generated language contains more strings than the prefix-closure of the marked language of G_2. The corresponding automata graphs are depicted in Figure 2.4. The alphabet of $G_1||G_2$ is $\Sigma_{1||2} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$. Not all states in $X_{1||2} = X_1 \times X_2$ are reachable from the initial state. Only reachable states are shown in the automaton graph of $G_1||G_2$. Also note that

states in $G_1||G_2$ are only marked if the corresponding states in G_1 and G_2 are both marked. For example, observe that in state $(3,2)$, the state 3 is marked in G_1 but 2 is not marked in G_2 and thus $(3,2)$ is not marked in $G_1||G_2$. Further on, the language recognized by $G_1||G_2$ is $L_\mathrm{m}(G_1||G_2) = \{\mathtt{abd}, \mathtt{adb}, \mathtt{adbc}, \mathtt{abdc}, \mathtt{abcd}\}$. This is equal to the language $L_1||L_2$ computed in Example 2.3, which complies with Lemma 2.5. □

This section introduced the notion of automata and related it to the concept of formal languages. It holds that any regular language can be recognized by a finite automaton and that operations such as the synchronous composition of languages can be computed by evaluating the synchronous composition of the corresponding automata.

2.3 Supervisory Control in a Language based Framework

In the previous sections, general definitions of regular languages and finite automata were given. Now, these notions are used to describe discrete event systems (DES), and in addition to that a framework for the control of these systems is established. To this end, we present the approach introduced in [RW87b] in a language framework. The equivalent formulation using finite automata is given in the next section.

2.3.1 Basic Definitions

In our work, we formally describe discrete systems as *control systems*. Similar to [Rut99], they consist of a pair of languages which fulfill certain requirements.

Definition 2.11 (Control System)
Let Σ be a set of symbols, also denoted *events*, and let $L_1, L_2 \subseteq \Sigma^*$ be two languages. The tuple $H = (L_1, L_2)$ is called a control system (CS) if

(i) L_1 and L_2 are regular,

(ii) L_1 is prefix-closed,

(iii) $L_2 \subseteq L_1$,

(iv) $\Sigma = \Sigma_\mathrm{uc} \dot{\cup} \Sigma_\mathrm{c}$, where Σ_uc is called the set of *uncontrollable events* and Σ_c is the set of *controllable events*.

□

The set Σ contains the events which can occur in a DES. The behavior of the control system is represented by the sequences of events which can happen. In this regard, the language L_2 describes the desirable strings of the system (for example strings which indicate the termination of a task). We refer to these strings as *marked strings*. L_1 includes all strings which can be generated by the system (in particular prefixes of strings in L_2). In addition to that, the event set is partitioned into controllable (Σ_c) and uncontrollable (Σ_{uc}) events. Controllable events can be prevented from occurring (e.g. actuator signals in a manufacturing system), and uncontrollable events Σ_{uc} cannot be disabled (e.g. sensor signals).

Considering the languages L_1 and L_2, we define a subset relation for control systems. This subset relation is useful for stating properties of control systems.

Definition 2.12 (Subset Relation for Control Systems)
Let $H_1 = (L_{1,1}, L_{1,2})$ and $H_2 = (L_{2,1}, L_{2,2})$ be control systems over the set of symbols Σ. $H_1 \subseteq H_2$ iff $L_{1,1} \subseteq L_{2,1}$ and $L_{1,2} \subseteq L_{2,2}$. □

This means a control system is a subset of another control system if its languages are subsets of the respective languages of the other control system.

Example 2.8 explains the notions defined above.

Example 2.8
Let $H = (L_1, L_2)$ with $L_1 = \{\varepsilon, \mathtt{a}, \mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$ and $L_2 = \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$. Also define the uncontrollable event set $\Sigma_{uc} = \{\mathtt{a}, \mathtt{b}\}$, and the controllable event set $\Sigma_c = \{\mathtt{c}\}$. It is readily observed that $\Sigma = \Sigma_{uc} \cup \Sigma_c$, and $\Sigma_{uc} \cap \Sigma_c = \emptyset$. As shown in Example 2.4, L_1 and L_2 are regular. It also holds that L_1 is prefix-closed and $L_2 \subseteq L_1$. Thus, H is a control system. A graphical representation of the two languages is given in Figure 2.1 with $L_2 = L$ and $L_1 = \overline{L}$. The illustration of control systems is facilitated (see Figure 2.5). If prefixes $s' < s$ of a string s are contained in the language L_1, then only the string s is shown. Also, strings in the language L_2 are not depicted explicitly, but they are marked with circles around ticks. For example, the prefix ab of the string aba is not drawn as a separate string, and the corresponding tick, labeled with b, is marked with a circle, as ab is an element of L_2. □

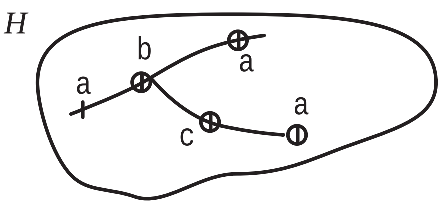


Figure 2.5: Control system H

The system behavior can be influenced by disabling or enabling controllable events according to Definition 2.11. Formalizing the idea of imposing control actions on a control system, the partition of the event set into controllable and uncontrollable events is used to introduce the concept of *control patterns*.

Definition 2.13 (Control Pattern and Set of Control Patterns [Won04])
Let $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$ be an event set with the controllable events Σ_c and the uncontrollable events Σ_{uc}. A *control pattern* is a set γ with $\Sigma_{uc} \subseteq \gamma \subseteq \Sigma$. The set of all control patterns is $\Gamma := \{\gamma | \Sigma_{uc} \subseteq \gamma \subseteq \Sigma\} \subseteq 2^{\Sigma}$. [8] $\square$

Analogous to [RW87b], we define the concept of a supervisor for a DES, where the events in a control pattern are regarded as enabled events.[9]

Definition 2.14 (Supervisor [RW87b])
Let $H = (L_1, L_2)$ be a control system. A *supervisor* S for H is a map

$$S : L_1 \rightarrow \Gamma,$$

where $S(s)$ represents the set of enabled events after the occurrence of a string $s \in L_1$. $\square$

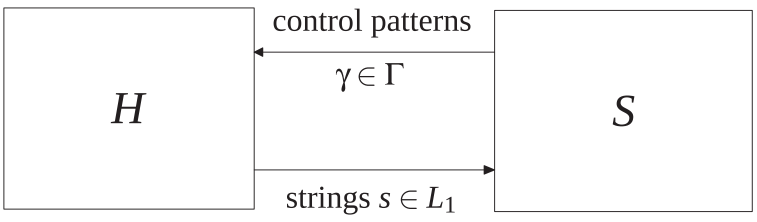


Figure 2.6: Feedback loop with the control system H and the supervisor S

The supervisor S together with the control system H are used in a feedback loop as depicted in Figure 2.6. That is, a supervisor observes the events occurring in the system H, and it can disable controllable events after any string $s \in L_1$, depending on the control pattern $\gamma \in \Gamma$ returned after the current string $s \in L_1$. It is important to note that the supervisor is not allowed to disable uncontrollable events according to the definition of the control patterns. A supervisory controller reduces the behavior of a system H to a smaller behavior S/H.[10] The languages of S/H are constructed iteratively.

[8] 2^{Σ} is the power set of Σ.
[9] Note that uncontrollable events are always enabled.
[10] S/H can be read as "S controlling H".

Definition 2.15 (Closed Loop Languages [RW87b])
Let H be an control system, let S be a supervisor and write $S/H = (L_1^c, L_2^c)$.[11] The closed loop language L_1^c is defined as

$$\begin{aligned}&\varepsilon \in L_1^c,\\ &s\sigma \in L_1^c \text{ iff } s\sigma \in L_1, s \in L_1^c \text{ and } \sigma \in S(s),\end{aligned}$$

and the closed loop language L_2^c is

$$L_2^c := L_1^c \cap L_2.$$

□

The next example describes the operation of a supervisor for the control system in Example 2.8.

Example 2.9
We define a supervisor S for the control system H in Example 2.8 for strings $s \in L_1$ as

$$S(s) := \begin{cases} \{\mathtt{a}, \mathtt{b}\} & \text{if } s = \mathtt{ab} \\ \{\mathtt{a}, \mathtt{b}, \mathtt{c}\} & \text{otherwise.} \end{cases}$$

This means S disables the event $\mathtt{c}$ after the occurrence of the string $\mathtt{ab}$. The resulting closed-loop behavior is $S/H = (L_1^c, L_2^c)$ with $L_1^c = \{\varepsilon, \mathtt{a}, \mathtt{ab}, \mathtt{aba}\}$, and $L_2^c = \{\mathtt{ab}, \mathtt{aba}\}$. □

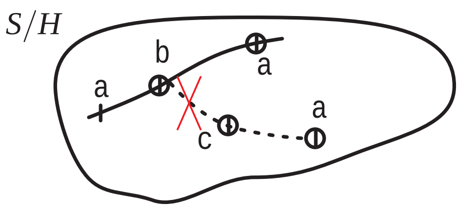


Figure 2.7: Supervisor and closed loop language

2.3.2 Controllability and Nonblocking Control

In the previous section, control systems and supervisors were introduced and it was pointed out, that control actions can be applied, yielding a reduced system behavior. However, the question how it is possible to control the system such that the closed loop system assumes some specified behavior remained unanswered. In the supervisory control context, desired system properties are formulated as regular languages. From these *specifications*, a supervisor which implements the specified behavior can be computed as shown in the sequel. At first *controllability* is introduced.

[11]The superscript "c"means "controlled".

Definition 2.16 (Controllability [RW87b])
Let $L = \overline{L} \subseteq \Sigma^*$ be a prefix-closed language, and let $\Sigma_{\text{uc}} \subseteq \Sigma$ be the set of uncontrollable events. The language $E \subseteq L$ is said to be *controllable* w.r.t. L, and the set of uncontrollable events Σ_{uc} if

$$\overline{E}\Sigma_{\text{uc}} \cap L \subseteq \overline{E}.$$

□

We write E is controllable w.r.t. L if the set of uncontrollable events is obvious from the context. The above condition states that if a string $s \in \overline{E}$ is extended with an uncontrollable event $\sigma \in \Sigma_{\text{uc}}$ such that $s\sigma$ is also in L, then $s\sigma$ must also be element of $\overline{E}$, i.e. it must not be prevented. Considering this, it can be shown that if $L = L_1$ for a control system $H = (L_1, L_2)$, then there exists a supervisor S such that $L_1^{\text{c}} = \overline{E}$ if E is controllable w.r.t. L_1.

Lemma 2.6 (Controllability [RW87b])
Let $H = (L_1, L_2)$ be a control system and let $E \subseteq L_1$ be a specification language. There exists a supervisor $S : L_1 \to \Gamma$ such that $L_1^{\text{c}} = \overline{E}$ for $S/H = (L_1^{\text{c}}, L_2^{\text{c}})$ iff E is controllable w.r.t. L_1. □

A proof for Lemma 2.6 is provided in [CL99]. Precisely, iff E is controllable with respect to L_1, then a supervisor S exists which restricts the control system $H = (L_1, L_2)$ such that $L_1^{\text{c}} = \overline{E}$.

In addition to controllability, *nonblocking behavior* is a further desirable property of a control system. We say that the control system $H = (L_1, L_2)$ is nonblocking if the prefix-closure of L_2 equals L_1.

Definition 2.17 (Nonblocking Control System)
Let $H = (L_1, L_2)$ be a control system. H is nonblocking if

$$\overline{L}_2 = L_1.$$

□

This means that every string s generated by the system ($s \in L_1$) can be extended to a string in L_2 ($\exists u \in \Sigma^*$ s.t. $su \in L_2$). Thus, a supervised system $S/H = (L_1^{\text{c}}, L_2^{\text{c}})$ is nonblocking if $\overline{L_2^{\text{c}}} = L_1^{\text{c}}$.

Up to now, it has only been considered that E is a subset of the language L_1 and a controllability result has been shown for this case. Yet, by controlling H, it might happen that the closed loop system $S/H = (L_1^{\text{c}}, L_2^{\text{c}})$ is blocking, i.e. controlling the system might lead to blocking behavior. The following theorem states conditions for the case that a specification language is controllable with respect to the control system H and at the same time the closed loop system is nonblocking. It is required that $E \subseteq L_2$.

Theorem 2.4 (Nonblocking Controllability Theorem [RW87b])
Let $H = (L_1, L_2)$ be a control system and let $E \subseteq L_2$ be a specification language. There exists a supervisor $S : L_1 \to \Gamma$ such that $L_2^c = E$ and $L_1^c = \overline{E}$ iff

(i) E is controllable w.r.t L_1,

(ii) $E = \overline{E} \cap L_2$. [12]

$\square$

A proof is given in [Won04]. Condition (ii) is denoted the L_2-closure and E is called L_2-closed if it fulfills the condition. For convenience, the set of L_2-closed languages is written $\mathcal{F}_{L_2}$. The above theorem only applies if the language E already complies with the properties (i) and (i). Now, the question is, what can be done if the above properties are not fulfilled.

In case that a specification language E is not controllable w.r.t a language L, it is interesting to investigate controllable sublanguages of E. At first the set of controllable sublanguages of a given language L is defined.

Definition 2.18 (Set of Controllable Sublanguages [WR87])
Let $L = \overline{L} \subseteq \Sigma^*$ be a prefix-closed language and $\Sigma_{\text{uc}} \subseteq \Sigma$ be the set of uncontrollable events. The set $\mathcal{C}(L)$ of controllable languages w.r.t. L and Σ_{uc} is

$$\mathcal{C}(L) = \{K \subseteq L |\ \overline{K}\Sigma_{\text{uc}} \cap L \subseteq \overline{K}\}.$$

$\square$

The set $\mathcal{C}(L)$ is closed under arbitrary union. Hence, for every specification language E, there exists a unique *supremal controllable sublanguage* of E w.r.t L.

Definition 2.19 (Supremal Controllable Sublanguage [WR87])
Let $L \subseteq \Sigma^*$ be a prefix-closed language and let $E \subseteq L$ be a specification language. The supremal controllable sublanguage of E with respect to L is

$$\kappa_L(E) := \cup\{K \in \mathcal{C}(L) \mid K \subseteq E\}$$

$\square$

$\kappa_L(E)$ is the union of all controllable sublanguages of L that do not violate the specification E. It holds that $\kappa_{L_1}(E)$ is controllable w.r.t. L_1 if $H = (L_1, L_2)$ is a control system and E is a specification for H. Thus there exists a supervisor S such that $L_1^c = \kappa_{L_1}(E)$. As $\kappa_{L_1}(E)$ constitutes the union of all controllable sublanguages of E w.r.t. L_1, S is called a *maximally permissive* supervisor.

Assuming that specifications E are chosen to be L_2-closed, it is important to investigate if the supremal controllable sublanguage $\kappa_{L_2}(E)$ is also L_2-closed.

[12] An intuitive explanation of this requirement on E is given in section 2.4.

Lemma 2.7 (L_2-closure of Supremal Controllable Sublanguages [ZC94, CL99])
Let $E \in \mathcal{F}_{L_2}$. Then $\kappa_{L_2}(E) \in \mathcal{F}_{L_2}$. □

Lemma 2.7 states that the property of L_2-closure is preserved under the computation of the supremal controllable sublanguage. A proof is given in [ZC94, CL99].

The systems under investigation are control systems with certain properties according to Definition 2.11. A very important property is the regularity of the languages of the control system. The following interesting result, which is provided by [WR87] states that regularity is preserved if both the system behavior and the specification are described by regular languages.

Lemma 2.8 (Supremal Controllable Sublanguage for Regular Languages [RW87b])
Let $L \subseteq \Sigma^*$ be a prefix-closed language and let $E \subseteq L$ be a specification language, where both L and E are regular. Then $\kappa_L(E)$ is regular. □

We use this result to show that the closed loop S/H is again a control system, if H is a control system, the specification language E is regular and the supervisor S implements the supremal controllable sublanguage of E w.r.t L_1.

Corollary 2.2
Let $H = (L_1, L_2)$ be an control system and let $E \in \mathcal{F}_{L_2}$. Then a supervisor S such that $L_1^c = \overline{\kappa_{L_2}(E)}$ and $L_2^c = \kappa_{L_2}(E)$ is maximally permissive and nonblocking (see also [CL99]). If E is regular, then S/H constitutes a control system. □

Proof: Maximal permissiveness holds as the supremal controllable sublanguage $\kappa_{L_2}(E)$ is computed. Nonblocking behavior follows from Theorem 2.4, because $\kappa_{L_2}(E)$ is controllable and also L_2-closed with Lemma 2.7. S/H is a control system because

(i) L_2^c is regular because of Lemma 2.8. Thus $L_1^c = \overline{L}_2$ is also regular.

(ii) $L_1^c = \overline{L}_2$ is prefix-closed.

(iii) $L_2^c \subseteq \overline{L}_2 = L_1^c$.

(iv) $\Sigma = \Sigma_{uc} \dot{\cup} \Sigma_c$ because of the definition of H.

□

The implementation of a nonblocking supervisor for a given specification is illustrated in the subsequent example.

Example 2.10
Let H be the control system in Example 2.8 (recall that $L_1 = \{\varepsilon, \mathtt{a}, \mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$ and $L_2 = \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\}$). The desired system behavior is given as a regular language $E = \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}\}$.

At first the L_2-closure is verified. $\overline{E} \cap L_2 = \{\varepsilon, \mathtt{a}, \mathtt{ab}, \mathtt{aba}, \mathtt{abc}\} \cap \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}, \mathtt{abca}\} = \{\mathtt{ab}, \mathtt{aba}, \mathtt{abc}\} = E$.

For checking controllability, the string abca is investigated. It holds that $\mathtt{abca} \in \overline{E}\Sigma_{\mathrm{uc}}$, and $\mathtt{abca} \in L_1$, that is $\mathtt{abca} \in \overline{E}\Sigma_{\mathrm{uc}} \cap L_1$ but $\mathtt{abca} \notin \overline{E}$. This means $\overline{E}\Sigma_{\mathrm{uc}} \cap L_1 \not\subseteq \overline{E}$, and thus E is not controllable w.r.t. L_1 and Σ_{uc}. Because of this reason, the supremal controllable sublanguage is determined. Controllability fails because the specification requires that the event a has to be disabled after the string abc. This is not possible, as a is uncontrollable. Thus, the occurrence of abc must be prevented by disabling c after the string ab. This is done by the supervisor shown in Example 2.9. The closed loop behavior S/H is also given in this example and it is readily observed that it constitutes a nonblocking control system. □

To sum up, discrete event systems are represented by control systems in this thesis. Specifications for these systems are given as languages and it is possible to compute a supervisor which implements the supremal controllable sublanguage of a specification. Furthermore, an important special case is taken into account. If the specification is a regular language, then the closed loop system is again a control system, i.e. it fulfills all requirements according to Definition 2.11.

2.4 Automata Representation in Supervisory Control

Up to now, discrete event systems have been modeled as control systems, as this type of model is convenient for theoretical considerations. It is more convenient to represent discrete event systems as finite automata in regard to applications of the supervisory control theory. We establish the link between control systems and finite automata, and relate the supervisory control methods elaborated in the previous section to an automata formalism.

The control system in Definition 2.11 can be represented as a finite automaton.

Lemma 2.9 (Automaton from Control System)
Let $H = (L_1, L_2)$ be a control system. Then there exists a minimal deterministic finite automaton $G = (\Sigma, X, \delta, x_0, X_{\mathrm{m}})$ which generates L_1 and which recognizes L_2, i.e. $L(G) = L_1$, $L_{\mathrm{m}}(G) = L_2$ and $|X| = ||L_1||$. □

Proof: As L_1 and L_2 are regular, there exist $G_{\text{gen}} = (\Sigma, X_{\text{gen}}, \delta_{\text{gen}}, x_{0,\text{gen}}, X_{\text{m,gen}})$ and $G_{\text{rec}} = (\Sigma, X_{\text{rec}}, \delta_{\text{rec}}, x_{0,\text{rec}}, X_{\text{m,rec}})$ such that $L_{\text{m}}(G_{\text{gen}}) = L(G_{\text{gen}}) = L_1$ and $L_{\text{m}}(G_{\text{rec}}) = L_2$ because of Corollary 2.1. G_{rec} is extended to a new automaton $\tilde{G}_{\text{rec}} = (\Sigma, \tilde{X}_{\text{rec}}, \tilde{\delta}_{\text{rec}}, \tilde{x}_{0,\text{rec}}, \tilde{X}_{\text{m,rec}})$ with a new state $x_{\text{d}} \notin \tilde{X}_{\text{m}}$, and the new transition function is defined for $x \in X$ and $\sigma \in \Sigma$ as

$$\tilde{\delta}_{\text{rec}}(x,\sigma) := \begin{cases} \delta_{\text{rec}}(x,\sigma) & \text{if } \delta_{\text{rec}}(x,\sigma)! \\ x_{\text{d}} & \text{otherwise} \end{cases}$$

$\tilde{G}_{\text{rec}}$ is constructed such that $L(\tilde{G}_{\text{rec}}) = \Sigma^*$ and $L_{\text{m}}(\tilde{G}_{\text{rec}}) = L_2$. Now we compute $\tilde{G} := G_{\text{gen}} || \tilde{G}_{\text{rec}}$. As $L_{\text{m}}(\tilde{G}_{\text{rec}}) = L_2 \subseteq L_{\text{m}}(G_{\text{gen}}) = L_1$, it holds that $L_{\text{m}}(\tilde{G}) = L_1 \cap L_2 = L_2$. With $L(G_{\text{gen}}) = L_1 \subseteq L(\tilde{G}_{\text{rec}}) = \Sigma^*$ it is true that $L(\tilde{G}) = L_1 \cap \Sigma^* = L_1$. Applying a state minimization algorithm ([HU79]) to $\tilde{G}$, the resulting automaton G has a minimal number of states and also $L(G) = L(\tilde{G}) = L_1$ and $L_{\text{m}}(G) = L_{\text{m}}(\tilde{G}) = L_2$. Because of Theorem 2.2, this automaton is unique and as G generates L_1, it holds that $|X| = ||L_1||$. □

This means that for any control system, a minimal automaton generating and marking the languages L_1 and L_2, respectively, can be found. In return, given a finite automaton G, $H_G = (L(G), L_{\text{m}}(G))$ is written for the corresponding control system.

Lemma 2.10 (Control System from Automaton)
Let $G = (\Sigma, X, \delta, x_0, X_{\text{m}})$ be a finite automaton with a partition $\Sigma = \Sigma_{\text{c}} \dot{\cup} \Sigma_{\text{uc}}$ of the alphabet Σ into uncontrollable events Σ_{uc} and controllable events Σ_{c}. Then $H_G := (L(G), L_{\text{m}}(G))$ is a control system. □

Proof: It has to be shown that all conditions in Definition 2.11 are fulfilled.

(i) $L(G)$ and $L_{\text{m}}(G)$ are regular because of Theorem 2.3.

(ii) $L(G)$ is prefix-closed because of Definition 2.8

(iii) $L_{\text{m}}(G) \subseteq L(G)$ because of Definition 2.8.

(iv) $\Sigma = \Sigma_{\text{c}} \dot{\cup} \Sigma_{\text{uc}}$ is given.

Thus H_G is a control system. □

The equivalence of control systems and finite automata is illustrated by the next example.

Example 2.11
Consider the control system $H = (L_1, L_2)$ from Example 2.8 with the languages $L_1 = \{\varepsilon$, `a`, `ab`, `aba`, `abc`, `abca`$\}$ and $L_2 = \{$`ab`, `aba`, `abc`, `abca`$\}$. The automaton G in Example 2.5 is the corresponding automata representation. □

Considering that a control system can be represented as a finite automaton, it is clear that the supervisor computation and implementation can also be realized in the finite automata framework. This is useful, as the definitions in section 2.3.1 involve possibly infinite sets (languages), whereas automata provide a finite representations of regular languages.

A regular language specification $E \subseteq L$ can always be represented by a finite automaton because of Theorem 2.3. There is also a finite automata implementation R of a supervisor S, as shown in the next lemma.[13]

Lemma 2.11 (Automata Representation of a Supervisor [Won04])
Let $H = (L_1, L_2)$ be a control system. Also let $S : L_1 \to \Gamma$ be a supervisor such that $L_1^{\mathrm{c}} = \overline{E}$ and $L_2^{\mathrm{c}} = E$ for a regular specification language E. The automaton R recognizing E and generating $\overline{E}$, implements the supervisor S, i.e. $L_{\mathrm{m}}(R) \cap L_2 = E$ and $L(R) \cap L_1 = \overline{E}$. □

The event sets of the automaton realization G of H and R are both Σ. This observation together with Lemma 2.11 and Lemma 2.5 can be used to find out how the supervisor automaton R has to be interconnected with G to yield the desired behavior.

Corollary 2.3 (Supervisor Implementation [Won04])
Let G be an automata implementation of a control system H, and let E and R be defined as above. Interconnecting G and R with the synchronous composition yields

(i) $L(G||R) = \overline{E}$,

(ii) $L_{\mathrm{m}}(G||R) = E$.

□

Proof: For the synchronous composition of G and R, observe that $\Sigma_1 = \Sigma_2 = \Sigma$ and $p_1 = p_2 =: p : \Sigma^* \to \Sigma^*$ is the natural projection. With Lemma 2.5, Definition 2.5 and Lemma 2.11, it holds that $L(G||R) = L(G)||L(R) = (p)^{-1}(L(G)) \cap (p)^{-1}(L(R)) = L(G) \cap L(R) = \overline{E}$ and $L_{\mathrm{m}}(G) = L_{\mathrm{m}}(G)||L_{\mathrm{m}}(R) = (p)^{-1}(L_{\mathrm{m}}(G)) \cap (p)^{-1}(L_{\mathrm{m}}(R)) = L_{\mathrm{m}}(G) \cap L_{\mathrm{m}}(R) = E$. □

This means that considering the feedback loop in Figure 2.6, the supervisor S (represented by R) follows the event sequences generated by G and allows all events which can occur in the respective state of R. This can be written as $S(s) = \Lambda(\delta_R(x_{R,0}, s)) \cup \Sigma_{\mathrm{uc}}$.[14]

Example 2.12
As an example, the supervisor computed in Example 2.10 is implemented. Figure 2.8 shows the resulting finite automaton. It is readily observed that the controllable event c is disabled in state 2.[15] □

[13]Note that this supervisor implementation is not unique i.e. there are other automata implementing the same supervisor.

[14]Recall that uncontrollable events are never disabled.

[15]Note that the event set of R is $\Sigma_R = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, and thus $\mathtt{c}$ is a shared event of G and R.

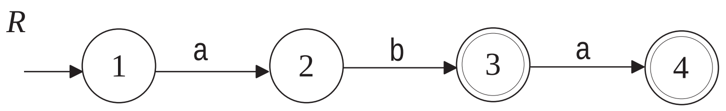


Figure 2.8: Automata implementation R of the supervisor S

In addition to realizing a supervisor for a controllable language E and a control system H, it is also possible to compute the supremal controllable sublanguage of a specification language based on automata representations. Algorithms for this computation are given in [WR87, RK91]. They initiate with an automata representation G and R of a control system H and the specification $E \in \mathcal{F}_{L_2}$, respectively. Then, the supremal controllable sublanguage $\kappa_{L_2}(E)$ is computed by eliminating states in the synchronous composition $G||R$ of G and R. For prefix-closed specifications $E = \overline{E}$, the algorithm is of complexity $\mathcal{O}(mn)$ ([RK91]), where m and n are the number of states of R and G, respectively. In case that E is not prefix-closed, the complexity is $\mathcal{O}(m^2n^2)$ ([WR87]).

Summarizing, in this section a finite automata representation for both a control system and a supervisor has been found. Hence, it is possible to implement the theoretical results by using a finite automata representation of control systems and supervisors. There are further control frameworks for discrete event systems which shall not be considered in this work [Pu00, dCCK02, Ma04, ZW01].

Chapter 3

Nonblocking Hierarchical Control

In the previous chapter, it was pointed out that the complexity for supervisor synthesis is polynomial in time ($\mathcal{O}(n^2m^2)$ if an automata implementation of a control system with n states and an automata implementation of a specification with m states is given). However, this does not imply that it is always possible to compute a supervisor. This is due to the fact that the number of states n of a discrete event systems which is composed of several components grows exponentially with its number of components. This state explosion is the reason why supervisory control for large scale systems fails.

One approach dealing with this problem is *hierarchical supervisory control*. An abstracted model (with fewer states) is computed, instead of synthesizing a supervisor for the real system model. For this *high-level model*, supervisor synthesis is feasible and the resulting supervisor has to be translated to the *low level*. Note that although the hierarchical approach facilitates the supervisor computation on the high level, it is still necessary to compute the overall low-level model.[1]

Within the framework elaborated in Chapter 2, we develop a control theory for hierarchical discrete event systems. Figure 3.1 illustrates the architecture underlying this approach.

On the low level, there is a control system H which describes the detailed behavior of the given system. The supervisor S^{lo} applies its low-level control action to H. Together, H and S^{lo} form a low-level closed-loop system, indicated by $\mathrm{Con}^{\mathrm{lo}}$ (control action from the supervisor) and Inf^{lo} (feedback information from the control system). Similarly, the high-level closed-loop consists of an abstracted plant model H^{hi} and the supervisor S^{hi}. It is important to note that the standard supervisory control framework can also be used on the high-level. The two levels are interconnected via $\mathrm{Com}^{\mathrm{hilo}}$ and $\mathrm{Inf}^{\mathrm{lohi}}$. As the control action of the high-level supervisor S^{hi} on G^{hi} is just virtual, it must be translated to the control action of a low-level supervisor S^{lo}, which directly controls the low-level system H. This is done by $\mathrm{Com}^{\mathrm{hilo}}$. The channel $\mathrm{Inf}^{\mathrm{lohi}}$ provides the necessary information for the progress of the high-level system H^{hi}.

[1] This issue is addressed in the next chapter.

From the perspective of the high-level supervisor, the forward path sequence $\mathrm{Com}^{\mathrm{hilo}}$, $\mathrm{Con}^{\mathrm{lo}}$ is usually designated “command and control”, while the feedback path sequence $\mathrm{Inf}^{\mathrm{lohi}}$, $\mathrm{Inf}^{\mathrm{hi}}$ is identified with “report and advise”.

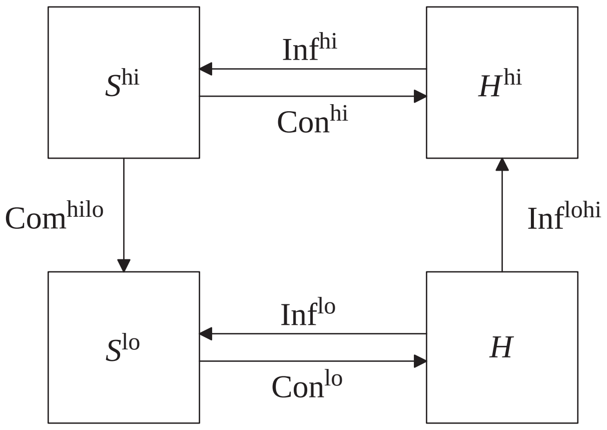

S^{hi}
$\mathrm{Inf}^{\mathrm{hi}}$
$\mathrm{Con}^{\mathrm{hi}}$
H^{hi}
$\mathrm{Com}^{\mathrm{hilo}}$
$\mathrm{Inf}^{\mathrm{lohi}}$
S^{lo}
$\mathrm{Inf}^{\mathrm{lo}}$
$\mathrm{Con}^{\mathrm{lo}}$
H

Figure 3.1: Hierarchical control architecture

The hierarchical method is related to the work in [Pu00, dCCK02, WW96, Won04]. All these approaches use the above hierarchical architecture, and are based on a low-level automata representation of a control system. They introduce a reporter map for abstracting the low-level behavior and represent the high-level model as an automaton. High-level events are generated if particular states which are labeled with tokens are reached. [Pu00], [Won04] and [WW96] investigate the "observer" and "weak observer" property for relating the high-level behavior to the low-level behavior and for implementing high-level supervisors in the low level. Also, controllability of high-level events in each high-level state is determined by investigating the low-level behavior corresponding to the respective high-level state. In [Pu00, Won04, WW96], high-level states are marked, if all low-level states, which are reached, when the marked high-level state is entered, are marked. Different from that, a new high-level control structure accounting for the controllability and marking properties of local behavior is employed in [dCCK02]. While the method in [Pu00] is worked out for multi-level hierarchies, the approach in [dCCK02] is formulated for a two-level hierarchy.

In our work, the computation of the abstracted system H^{hi} is done by applying the natural projection of the control system H on a predefined set of high-level events. This projection is the operation carried out by the information channel $\mathrm{Inf}^{\mathrm{lohi}}$. As in [Won04], the observer property is needed, but a less restrictive high-level marking condition is required. Furthermore, as high-level events are elements of the low-level event set, the controllability properties are carried over from the low level[2]. Consequently, the Ramadge/Wonham framework is used on both the high and the low level, and the method is readily extended to a multi-level hierarchy.

The chapter is organized as follows: In Section 3.1, basic definitions for the hierarchical control framework are given. The notion of *hierarchical consistency* is explained in Section 3.2. Our method for *hierarchically consistent* and *nonblocking control* of hierarchical systems is established in Section 3.3. These properties are guaranteed by making use of different conditions on the structure of hierarchical systems. The chapter concludes with an algorithmic implementation of the concepts described before in the automata representation.

3.1 Basic Definitions

The information channel $\mathrm{Inf}^{\mathrm{lohi}}$ is realized by a natural projection. Formally, both languages of a control system are projected to a given event set. The following lemma states that the resulting languages again form a control system.

Lemma 3.1 (Projected Control System)
Let $H = (L_1, L_2)$ be a control system and let $p_0 : \Sigma^* \to \Sigma_0^*$ be the natural projection where $\Sigma_0 = \Sigma_{0,\mathrm{c}} \dot{\cup} \Sigma_{0,\mathrm{uc}} \subseteq \Sigma$ and $\Sigma_{0,\mathrm{c}} = p_0(\Sigma_\mathrm{c})$ and $\Sigma_{0,\mathrm{uc}} = p_0(\Sigma_\mathrm{uc})$. Further on, the projection is generalized to control systems by defining $p_0\big((L_1, L_2)\big) := \big(p_0(L_1), p_0(L_2)\big)$. Then $H_0 := p_0(H)$ is again a control system. The tuple (H, p_0, H_0) is denoted a *projected control system* (PCS). □

Proof: It has to be shown that the conditions in Definition 2.11 are fulfilled.

(i) Because of Lemma 2.3, both $p_0(L_1)$ and $p_0(L_2)$ are regular.

(ii) With Lemma 2.1, $p_0(L_1)$ is prefix-closed.

(iii) $p_0(L_2) \subseteq p_0(L_1)$ directly follows from monotony of the projection p_0.

(iv) $\Sigma_0 = \Sigma_{0,\mathrm{c}} \dot{\cup} \Sigma_{0,\mathrm{uc}}$ by definition.

Consequently, H_0 is a control system. □

[2]The latter condition may sound very restrictive, but it is tailored for the hierarchical and decentralized approach presented in the next chapter.

Integrating the projected control system, the *hierarchical closed-loop system* is the basis for further considerations. It establishes the abstraction of a low-level control system H via the natural projection p^{hi} on the high-level events Σ^{hi}, yielding the high-level control system H^{hi}. It also includes the high-level supervisor S^{hi} and the low-level supervisor S^{lo} and poses a condition on the relation between S^{hi} and S^{lo}.

Definition 3.1 (Hierarchical Closed-Loop System)
Referring to Lemma 3.1, a *hierarchical closed-loop system (HCLS)* $Q = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ consists of a projected control system $P = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ equipped with a *high-level supervisor* S^{hi} and a *low-level supervisor* S^{lo}, where

(i) $S^{\mathrm{hi}} : L_1^{\mathrm{hi}} \to \Gamma^{\mathrm{hi}}$ with the high-level control patterns $\Gamma^{\mathrm{hi}} := \{\gamma | \Sigma_{\mathrm{uc}}^{\mathrm{hi}} \subseteq \gamma \subseteq \Sigma^{\mathrm{hi}}\}$.

(ii) $S^{\mathrm{lo}} : L_1 \to \Gamma$.

S^{lo} is called *valid* w.r.t. S^{hi} if $p^{\mathrm{hi}}(S^{\mathrm{lo}}/H) \subseteq S^{\mathrm{hi}}/H^{\mathrm{hi}}$. Q is *finite* if the languages L_1^{c}, L_2^{c}, $L_1^{\mathrm{hi,c}}$, $L_2^{\mathrm{hi,c}}$ are regular with $S^{\mathrm{lo}}/H =: (L_1^{\mathrm{c}}, L_2^{\mathrm{c}})$ and $S^{\mathrm{hi}}/H^{\mathrm{hi}} =: (L_1^{\mathrm{hi,c}}, L_2^{\mathrm{hi,c}})$.[3] □

In the above definition, the choice of the command channel $\mathrm{Com}^{\mathrm{hilo}}$ is still arbitrary unless validity is required. If this is the case, the low-level supervisor must guarantee that the abstracted low-level closed-loop behavior stays inside the high-level closed-loop behavior. This is a desirable property as the low-level supervisor should be able to realize the control actions requested by the high-level supervisor.

Starting from these observations, the crucial point is to find a valid low-level supervisor and to achieve nonblocking behavior of the hierarchical closed-loop system. This requirement is formally stated in the following definition.

Definition 3.2 (Hierarchical Control Problem)
Given a projected control system $P = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ and a nonblocking high-level supervisor S^{hi}, compute a valid low-level supervisor S^{lo} such that the HCLS $Q = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ yields a nonblocking low-level closed-loop system S^{lo}/H. □

If such low-level implementation of the a high-level supervisor has been found, the question remains, if the corresponding abstracted behavior $p^{\mathrm{hi}}(L_1^{\mathrm{c}}, L_2^{\mathrm{c}})$ is also nonblocking. The positive answer to this question is given in Lemma 3.2.

Lemma 3.2 (Nonblocking HCLS)
Let $Q = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ be a HCLS with a nonblocking valid low-level supervisor S^{lo}. Then the abstraction $p^{\mathrm{hi}}(S^{\mathrm{lo}}/H)$ is nonblocking. □

[3]This definition of $S^{\mathrm{hi}}/H^{\mathrm{hi}}$ and S^{lo}/H will be used throughout the thesis.

Proof: It has to be shown that for any string $s^{\mathrm{hi}} \in p^{\mathrm{hi}}(L_1^{\mathrm{c}})$, there exists $t \in (\Sigma^{\mathrm{hi}})^*$ s.t. $s^{\mathrm{hi}}t \in p^{\mathrm{hi}}(L_2^{\mathrm{c}})$. Let $s^{\mathrm{hi}} \in p^{\mathrm{hi}}(L_1^{\mathrm{c}})$. Then, there is a string $s \in L_1^{\mathrm{c}}$ s.t. $p^{\mathrm{hi}}(s) = s^{\mathrm{hi}}$. As S^{lo} is nonblocking, there exists a $u \in \Sigma^*$ s.t. $su \in L_2^{\mathrm{c}}$. But then $p^{\mathrm{hi}}(su) \in p^{\mathrm{hi}}(L_2^{\mathrm{c}})$ because of Lemma 3.1. □

The above notions are explained in the next example.

Example 3.1
Consider the projected control system $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ in Figure 3.2 with the control system H, the natural projection $p^{\mathrm{hi}} : \Sigma^* \rightarrow (\Sigma^{\mathrm{hi}})^*$ and the abstracted control system H^{hi}. The low-level languages are $L_1 = \overline{\alpha\mathsf{a}(\mathsf{ba}(\mathsf{dc})^*(\gamma + \mathsf{d}\beta) + \mathsf{d}\gamma)}$ and $L_2 = \varepsilon + \alpha\mathsf{aba}(\mathsf{dc})^*(\varepsilon + \gamma + \mathsf{d}\beta) + \alpha\mathsf{ad}\gamma$ with the low-level alphabet $\Sigma = \Sigma_{\mathrm{uc}} \dot\cup \Sigma_{\mathrm{c}} = \{\alpha, \mathsf{a}, \mathsf{d}\} \dot\cup \{\beta, \gamma, \mathsf{b}, \mathsf{c}\}$. The high-level alphabet is chosen as $\Sigma^{\mathrm{hi}} = \Sigma_{\mathrm{uc}}^{\mathrm{hi}} \dot\cup \Sigma_{\mathrm{c}}^{\mathrm{hi}} = \{\alpha\} \dot\cup \{\beta, \gamma\}$, and the high-level languages are $L_1^{\mathrm{hi}} = L_2^{\mathrm{hi}} = \overline{\alpha(\beta + \gamma)}$.

The supervisor S^{hi} with

$$S^{\mathrm{hi}}(s^{\mathrm{hi}}) := \begin{cases} \{\alpha, \beta\} & \text{for } s^{\mathrm{hi}} = \alpha \\ \{\alpha, \beta, \gamma\} & \text{otherwise} \end{cases} \tag{3.1}$$

is chosen for $s^{\mathrm{hi}} \in L_1^{\mathrm{hi}}$ for high-level control, i.e. the event γ is disabled after the high-level string α. A valid low-level supervisor for S^{hi} is for example

$$S^{\mathrm{lo}}(s) := \begin{cases} \{\alpha, \beta, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\} & \text{if } s = \alpha\mathsf{ad} \\ \{\alpha, \beta, \gamma, \mathsf{a}, \mathsf{c}, \mathsf{d}\} & \text{if } s = \alpha\mathsf{a} \\ \Sigma & \text{otherwise.} \end{cases} \tag{3.2}$$

It disables the event γ and b for the strings $\alpha\mathsf{ad}$ and $\alpha\mathsf{a}$, respectively. For the other low-level strings all events are enabled.

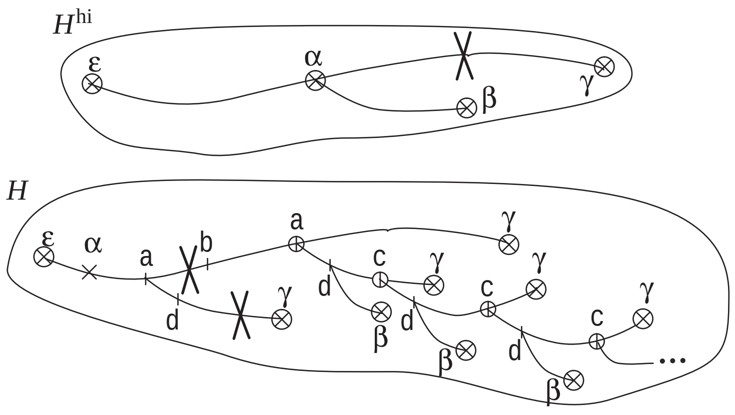


Figure 3.2: Hierarchical closed-loop system

The resulting closed-loop projected system is $(S^{\mathrm{lo}}/H, p^{\mathrm{hi}}, S^{\mathrm{hi}}/H^{\mathrm{hi}})$ with the low-level closed-loop languages $L_1^{\mathrm{c}} = \overline{\alpha \mathrm{ad}}$ and $L_2^{\mathrm{c}} = \varepsilon$. The closed-loop languages in the high level are $L_1^{\mathrm{hi,c}} = L_2^{\mathrm{hi,c}} = \overline{\alpha\beta}$. Note that $p^{\mathrm{hi}}(L_1^{\mathrm{c}}) = \overline{\alpha} \subset L_1^{\mathrm{hi,c}}$ and $p^{\mathrm{hi}}(L_2^{\mathrm{c}}) = \varepsilon \subset L_2^{\mathrm{hi,c}}$, which proves validity of S^{lo} as $p^{\mathrm{hi}}(S^{\mathrm{lo}}/H) \subseteq S^{\mathrm{hi}}/H^{\mathrm{hi}}$. □

The example shows an instance of a valid low-level supervisor S^{lo} for the high-level supervisor S^{hi}, i.e. the high-level abstraction of the low-level controlled behavior is included in the desired high-level behavior $S^{\mathrm{hi}}/H^{\mathrm{hi}}$. In the subsequent section, the preferable case of *hierarchical consistency* is addressed, i.e. the abstraction of the low-level closed-loop behavior equals the desired high-level behavior. It can be proven that the hierarchical closed-loop system in Definition 3.1 is hierarchically consistent for a particular *consistent implementation* of the low-level supervisor.

3.2 Hierarchical Consistency

Before elaborating the main results of this section, some basic representations of local behavior of projected control systems are introduced. Similar concepts are used in [Pu00, dCCK02, Won04]. The set of *entry strings*[4] contains all low-level strings which are just projected to a given high-level string.

Definition 3.3 (Entry Strings [dCCK02])
Let $P = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a projected system and assume $s^{\mathrm{hi}} \in L_1^{\mathrm{hi}}$. The set of entry strings of s^{hi} is

$$L_{\mathrm{en},s^{\mathrm{hi}}} := \{s \in L_1 | p^{\mathrm{hi}}(s) = s^{\mathrm{hi}} \wedge \nexists s' < s \text{ s.t. } p^{\mathrm{hi}}(s') = s^{\mathrm{hi}}\} \subseteq \Sigma^*$$

□

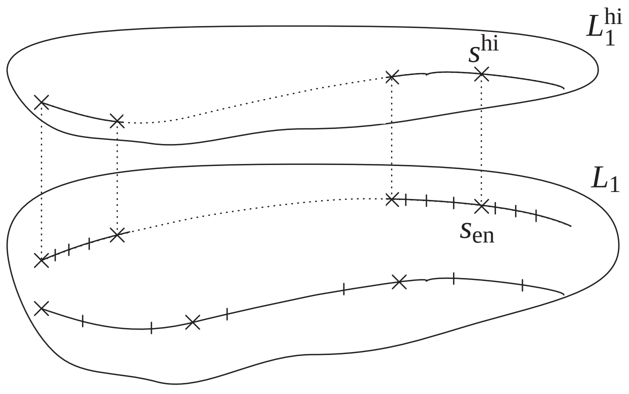


Figure 3.3: Illustration of entry strings.

[4]Entry strings also called vocal strings in the literature.

The local behavior after strings $s \in L_1$ is described next. It represents the behavior which can occur locally after the observation of a high-level event. For any string $s \in L_1$, the continuation of s with strings $u \in (\Sigma - \Sigma^{\mathrm{hi}})^*(\Sigma \cup \varepsilon)$ in L_1 is a prefix-closed language. A second (not necessarily prefix-closed) language contains local continuations of s in L_2 and also local continuations in L_1 terminating with a high-level event.

Definition 3.4 (Local Languages [SMP05])
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a projected control system, and let $s \in L_1$ for $s^{\mathrm{hi}} := p^{\mathrm{hi}}(s) \in L_1^{\mathrm{hi}}$. The local prefix-closed language $L_{s,1}$ is

$$L_{s,1} := \{u \in (\Sigma - \Sigma^{\mathrm{hi}})^* | su \in L_1\} \subseteq \Sigma^*$$

and the local language $L_{s,2}$ is

$$L_{s,2} := \{u \in (\Sigma - \Sigma^{\mathrm{hi}})^* | su \in L_2\} \subseteq \Sigma^*.$$

$\square$

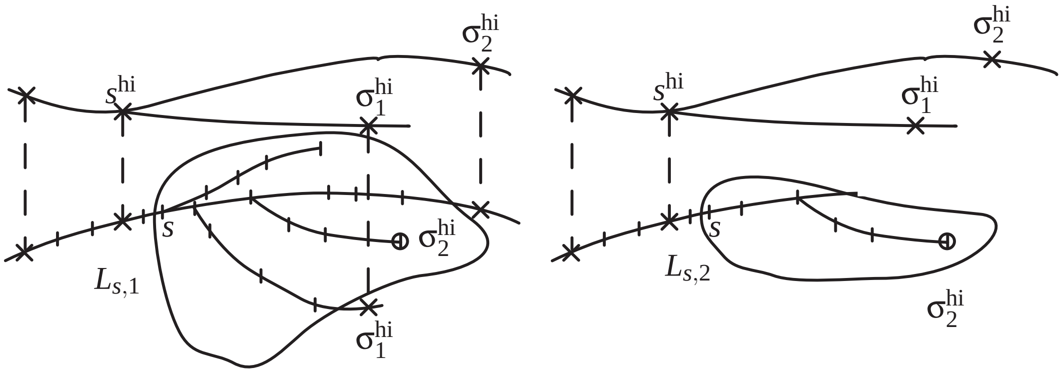


Figure 3.4: Illustration of the local languages.

$L_{s,1}$ can be thought of as the local behavior of H after the string s until a new high-level event occurs, i.e. until the progress of the system can be observed from the high-level. The language $L_{s,2}$ consists of all continuations of s in $L_{s,1}$ which are either marked strings or which can just be observed in the high-level.[5]

Combining the two languages defined above, it turns out that the tuple $(L_{s,1}, L_{s,2})$ is a control system. As pointed out above, it represents the local behavior which is possible after the occurrence of the string s. We show this result in the following lemma.

[5] $L_{s,2} \subseteq L_{s,1}$ is shown below.

Lemma 3.3 (Local Control System)
Let $P = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a projected control system, and let $s \in L_1$ for $s^{\mathrm{hi}} := p^{\mathrm{hi}}(s) \in L_1^{\mathrm{hi}}$. Also let $L_{s,1}$ and $L_{s,2}$ be defined as in Definition 3.4. Then $H_s := (L_{s,1}, L_{s,2})$ forms a control system. □

Proof: The properties required in Definition 2.11 have to be verified.

(i) $s(\Sigma - \Sigma^{\mathrm{hi}})^*$ is a regular set because of Definition 2.6 and 3.4. Also L_1 and L_2 are chosen to be regular. Considering Lemma 2.2, the intersections $s(\Sigma - \Sigma^{\mathrm{hi}})^* \cap L_1$ and $s(\Sigma - \Sigma^{\mathrm{hi}})^* \cap L_2$ are regular. By Definition 3.4, it holds that $s(\Sigma - \Sigma^{\mathrm{hi}})^* \cap L_1 = sL_{s,1}$ and $s(\Sigma - \Sigma^{\mathrm{hi}})^* \cap L_2 = sL_{s,2}$. Again, because of Lemma 2.2, $sL_{s,1}$ and $sL_{s,2}$ are regular sets and with Definition 2.6, $L_{s,1}$ and $L_{s,2}$ are regular.

(ii) $\bar{s}(\Sigma - \Sigma^{\mathrm{hi}})^*$ and L_1 are prefix-closed. Hence $\bar{s}(\Sigma - \Sigma^{\mathrm{hi}})^* \cap L_1 = \bar{s}L_{s,1}$ is prefix-closed. As a consequence, $L_{s,1}$ is prefix-closed.

(iii) Showing $L_{s,2} \subseteq L_{s,s^{\mathrm{hi}}}$ is equivalent to showing $sL_{s,2} \subseteq sL_{s,1}$. Let $s' \in sL_{s,2}$. Then $s' \in s(\Sigma - \Sigma^{\mathrm{hi}})^* \cap L_2$. Hence, as $L_2 \subseteq L_1$, it holds that $s' \in s(\Sigma - \Sigma^{\mathrm{hi}})^* \cap L_2 \subseteq s(\Sigma - \Sigma^{\mathrm{hi}})^* \cap L_1 = sL_{s,1}$.

(iv) $\Sigma = \Sigma_{\mathrm{uc}} \dot{\cup} \Sigma_{\mathrm{c}}$ by definition.

□

In the hierarchical control framework, the high-level supervisor is designed for a high-level specification, and a corresponding low-level supervisor is computed. A change in the control strategy of the low-level supervisor is only meaningful, if the control action of the high-level supervisor changes. As entry strings are just projected to high-level strings, we synthesize one local supervisor for each local behavior which can happen after the respective entry string. We denote these local supervisors *consistent implementations*.

Definition 3.5 (Consistent Implementation [SMP05])
Given a projected control system $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ and a supervisor S^{hi}, the consistent implementation S^{lo} is defined as follows. For $s \in L_1$, let $s^{\mathrm{hi}} := p^{\mathrm{hi}}(s)$ and $s_{\mathrm{en}} \in L_{\mathrm{en},s^{\mathrm{hi}}}$, $u \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ s.t. $s = s_{\mathrm{en}}u$. Then

$$S^{\mathrm{lo}}(s) := \begin{cases} S^{\mathrm{hi}}(s^{\mathrm{hi}}) \cup (\Sigma - \Sigma^{\mathrm{hi}}) & \text{if } \Sigma^{\mathrm{hi}}(s^{\mathrm{hi}}) \cap S^{\mathrm{hi}}(s^{\mathrm{hi}}) \neq \emptyset, \\ \{\sigma \in (\Sigma - \Sigma^{\mathrm{hi}}) | u\sigma \in \kappa_{L_{s_{\mathrm{en}},1}}(L_{s_{\mathrm{en}},2})\} \cup \Sigma_{\mathrm{uc}} & \text{else.} \end{cases}$$

$(H, p^{\mathrm{hi}}, H^{\mathrm{hi}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ is called a HCLS with a consistent implementation. □

The *consistent implementation* needs the computation of a supremal controllable sublanguage if there are no successor events allowed by the high-level supervisor after a high-level string. This guarantees that no blocking can occur in the low-level behavior if there are no successor events after the corresponding high-level string. Otherwise, the consistent implementation enables all low-level events and just disables the occurrence of high-level events if necessary.

Taking into account the properties of a hierarchical closed-loop system, it has to be shown that the supervisor implementation in Definition 3.5 is admissible, i.e. it agrees with Definition 2.14.

Lemma 3.4 (Admissible Supervisor Implementation)
Let $(H, p^{\text{hi}}, H^{\text{hi}})$ be a projected control system, and S^{hi} be an admissible supervisor with a consistent implementation S^{lo}. Then S^{lo} is admissible, i.e. $(H, p^{\text{hi}}, H^{\text{hi}}, S^{\text{hi}}, S^{\text{lo}})$ is a HCLS. $\square$

Proof: It must be proven that $\Sigma_{\text{uc}} \subseteq S^{\text{lo}}(s)$. Let $s \in L_1$ and $s^{\text{hi}} := p^{\text{hi}}(s)$. Then either $S^{\text{hi}}(s^{\text{hi}}) \cap \Sigma^{\text{hi}}(s^{\text{hi}}) \neq \emptyset$ or $S^{\text{hi}}(s^{\text{hi}}) \cap \Sigma^{\text{hi}}(s^{\text{hi}}) = \emptyset$. In the first case, $\Sigma^{\text{hi}}_{\text{uc}} \subseteq S^{\text{hi}}(s^{\text{hi}})$ and $(\Sigma_{\text{uc}} - \Sigma^{\text{hi}}_{\text{uc}}) \subseteq (\Sigma - \Sigma^{\text{hi}})$. Thus, $\Sigma_{\text{uc}} \subseteq S^{\text{hi}}(s^{\text{hi}}) \cup (\Sigma - \Sigma^{\text{hi}}) = S^{\text{lo}}(s)$. In the second case $\Sigma_{\text{uc}} \subseteq S^{\text{lo}}(s)$ by definition.

$\square$

Looking at Definition 3.1, a low-level supervisor is considered to be *valid* if the projection of the low-level closed-loop behavior stays inside the high-level closed-loop behavior, i.e. the system restriction imposed by the virtual high-level supervisor can be implemented by low-level control.

Yet, this requirement does not guarantee that the controlled low-level behavior is nonempty. A more restrictive condition, *hierarchical consistency* ensures nonempty low-level closed-loop behavior. It states that the low-level control is such that the behavior of the abstracted low-level supervised system equals the behavior of the high-level closed-loop system.

Definition 3.6 (Hierarchical Consistency [ZW90])
Let $Q = (H, p^{\text{hi}}, H^{\text{hi}}, S^{\text{hi}}, S^{\text{lo}})$ be a hierarchical closed-loop system. Q is hierarchically consistent if $p^{\text{hi}}(L_1^{\text{c}}) = L_1^{\text{hi,c}}$. $\square$

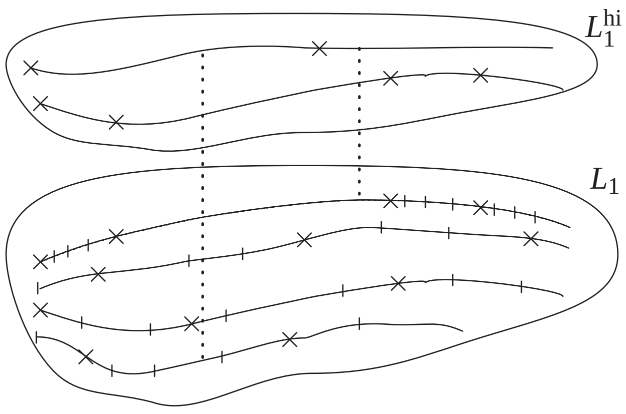


Figure 3.5: Illustration of hierarchical consistency.

The consistent implementation in Definition 3.5 only disables low-level events after a low-level string, if there is no feasible high-level event after the corresponding high-level string. Because of this reason, using a consistent implementation for the low-level supervisor is sufficient for hierarchical consistency of a hierarchical closed-loop system.

Proposition 3.1 (Hierarchical Consistency)
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a projected control system, and let S^{hi} be a high-level supervisor. If S^{lo} is the consistent implementation of S^{hi}, then the HCLS $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ is *hierarchically consistent.* $\square$

Before proving this result, we establish the following technical lemmas.

In Lemma 3.5, we assume that a low-level string in the low-level closed-loop behavior has a local extension to some high-level event in the uncontrolled system. Then it has the same extension in the closed-loop system if the high-level event is enabled by the high-level supervisor after the corresponding high-level string.

Lemma 3.5
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ be a hierarchical closed-loop system with a consistent implementation. Assume that $s^{\mathrm{hi}} \in L_1^{\mathrm{hi,c}}$ and $s \in L_1^{\mathrm{c}}$ with $p^{\mathrm{hi}}(s) = s^{\mathrm{hi}}$. If $su\sigma \in L_1$ for $u \in (\Sigma - \Sigma^{\mathrm{hi}})^*$, $\sigma \in \Sigma^{\mathrm{hi}}$ and $s^{\mathrm{hi}}\sigma \in L_1^{\mathrm{hi,c}}$, then $su\sigma \in L_1^{\mathrm{c}}$. $\square$

Proof: $su \in L_1^{\mathrm{c}}$ is proven by induction. We write $u = \sigma_0\sigma_1 \dots \sigma_m$ with $\sigma_0 = \varepsilon$ and $\sigma_i \in (\Sigma - \Sigma^{\mathrm{hi}})$ for $i = 1, \dots, m$. Then $s\sigma_0 = s \in L_1^{\mathrm{c}}$. Now assume that $s\sigma_0 \dots \sigma_{i-1} \in L_1^{\mathrm{c}}$ for $i \in \{1, \dots, m\}$. From $s\sigma_0 \dots \sigma_{i-1}\sigma_i \in L_1$ and $\sigma_i \in S^{\mathrm{lo}}(s\sigma_0 \dots \sigma_{i-1}) = S^{\mathrm{hi}}(s^{\mathrm{hi}}) \cup (\Sigma - \Sigma^{\mathrm{hi}})$, observe that $s\sigma_0 \dots \sigma_{i-1}\sigma_i \in L_1^{\mathrm{c}}$. As this is true for all $i = 1, \dots, m$, it holds that $su \in L_1^{\mathrm{c}}$. But then, also $su\sigma \in L_1^{\mathrm{c}}$ as $su\sigma \in L_1$ and $\sigma \in S^{\mathrm{lo}}(su) = S^{\mathrm{hi}}(s^{\mathrm{hi}}) \cup (\Sigma - \Sigma^{\mathrm{hi}})$. $\square$

The next lemma shows that any entry string corresponding to a high-level string s^{hi} in the closed-loop behavior $S^{\mathrm{hi}}/H^{\mathrm{hi}}$ is an element of the low-level closed-loop behavior S^{lo}/H.

Lemma 3.6
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ be a hierarchical closed-loop system with a consistent implementation. If $s^{\mathrm{hi}} \in L_1^{\mathrm{hi,c}}$ and $s_{\mathrm{en}} \in L_{\mathrm{en},s^{\mathrm{hi}}}$, then $s_{\mathrm{en}} \in L_1^{\mathrm{c}}$. $\square$

Proof: s_{en} and s^{hi} can be written as $s_{\mathrm{en}} = u_0\sigma_0u_1\sigma_1 \dots u_m\sigma_m$ and $s^{\mathrm{hi}} = \sigma_0\sigma_1 \dots \sigma_m$, respectively, where $u_i \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ and $\sigma_i \in \Sigma^{\mathrm{hi}}$ for $i = 1, \dots, m$ and $u_0 = \sigma_0 = \varepsilon$. It is shown that $s_{\mathrm{en}} \in L_1^{\mathrm{c}}$ by induction. It is readily observed that $u_0\sigma_0 = \varepsilon \in L_1^{\mathrm{c}}$ and $\sigma_0 = \varepsilon \in L_1^{\mathrm{hi,c}}$. Now let $u_0\sigma_0u_1 \dots u_{i-1}\sigma_{i-1} \in L_1^{\mathrm{c}}$. Then $u_0\sigma_0u_1 \dots u_{i-1}\sigma_{i-1}u_i\sigma_i \in L_1$ and because of Lemma 3.5, $u_0\sigma_0 \dots \sigma_{i-1}u_i\sigma_i \in L_1^{\mathrm{c}}$. As this holds for all $i = 1, \dots, m$, the result is $u_0\sigma_0 \dots u_m\sigma_m = s_{\mathrm{en}} \in L_1^{\mathrm{c}}$. $\square$

Using the above Lemmas, hierarchical consistency is proven.

Proof: Validity of the supervisor S^{lo}, i.e. $p^{\mathrm{hi}}(L_1^{\mathrm{c}}) \subseteq L_1^{\mathrm{hi,c}}$, is shown by induction. First, $\varepsilon \in L_1^{\mathrm{c}}$ follows from the admissibility of S^{lo} (see also Lemma 3.4). Now let $s \in L_1^{\mathrm{c}}$, $s\sigma \in L_1^{\mathrm{c}}$ for some $\sigma \in \Sigma$ and $s^{\mathrm{hi}} := p^{\mathrm{hi}}(s) \in L_1^{\mathrm{hi,c}}$. It holds that either $\sigma \in \Sigma^{\mathrm{hi}}$ or $\sigma \in (\Sigma - \Sigma^{\mathrm{hi}})$. In the first case, $\sigma \in S^{\mathrm{hi}}(s^{\mathrm{hi}})$ as $\sigma \in S^{\mathrm{lo}}(s) = S^{\mathrm{hi}}(s^{\mathrm{hi}}) \cup (\Sigma - \Sigma^{\mathrm{hi}})$ and $\sigma \notin (\Sigma - \Sigma^{\mathrm{hi}})$. In the second case, $p^{\mathrm{hi}}(s\sigma) = p^{\mathrm{hi}}(s) = s^{\mathrm{hi}} \in L_1^{\mathrm{hi,c}}$. For the reverse direction, it has to be proven that $L_1^{\mathrm{hi,c}} \subseteq p^{\mathrm{hi}}(L_1^{\mathrm{c}})$. Assume that $s^{\mathrm{hi}} \in L_1^{\mathrm{hi,c}}$. Then there exists a $s_{\mathrm{en}} \in L_{\mathrm{en},s^{\mathrm{hi}}}$ as $L_1^{\mathrm{hi,c}} \subseteq L_1^{\mathrm{hi}} = p^{\mathrm{hi}}(L_1)$. But this means $s_{\mathrm{en}} \in L_1^{\mathrm{c}}$ with Lemma 3.6. □

Summing up, the consistent implementation is a straightforward supervisor implementation that guarantees hierarchical consistency without further requirements on the system behavior. This means any hierarchical closed-loop system equipped with a consistent implementation is hierarchically consistent. However, it need not be the case that the closed-loop low-level system is nonblocking. It can happen that although the desired high-level behavior can be achieved by low-level control, there are local paths which lead to deadlock or livelock situations. This is demonstrated in the following example.

Example 3.2
Let H, H^{hi} and S^{hi} be the control system, abstracted control system and high-level supervisor of Example 3.1, respectively. The local control system of the high-level string α with the corresponding entry string $\alpha \in L_{\mathrm{en},\alpha}$ is $H_{\alpha,\alpha} = (L_{\alpha,\alpha}, L_{\alpha,\alpha,\{\alpha,\beta,\gamma\}})$ with $L_{\alpha,\alpha} = \overline{\mathtt{a}\big(\mathtt{ba}(\mathtt{dc})^*(\gamma + \mathtt{d}\beta) + \mathtt{d}\gamma\big)}$ and $L_{\alpha,\alpha,\{\alpha,\beta,\gamma\}} = \mathtt{a}\big(\mathtt{ba}(\mathtt{dc})^*(\varepsilon + \gamma + \mathtt{d}\beta) + \mathtt{d}\gamma\big)$. As $S^{\mathrm{hi}}(\alpha) \cap \Sigma^{\mathrm{hi}}(\alpha) = \{\alpha,\beta\} \cap \{\beta,\gamma\} \neq \emptyset$, the consistent implementation for the high-level string α is

$$S^{\mathrm{lo}}(s) := \begin{cases} \{\alpha,\beta,\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\} & \text{if } s \in \alpha\, \overline{\mathtt{a}(\mathtt{ba}(\mathtt{dc})^* + \mathtt{d})} \\ \Sigma & \text{otherwise.} \end{cases} \tag{3.3}$$

The resulting closed-loop behavior is $S^{\mathrm{lo}}/H = (L_1^{\mathrm{c}}, L_2^{\mathrm{c}})$ with $L_1^{\mathrm{c}} = \overline{\alpha\mathtt{a}\big(\mathtt{ba}(\mathtt{dc})^*\mathtt{d}\beta + \mathtt{d}\big)}$ and $L_2^{\mathrm{c}} = \alpha\mathtt{aba}(\mathtt{dc})^*(\varepsilon + \mathtt{d}\beta)$. It is readily observed that the hierarchical control system is hierarchically consistent but blocking, as $L_1^{\mathrm{c}} \neq \overline{L_2^{\mathrm{c}}}$. □

In Example 3.2, nonblocking control fails because of two reasons.

(i) the high level considers the string α as marked although there are both marked and non marked corresponding low-level strings $\alpha\mathtt{aba}(\mathtt{dc})^*$ and $\alpha\mathtt{ad}$, respectively.

(ii) the high level assumes that the event β can always be generated after the occurrence of α, but this is not possible after the low-level string $\alpha\mathtt{ad}$.

3.3 Nonblocking Control

The two blocking issues from above are addressed by investigating sufficient conditions for nonblocking control of hierarchical closed-loop systems. Therefore, two additional conditions — *marked string acceptance* and *locally nonblocking projected control systems* — are introduced. Before discussing these properties, the set of exit strings is defined. It contains all strings which have a high-level successor event.

Definition 3.7 (Exit Strings)
Let $P = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a projected control system, and assume $s^{\mathrm{hi}} \in L_1^{\mathrm{hi}}$. The set of exit strings of s^{hi} is

$$L_{\mathrm{ex},s^{\mathrm{hi}}} := \{s \in L_1 | p^{\mathrm{hi}}(s) = s^{\mathrm{hi}} \wedge (\exists \sigma^{\mathrm{hi}} \in \Sigma^{\mathrm{hi}} \text{ s.t. } s\sigma^{\mathrm{hi}} \in L_1)\} \subseteq \Sigma^*.$$

□

In Example 3.2, one reason why nonblocking control fails is that not all local strings corresponding to marked high-level strings are also marked. A solution to this problem is the requirement that if the high-level observes a string in L_2^{hi}, the low-level also has to pass a string in L_2. This means if a high-level string s^{hi} is contained in the language L_2^{hi}, then it must be guaranteed that any low-level string which is projected to s^{hi} and which has a high-level successor event, must have a prefix in L_2 and with the same projection s^{hi}. This property is denoted *marked string acceptance*.

Definition 3.8 (Marked String Acceptance)
Let $P = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a projected control system. The string $s^{\mathrm{hi}} \in L_2^{\mathrm{hi}}$ is marked string accepting[6] if for all $s_{\mathrm{ex}} \in L_{\mathrm{ex},s^{\mathrm{hi}}}$

$$\exists s' \leq s_{\mathrm{ex}} \text{ with } p^{\mathrm{hi}}(s') = s^{\mathrm{hi}} \text{ and } s' \in L_2.$$

P is marked string accepting if s^{hi} is marked string accepting for all $s^{\mathrm{hi}} \in L_2^{\mathrm{hi}}$. □

Example 3.3
The hierarchical closed-loop system in Example 3.1 is not marked string accepting. For the marked high-level string α, the string $\alpha\mathtt{ad}$ is an exit string in $L_{\mathrm{ex},\alpha}$ but there is no string $s' \in L_2$ s.t. $s' \leq \alpha\mathtt{ad}$ and $p^{\mathrm{hi}}(s') = \alpha$. □

The second issue in Example 3.2 originates from the construction of the low-level supervisor. It is based on the assumption that after a low-level string, all high-level events which are feasible in the corresponding high-level string can be generated.

[6]Note that $s^{\mathrm{hi}} \in L_1^{\mathrm{hi}} - L_2^{\mathrm{hi}} \Rightarrow (p^{\mathrm{hi}})^{-1}(s^{\mathrm{hi}}) \cap L_2 = \emptyset$.

The property for dealing with this issue is equivalent to the *observer* property in [WW96, Pu00, Won04]. We state it for the choice of the natural projection as a causal reporter map, where the high-level events are a subset of the event set of the low-level system. Systems with this property are denoted *locally nonblocking projected control systems* in the sequel.

Low-level strings s fulfill the following condition if the projected control system is locally nonblocking: For all high-level events which are feasible after the corresponding high-level string $p^{\mathrm{hi}}(s)$, a local path starting from s must exist, such that the high-level event is possible.

Definition 3.9 (Locally Nonblocking Projected Control Systems)
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a projected control system. The string $s^{\mathrm{hi}} \in L_1^{\mathrm{hi}}$ is locally nonblocking if for all $s \in L_1$ with $p^{\mathrm{hi}}(s) = s^{\mathrm{hi}}$ and $\forall \sigma \in \Sigma^{\mathrm{hi}}(s^{\mathrm{hi}})$, $\exists u_\sigma \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ s.t. $su_\sigma\sigma \in L_1$. $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ is locally nonblocking if s^{hi} is locally nonblocking $\forall s^{\mathrm{hi}} \in L_1^{\mathrm{hi}}$. □

Locally nonblocking projected control systems prove very useful in the decentralized framework which is addressed in Chapter 4. Yet, there are also interesting results concerning nonblocking control for the monolithic hierarchical architecture. Two alternative requirements are discussed in the subsequent sections that — apart from each other — guarantee hierarchical consistent and nonblocking behavior of the HCLS. First, a condition for the high-level closed-loop behavior is investigated. It is required to be *live*, i.e. any of its strings must have successor events. The second condition involves the system structure of the projected control system. It deals with low-level strings corresponding to high-level strings without any successor events in the high-level closed loop behavior.

3.3.1 Condition on the High-level Closed Loop

A language is *live*, if any of its strings can be extended by some successor event.

Definition 3.10 (Live Regular Language)
A regular language $L \in \Sigma^*$ is called live if $\forall s \in L, \exists \sigma \in \Sigma$ s.t. $s\sigma \in L$. □

Looking at hierarchical closed-loop systems, the high-level closed-loop language being live means that there is always a continuation of high-level strings. Together with the locally nonblocking condition, this means that also low-level strings can always be extended and thus the low level can never get stuck.

In the following theorem, we consider locally nonblocking and marked string accepting projected control systems with a consistent implementation. The additional requirement of a live high-level closed-loop language guarantees that the hierarchical closed-loop system is nonblocking and hierarchically consistent.

Theorem 3.1 (Live Nonblocking Control [SPM05])
Let $(H, p^{\text{hi}}, H^{\text{hi}}, S^{\text{hi}}, S^{\text{lo}})$ be a hierarchical closed-loop system with a consistent implementation. Also let the projected control system $(H, p^{\text{hi}}, H^{\text{hi}})$ be marked string accepting and locally nonblocking. If the high-level closed-loop language $L_1^{\text{hi,c}}$ is live, then S^{lo} solves the hierarchical control problem in Definition 3.2, and the HCLS is hierarchically consistent. $\square$

The following three technical Lemmas support the proof of Theorem 3.1.

Lemma 3.7 states that if a locally nonblocking projected control system is equipped with a consistent implementation of a nonblocking high-level supervisor, then the resulting projected control system is again locally nonblocking.

Lemma 3.7
Let $(H, p^{\text{hi}}, H^{\text{hi}})$ be a locally nonblocking projected control system, and let S^{hi} be a high-level supervisor. Also let S^{lo} be a consistent implementation of S^{hi}. Then $(S^{\text{lo}}/H, p^{\text{hi}}, S^{\text{hi}}/H^{\text{hi}})$ is a locally nonblocking projected control system. $\square$

Proof: Let $s^{\text{hi}} \in L_1^{\text{hi,c}}$ and let $s \in L_1^{\text{c}}$ s.t. $p^{\text{hi}}(s) = s^{\text{hi}}$. For proving Lemma 3.7, it has to be shown that $\forall\sigma \in S^{\text{hi}}(s^{\text{hi}}) \cap \Sigma^{\text{hi}}(s^{\text{hi}})$ there exists $u_\sigma \in (\Sigma - \Sigma^{\text{hi}})^*$ s.t. $su_\sigma\sigma \in L_1^{\text{c}}$. If $S^{\text{hi}}(s^{\text{hi}}) \cap \Sigma^{\text{hi}}(s^{\text{hi}}) = \emptyset$, the condition is fulfilled automatically. Thus $\sigma \in S^{\text{hi}}(s^{\text{hi}}) \cap \Sigma^{\text{hi}}(s^{\text{hi}}) \neq \emptyset$ is assumed. As $(H, p^{\text{hi}}, H^{\text{hi}})$ is locally nonblocking, there exists $u_\sigma \in (\Sigma - \Sigma^{\text{hi}})^*$ s.t. $su_\sigma\sigma \in L_1$. Then, because of Lemma 3.5, $su_\sigma\sigma \in L_1^{\text{c}}$. As s and σ were chosen arbitrarily, Lemma 3.7 is true. $\square$

It is also valid that if a high-level string can be extended in the high-level closed-loop behavior, then any corresponding low-level string can be extended such that its projection yields the extended high-level string. This is shown in the subsequent lemma.

Lemma 3.8
Let $(H, p^{\text{hi}}, H^{\text{hi}}, S^{\text{hi}}, S^{\text{lo}})$ be a hierarchical closed-loop system with a locally nonblocking projected control system $(H, p^{\text{hi}}, H^{\text{hi}})$, and let S^{lo} be a consistent implementation. Assume $s \in L_1^{\text{c}}$ and $s^{\text{hi}} := p^{\text{hi}}(s) \in L_1^{\text{hi,c}}$. If $t \in (\Sigma^{\text{hi}})^*$ s.t. $s^{\text{hi}}t \in L_1^{\text{hi,c}}$, then there exists a $u \in \Sigma^*$ with $p^{\text{hi}}(u) = t$ and $su \in L_1^{\text{c}} \cap L_{\text{en},s^{\text{hi}}t}$. $\square$

Proof: Let s^{hi}, s and t be given as in Lemma 3.8. Defining $u_0 = \sigma_0 = \varepsilon$, t can be represented as $t = \sigma_0\sigma_1\sigma_2\cdots\sigma_m$ with $\sigma_i \in \Sigma^{\text{hi}}$ for $i = 1,\dots,m$. First it is shown that there exists a string $u = u_0\sigma_0u_1\sigma_1\cdots u_m\sigma_m \in \Sigma^*$ with $u_i \in (\Sigma - \Sigma^{\text{hi}})^*$ for $i = 1,\dots,m$ s.t. $su \in L_1^{\text{c}}$ by induction. The base case is easily verified as $su_0\sigma_0 = s \in L_1^{\text{c}}$. For the induction step, let $u_0\sigma_0u_1\sigma_1\cdots u_i\sigma_i \in \Sigma^*$ s.t. $su_0\sigma_0u_1\cdots\sigma_i \in L_1^{\text{c}}$ for $i \in 1,\dots,m$. Then, as $(S^{\text{lo}}/H, p^{\text{hi}}, S^{\text{hi}}/H^{\text{hi}})$ is locally nonblocking (Definition 3.9 and Lemma 3.7), there exists $u_{i+1} \in (\Sigma - \Sigma^{\text{hi}})^*$ s.t. $su_0\sigma_0u_1\cdots\sigma_iu_{i+1}\sigma_{i+1} \in L_1^{\text{c}}$. As this applies for all $i = 0,\dots,m$, it holds that $su = su_0\sigma_0\cdots u_m\sigma_m \in L_1^{\text{c}}$ and $p^{\text{hi}}(u) = t$. Because of the construction of u and with Definition 3.3, $su \in L_{\text{en},s^{\text{hi}}t} \cap L_1^{\text{c}}$. $\square$ $\square$

It is also valid that every entry string in the low-level closed-loop behavior can be extended to a marked low-level string if the projected control system is marked string accepting and the hierarchical closed-loop system has a consistent implementation.

Lemma 3.9
Let $(H, p^{\text{hi}}, H^{\text{hi}}, S^{\text{hi}}, S^{\text{lo}})$ be a hierarchical closed-loop system with a marked string accepting and locally nonblocking projected control system $(H, p^{\text{hi}}, H^{\text{hi}})$ and let S^{lo} be a consistent implementation. Also let $s_{\text{en}} \in L_{\text{en},s^{\text{hi}}} \cap L_1^{\text{c}}$ for $s^{\text{hi}} \in L_2^{\text{hi,c}}$ with $S^{\text{hi}}(s^{\text{hi}}) \cap \Sigma^{\text{hi}}(s^{\text{hi}}) \neq \emptyset$. Then there exists $u' \in (\Sigma - \Sigma^{\text{hi}})^*$ s.t. $s_{\text{en}}u' \in L_2^{\text{c}}$. □

Proof: Let $\sigma \in S^{\text{hi}}(s^{\text{hi}}) \cap \Sigma^{\text{hi}}(s^{\text{hi}}) \neq \emptyset$. Considering that $(H, p^{\text{hi}}, H^{\text{hi}})$ is locally nonblocking and with Lemma 3.5, there exists $u \in (\Sigma - \Sigma^{\text{hi}})^*$ s.t. $s_{\text{en}}u\sigma \in L_1^{\text{c}}$. Then, because of Definition 3.7, $s_{\text{en}}u \in L_{\text{ex},s^{\text{hi}}}$. Marked string acceptance states that there exists $u' \leq u$ s.t. $s_{\text{en}}u' \in L_2$. Because of the consistent implementation, $s_{\text{en}}u' \in L_2^{\text{c}}$. □

Now, Theorem 3.1 can be proven.

Proof: Let $s \in L_1^{\text{c}}$ and $s^{\text{hi}} := p^{\text{hi}}(s) \in L_1^{\text{hi,c}}$. Then, as $L_1^{\text{hi,c}}$ is live, $\exists \sigma_1 \in \Sigma^{\text{hi}}$ s.t. $s^{\text{hi}}\sigma_1 \in L_1^{\text{hi,c}}$. As S^{hi} is nonblocking, $\exists t = \sigma_2 \cdots \sigma_m \in (\Sigma^{\text{hi}})^*$ s.t. $s^{\text{hi}}\sigma_1 t \in L_2^{\text{hi,c}}$. Considering Lemma 3.8, $\exists u \in \Sigma^*$ s.t. $su \in L_1^{\text{c}} \cap L_{\text{en},s^{\text{hi}}\sigma_1 t}$. Then, using Lemma 3.9, $\exists u' \in (\Sigma - \Sigma^{\text{hi}})^*$ s.t. $suu' \in L_2^{\text{c}}$ and hence $s \in \overline{L_2^{\text{c}}}$. □

Recapitulating, three conditions have to be fulfilled. Projected control systems are required to be marked string accepting and locally nonblocking. Furthermore the high-level closed-loop behavior has to be live. Altogether, these conditions guarantee nonblocking behavior of the hierarchical closed-loop system, if a particular low-level supervisor implementation — the consistent implementation — is chosen.

Marked string acceptance ensures that if a marked string is passed in the high-level, then also a marked low-level string is passed. If the control system is *locally nonblocking*, then any low-level string can be extended to generate the high-level events which are feasible after the corresponding high-level string. Together with the *consistent implementation* which allows all low-level paths by default, this condition guarantees that if a high-level event is enabled after some high-level string, then for any corresponding low-level string, there is an extension containing the high-level event. Considering *liveness*, this means that the closed-loop system will not get stuck, as there are always enabled high-level events and thus any low-level path can be extended. This already explains why the hierarchical control system is hierarchically consistent. Combining the last observation with *marked string acceptance* yields nonblocking behavior. It is interesting to note that the second part in the definition of the consistent implementation (Definition 3.5) is never used as the high-level closed-loop system is live. This observation leads to the following lemma.

Lemma 3.10
Let $(H, p^{\text{hi}}, H^{\text{hi}}, S^{\text{hi}}, S^{\text{lo}})$ be a hierarchical closed-loop system with a consistent implementation and the low-level closed-loop system S^{lo}/H. Then it holds that

$$L_1^{\text{c}} = L_1^{\text{hi,c}} || L_1.$$

□

Proof: At first note that $L_1^{\text{hi,c}}||L_1 = L^{\text{hi,c}}||(\Sigma - \Sigma^{\text{hi}})^* \cap L_1$, as $L^{\text{hi,c}} \subseteq (\Sigma^{\text{hi}})^*$ and $L_1 \subseteq \Sigma^*$. As S^{hi} is admissible, it holds that $\varepsilon \in L^{\text{hi,c}}$. Observing that $\varepsilon \in L_1$, it is also true that $\varepsilon \in L^{\text{hi,c}}||(\Sigma - \Sigma^{\text{hi}})^* \cap L_1$. Analogously, $\varepsilon \in L_1^{\text{c}}$ because S^{lo} is an admissible supervisor. Starting from this, Lemma 3.10 is proven by induction. Assume $s \in L_1^{\text{c}}$ and $s \in L^{\text{hi,c}}||(\Sigma - \Sigma^{\text{hi}})^* \cap L_1$.

It is first shown that $L_1^{\text{c}} \subseteq L^{\text{hi,c}}||(\Sigma - \Sigma^{\text{hi}})^* \cap L_1$. Let $\sigma \in \Sigma$ s.t. $s\sigma \in L_1^{\text{c}}$. Then $s\sigma \in L_1$ and $\sigma \in S^{\text{lo}}(s)$. If $\sigma \in \Sigma^{\text{hi}}$, then $\sigma \in S^{\text{hi}}(s^{\text{hi}})$ because of Definition 3.5 and $\sigma \notin (\Sigma - \Sigma^{\text{hi}})$. Thus $s\sigma \in L_1^{\text{hi,c}}||(\Sigma - \Sigma^{\text{hi}})^*$. If $\sigma \in (\Sigma - \Sigma^{\text{hi}})$, then $s\sigma \in L_1^{\text{hi,c}}||(\Sigma - \Sigma^{\text{hi}})^*$, too. Thus $s\sigma \in L_1^{\text{hi,c}}||(\Sigma - \Sigma^{\text{hi}})^* \cap L_1$.

For the reverse direction, $L^{\text{hi,c}}||(\Sigma - \Sigma^{\text{hi}})^* \cap L_1 \subseteq L_1^{\text{c}}$ has to be shown. Let $\sigma \in \Sigma$ s.t. $s\sigma \in L_1^{\text{hi,c}}||(\Sigma - \Sigma^{\text{hi}})^* \cap L_1$. Then $s\sigma \in L_1$ and $p^{\text{hi}}(s\sigma) \in L_1^{\text{hi,c}}$. If $\sigma \in \Sigma^{\text{hi}}$, then $p^{\text{hi}}(s)\sigma \in L_1^{\text{hi,c}}$ and thus $\sigma \in S^{\text{hi}}(s^{\text{hi}})$. Because of Definition 3.5, $\sigma \in S^{\text{lo}}(s)$ and hence $s\sigma \in L_1^{\text{c}}$. If $\sigma \in (\Sigma - \Sigma^{\text{hi}})$, then $\sigma \in S^{\text{lo}}(s)$, too. Consequently $s\sigma \in L_1^{\text{c}}$. □

The above result is of particular interest for implementation purposes. It states that the high-level closed-loop system can directly be used as the low-level supervisor, i.e. now additional low-level supervisor has to be computed. This facilitates implementing the supervisor tremendously as will be shown in Section 3.4.6.

In the above consideration, the *liveness* condition made sure that the control system cannot get stuck.

Now we consider the case that the high-level closed loop system is not supposed to be live. Then, it is possible that a marked high-level string cannot be extended any further. According to Definition 3.5, corresponding low-level strings (which need not be marked) also might not have a further extension. This leads to blocking in the low level. Thus, for high-level strings with no successor events, a controllability computation for the local control system of the respective high-level string is necessary. The next section addresses this issue, and *marked string controllability* is introduced to solve the problem.

3.3.2 Structural Condition

In case a high-level supervisor disables all events after some high-level string, a nonblocking low-level supervisor must take care of the possible future local behaviors as no more changes in the high-level can happen. This is achieved by computing the supremal controllable sublanguage of the second language (L_2) of the local control system w.r.t. its first language (L_1). A nonblocking supervisor can only be implemented if this supremal controllable sublanguage is nonempty. This is guaranteed if the *marked string controllability* condition is fulfilled.[7]

Definition 3.11 (Marked String Controllability)
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a projected control system. Let $s^{\mathrm{hi}} \in L_2^{\mathrm{hi}}$ with $\gamma^{\mathrm{hi}} \in \Gamma^{\mathrm{hi}}$ s.t. $\gamma^{\mathrm{hi}} \cap \Sigma^{\mathrm{hi}}(s^{\mathrm{hi}}) = \emptyset$. s^{hi} is marked string controllable if for all $s_{\mathrm{en}} \in L_{\mathrm{en},s^{\mathrm{hi}}}$, the language $\kappa_{L_{s_{\mathrm{en}},1}}(L_{s_{\mathrm{en}},2}) \neq \emptyset$. $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ is marked string controllable if s^{hi} is marked string controllable $\forall s^{\mathrm{hi}} \in L_2^{\mathrm{hi}}$. $\square$

With the above property, it is no longer necessary that the high-level closed-loop behavior must always be able to generate events. Replacing liveness with marked string controllability, the following theorem provides the same result as Theorem 3.1.

Theorem 3.2 (Nonblocking Hierarchical Control)
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ be a hierarchical closed-loop system with a marked string accepting, marked string controllable and locally nonblocking projected control system $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$. Also let S^{hi} be a high-level supervisor with a consistent implementation S^{lo}. Then S^{lo} solves the hierarchical control problem in Definition 3.2, and the HCLS is hierarchically consistent. $\square$

The proof is similar to the proof of Theorem 3.1. It also accounts for the case that there are high-level strings which cannot be extended.

Proof: Hierarchical consistency directly follows from Proposition 3.1.

For proving nonblocking supervision, it has to be shown that $\forall s \in L_1^{\mathrm{c}}$, $\exists u \in \Sigma^*$ s.t. $su \in L_2^{\mathrm{c}}$. Because of hierarchical consistency, $s^{\mathrm{hi}} := p^{\mathrm{hi}}(s) \in L_1^{\mathrm{hi,c}}$. There are two cases. First let $S^{\mathrm{hi}}(s^{\mathrm{hi}}) \cap \Sigma^{\mathrm{hi}}(s^{\mathrm{hi}}) = \emptyset$. Then, writing $s = s_{\mathrm{en}}u'$ with $s_{\mathrm{en}} \in L_{\mathrm{en},s^{\mathrm{hi}}}$, $u' \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ and noting that $s \in L_1^{\mathrm{c}}$, it holds that $u' \in \overline{\kappa_{L_{s_{\mathrm{en}},1}}(L_{s_{\mathrm{en}},2})}$. Thus, there exists $u'' \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ s.t. $u = u'u'' \in \kappa_{L_{s_{\mathrm{en}},1}}(L_{s_{\mathrm{en}},2})$. Because of the consistent implementation, $su \in L_2^{\mathrm{c}}$. Now let $S^{\mathrm{hi}}(s^{\mathrm{hi}}) \cap \Sigma^{\mathrm{hi}}(s^{\mathrm{hi}}) \neq \emptyset$. As S^{hi} is nonblocking, there exists $t \neq \varepsilon$ s.t. $s^{\mathrm{hi}}t \in L_2^{\mathrm{hi,c}}$. Because of Lemma 3.7 and Lemma 3.8, there is $u' \in \Sigma^*$ s.t. $su' \in L_1^{\mathrm{c}} \cap L_{\mathrm{en},s^{\mathrm{hi}}t}$. Then, considering that $s^{\mathrm{hi}}t \in L_2^{\mathrm{hi,c}}$ and $su' \in L_{\mathrm{en},s^{\mathrm{hi}}t}$, there exists $u'' \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ s.t. $su'u'' \in L_2^{\mathrm{c}}$ because of Lemma 3.9. In both cases, $u = u'u'' \in \Sigma^*$ s.t. $su \in L_2^{\mathrm{c}}$. $\square$

Thus, if liveness of the controlled high-level behavior is not given, marked string controllability ensures nonblocking behavior of the hierarchical control system.[8]

In this section, two theorems for nonblocking and hierarchically consistent control were derived. Nonblocking supervision was established by focusing on two cases. In the first case, the high-level closed-loop system was required to be live. In the second case, a structural condition for local

[7] Note that the definition of the consistent implementation already captures this case.
[8] Hierarchical consistency is already guaranteed by the consistent implementation.

behaviors was needed. In the next section the language-based results are worked out in a finite automaton framework. In this representation, algorithms for checking the above conditions are elaborated.

3.4 Hierarchical Control For Finite Automata

It has been pointed out how nonblocking hierarchical control for discrete event systems can be achieved. For this purpose, a language based framework with several conditions on control systems was provided. We rewrite the above results in an automata framework for algorithmic realization of these concepts. The relevant algorithms for verifying the required conditions and for synthesizing supervisory controllers are also provided.

We use the following notation for a high-level automaton G^{hi} and a low-level automaton G, for investigating the computational complexity of the algorithms.

- G^{hi}: number of states: $n^{\text{hi}} := |X^{\text{hi}}|$ and number of events: $e^{\text{hi}} := |\Sigma^{\text{hi}}|$.
- G: number of states: $n := |X|$.

3.4.1 Natural Projection

From Lemma 2.9, it is clear that any projected system $(H, p^{\text{hi}}, H^{\text{hi}})$ can be represented by automata G and G^{hi}, where $L(G) = L_1$, $L_{\text{m}}(G) = L_2$, and $L(G^{\text{hi}}) = L_1^{\text{hi}}$, $L_{\text{m}}(G^{\text{hi}}) = L_2^{\text{hi}}$, respectively. In addition to that Lemma 2.11 states that the supervisors in a hierarchical closed-loop system can be implemented as automata, too. If the hierarchical closed-loop system is finite[9], then any of the above automata is finite, and thus a finite representation of the HCLS is given.

Corollary 3.1 (Automata Representation of a Hierarchical Closed Loop System)
Let $Q = (H, p^{\text{hi}}, H^{\text{hi}}, S^{\text{lo}}, S^{\text{hi}})$ be a hierarchical closed-loop system. There exists a finite automata representation $(G, G^{\text{hi}}, R, R^{\text{hi}})$, where $G, G^{\text{hi}}, R, R^{\text{hi}}$ are finite automata, s.t.

(i) $H = (L(G), L_{\text{m}}(G))$,

(ii) $H^{\text{hi}} = (L(G^{\text{hi}}), L_{\text{m}}(G^{\text{hi}}))$,

(iii) $S^{\text{lo}}/H = (L(R), L_{\text{m}}(R))$,

(iv) $S^{\text{hi}}/H^{\text{hi}} = (L(R^{\text{hi}}), L_{\text{m}}(R^{\text{hi}}))$.

[9] Recall that a hierarchical closed-loop system is finite if its languages are regular.

$\square$

Proof: Item (i) and (ii) directly follow from Lemma 2.9. Corollary 3.1 states that S^{lo}/H and $S^{\text{hi}}/H^{\text{hi}}$ are control systems and thus applying Lemma 2.9 proves (iii) and (iv). $\square$

Considering this result raises the question of computing the various automata. The automaton G is the system model, and it is the automaton which is provided by the system designer or modeler. For determining the automata representation of the abstracted system behavior G^{hi}, an implementation of the natural projection is needed.

To this end, a nondeterministic automaton G_{nd}, generating and recognizing the languages of the projected control system is constructed. This is done by eliminating local paths in the given automaton $G = (\Sigma, X, \delta, x_0, X_{\text{m}})$.[10] $G_{\text{nd}} = (\Sigma^{\text{hi}}, X_{\text{nd}}, \delta_{\text{nd}}, x_{0,\text{nd}}, X_{\text{m,nd}})$ has the high-level event set Σ^{hi} as its alphabet and its state sets $X_{\text{nd}} = X$, $x_{0,\text{nd}} = x_0$ and $X_{\text{m,nd}} = X_{\text{m}}$ are directly adopted from G.

The transition function of G_{nd} is given for every state $x \in X$ as follows. For any $\sigma \in \Sigma^{\text{hi}}$

$$\delta_{\text{nd}}(x,\sigma) := \begin{cases} \tilde{x} & \text{if } \delta(x, u\sigma) = \tilde{x} \text{ for } u \in (\Sigma - \Sigma^{\text{hi}})^* \\ \text{not defined} & \text{else} \end{cases}$$

For constructing a deterministic high-level automaton from G_{nd}, Lemma 2.4 can be used, and the corresponding algorithm ([HU79]) can be applied.[11] The result of this computation is a deterministic automaton G^{hi}, every state of which corresponds to a set of states in G_{nd}. The function mapping a high-level state to its corresponding set of states in G_{nd} is defined as $f^{\text{hilo}} : X^{\text{hi}} \to 2^X$ with $f^{\text{hilo}}(x^{\text{hi}}) := \{x \in X_{\text{nd}} | \delta_{\text{nd}}(x_{0,\text{nd}}, s^{\text{hi}}) = x\}$ for $x^{\text{hi}} = \delta^{\text{hi}}(x_0^{\text{hi}}, s^{\text{hi}}) \in X^{\text{hi}}$.[12]

An algorithmic implementation of the natural projection operation is given in the sequel. The main function "compute_gnd" gets the low-level automaton G and the high-level alphabet Σ^{hi} as its inputs. In the main loop, the recursive function "hl_reachable" is called. It returns the states which are reachable via a string including a high-level event from a state in G. Note that in the theoretical representation, the state set of G_{nd} equals the state set of G, i.e. $X_{\text{nd}} = X$. In the practical computation, it turns out that there are states (states which are reached by local strings) in X_{nd} which are not reachable from the initial state. The algorithm just considers the reachable states and thus $X_{\text{nd}} \subseteq X$ in the resulting automaton.

[10] A local path is a sequence of low-level events in $\Sigma - \Sigma^{\text{hi}}$.

[11] Normally, this is the costly step in the computation. In the worst case, the complexity of the projection is exponential in the number of states of the original automaton. Yet, it will be shown that the complexity is polynomial for our approach.

[12] f^{hilo} is needed later.

```
/* Computation of $G_{\mathrm{nd}}$: compute_gnd */
compute_gnd($G$, $\Sigma^{\mathrm{hi}}$)

/* Initialization of the waiting list */
waiting $= \{x_0\}$

/* Initialization of $G_{\mathrm{nd}}$ */
$\Sigma_{\mathrm{nd}} = \Sigma^{\mathrm{hi}}$, $X_{\mathrm{nd}} = \{x_0\}$, $\delta_{\mathrm{nd}} =$ n.def., $X_{0,\mathrm{nd}} = \{x_0\}$, $X_{\mathrm{m,nd}} = X_{\mathrm{m}} \cap \{x_0\}$

/* Loop through all states in the waiting list */
while waiting $\neq \emptyset$
    pick $x \in$ waiting, set waiting $=$ waiting $- \{x\}$.
    /* Initialize the state set $x_{\mathrm{done}}$ for each cycle */
    $x_{\mathrm{done}} = \{x\}$
    /* Call the recursive function hl_reachable */
    $x_{\mathrm{next}} = x$
    $(x_{\mathrm{done}}, G_{\mathrm{nd}})$ = hl_reachable($x, x_{\mathrm{next}}, x_{\mathrm{done}}, \Sigma^{\mathrm{hi}}, G_{\mathrm{nd}}$),
end while
return($G_{\mathrm{nd}}$)
```

Figure 3.6: Computation of the nondeterministic automaton G_{nd}

Starting from the state x, the recursive function hl_reachable proceeds along local strings until a high-level event can occur. To this end, the function loops through all successor events of the current state x_{next} and checks if they are high-level events or not. In the first case, the successor state for the detected high-level event is a high-level successor of the state x. If it is not an element of X_{nd} yet, the transition function δ_{nd} of G_{nd} as well as the state sets X_{nd} and $X_{\mathrm{m,nd}}$ are updated and the new successor state is added to the waiting list. If it is an element of X_{nd}, just the transition function is updated. In the second case, the transition to the new state is not seen by the high level and thus the function "hl_reachable" is evaluated for the new state.

The recursive function terminates if either a state is reached which has already been investigated before, i.e. it is contained in the x_{done} list, or if all transitions have been examined.

As mentioned before, the automaton G_{nd} is nondeterministic. It can be represented by a deterministic finite automaton as stated in Lemma 2.4. An algorithm for computing such automaton G^{hi} is given in [HU79]. In this thesis, the function is called "gnd2ghi". Its output is a deterministic automaton G^{hi}, recognizing $L_{\text{m}}(G_{\text{nd}})$, i.e. $L_{\text{m}}(G^{\text{hi}}) = L_{\text{m}}(G_{\text{nd}})$.

```
/* Find all states which are reachable from x via a local path terminated with a high-level event:
hl_reachable */
hl_reachable($x$, $x_{\text{next}}$, $x_{\text{done}}$, $\Sigma^{\text{hi}}$, $G_{\text{nd}}$)
/* If $x_{\text{next}}$ was not investigated yet, it is put into the $x_{\text{done}}$ list */
if $x_{\text{next}} \notin x_{\text{done}}$
$x_{\text{done}} = x_{\text{done}} \cup \{x_{\text{next}}\}$
/* If $x_{\text{next}}$ was already examined, it need not be examined again */
else
return$(x_{\text{done}}, G_{\text{nd}})$
/* The current state x is marked if any of the locally reachable states is marked. */
if $x \notin X_{\text{m,nd}} \wedge x_{\text{next}} \in X_{\text{m}}$
    $X_{\text{m,nd}} = X_{\text{m,nd}} \cup \{x\}$
/* All transitions in $x_{\text{next}}$ are investigated */
$T$ = transitions($x_{\text{next}}$)
while($T \neq \emptyset$)
    pick $t \in T$, $T = T - \{t\}$
    /* If t is a high-level event, a new transition from the original state x is added to $\delta_{\text{nd}}$.
    if($t \in \Sigma^{\text{hi}}$)
        $\delta_{\text{nd}}(x,t) := \delta(x_{\text{next}},t)$
        If the new state is not an element of $X_{\text{nd}}$, it is added to $X_{\text{nd}}$ and to the waiting list */
        if($\delta(x_{\text{next}},t) \notin X_{\text{nd}}$)
            $X_{\text{nd}} = X_{\text{nd}} \cup \{\delta(x_{\text{next}},t)\}$
            waiting = waiting $\cup\{\delta(x_{\text{next}},t)\}$
    /* If t is a low-level event, the recursive function is called for the new successor state */
    else
        $(x_{\text{done}}, G_{\text{nd}})$ = hl_reachable($x, \delta(x_{\text{next}},t), x_{\text{done}}, \Sigma^{\text{hi}}, G_{\text{nd}}$)
end while
/* Base case of the recursive function: end of the loop is reached */
return$(x_{\text{done}}, G_{\text{nd}})$
```

Figure 3.7: Computation of locally reachable states

Combining the functions "compute_gnd" and "gnd2ghi" , it is possible to compute a deterministic automaton which recognizes the projected language $p^{\mathrm{hi}}(L_{\mathrm{m}}(G))$. The complete algorithm "projection" is shown in the next figure.

```
/* Compute a deterministic automaton $G_{\mathrm{d}}$ recognizing $p^{\mathrm{hi}}(L_{\mathrm{m}}(G)$: projection */
projection($G$, $\Sigma^{\mathrm{hi}}$)
/* Compute the nondeterministic automaton $G_{\mathrm{nd}}$ */
$G_{\mathrm{nd}}$ = compute_gnd($G$,$\Sigma^{\mathrm{hi}}$)
/* Compute the deterministic automaton $G^{\mathrm{hi}}$ */
$(G^{\mathrm{hi}}, f^{\mathrm{hilo}})$ = gnd2ghi($G_{\mathrm{nd}}$)
return($G^{\mathrm{hi}}$)
```

Figure 3.8: Computation of the projected automaton G^{hi}

The function f^{hilo} is explained in the next section. In the worst case, the number of states of a canonical recognizer for the natural projection of a language is exponential in the number of states of the original automaton. Nevertheless, [Won97] provides a more positive result which is adapted to the framework presented in this chapter.

Theorem 3.3 (Automata Representation of Projected Languages [Won97])
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a marked string accepting and locally nonblocking projected control system with the automata representation (G, G^{hi}). Then, G^{hi} has an equal or smaller number of states than G, i.e. $|X^{\mathrm{hi}}| \leq |X|$.[13] □

This means that for systems considered in this chapter, it is never the case that the projected automaton has a larger number of states than the original automaton. In applications, it turns out that the number of states of G^{hi} is smaller than the number of states of G. The proof of Theorem 3.3 is given in Appendix A.2. It is based on the fact that locally nonblocking and marked string accepting projected control systems obtain the observer property used in the result in [Won97].

For the time complexity of the natural projection, there is a similar result.

Theorem 3.4 (Time Complexity of the Natural Projection [Won97])
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a marked string accepting and locally nonblocking projected control system with the automata representation (G, G^{hi}). The time complexity of computing G^{hi} is at worst polynomial in the state size of G and the number of high-level events Σ^{hi}. □

[13]Note that both G and G^{hi} are canonical recognizers.

In [Won97], an algorithm with the complexity $\mathcal{O}(n^7(e^{\text{hi}})^2)$ is developed (n is the number of states of G and e^{hi} is the number of high-level events).

After providing an algorithm for computing the natural projection, it is possible to compute automata representations of projected control systems. In the next step, the properties established in Section 3.3 have to be verified. At first, marked string acceptance is inspected.

3.4.2 Algorithmic Verification of Marked String Acceptance

Marked string acceptance can be verified algorithmically using finite automata. Marked string acceptance fails, if for some marked high-level string, it is possible to find a local path from a corresponding entry string to an exit string without passing a marked string. For checking this property, a representation of entry strings in the automata framework ir required. It turns out, that this representation can be derived by evaluating the states of the automata G and G_{nd} defined above. It is denoted the *set of entry states* in the sequel.

Definition 3.12 (Entry States)
Let G and G^{hi} be given as above. Also assume $s^{\text{hi}} \in L(G^{\text{hi}})$ and $x^{\text{hi}} = \delta^{\text{hi}}(x_0^{\text{hi}}, s^{\text{hi}})$. The set of entry states $X_{\text{en},x^{\text{hi}}}$ is

$$X_{\text{en},x^{\text{hi}}} := \{x \in X | x = \delta(x_0, s_{\text{en}}) \text{ for } s_{\text{en}} \in L_{\text{en},s^{\text{hi}}}\} \subseteq X.$$

□

Regarding Definition 3.12, it is interesting to take a closer look at the map f^{hilo}. It turns out that the set of states $f^{\text{hilo}}(x^{\text{hi}})$ for some high-level state $x^{\text{hi}} \in X^{\text{hi}}$ equals the set $X_{\text{en},x^{\text{hi}}}$ of entry states of x^{hi}.

Lemma 3.11 (Entry States)
Let G, G^{hi} and f^{hilo} be given as above. Also assume $s^{\text{hi}} \in L(G^{\text{hi}})$ and $x^{\text{hi}} = \delta^{\text{hi}}(x_0^{\text{hi}}, s^{\text{hi}})$. The set of entry states $X_{\text{en},x^{\text{hi}}}$ of x^{hi} is

$$X_{\text{en},x^{\text{hi}}} = f^{\text{hilo}}(x^{\text{hi}}).$$

□

Proof: Assume $x_m \in X_{\text{en},x^{\text{hi}}}$. Then $\exists s_{\text{en}} \in L_{s_{\text{en}},s^{\text{hi}}}$ s.t. $\delta(x_0, s_{\text{en}}) = x_m$. s_{en} can be written as $s_{\text{en}} = u_0\sigma_0 \cdots u_m\sigma_m$ for $u_0 = \sigma_0 = \varepsilon$ and $u_i \in (\Sigma - \Sigma^{\text{hi}})^*$, $\sigma_i \in \Sigma^{\text{hi}}$ for $i = 1, \ldots, m$. $x_m \in \delta_{\text{nd}}(x_{0,\text{nd}}, s^{\text{hi}})$ is shown by induction. It holds that $x_{0,\text{nd}} \in \delta_{\text{nd}}(x_{0,\text{nd}}, \sigma_0)$ and $x_{0,\text{nd}} = x_0 \in \delta(x_0, \sigma_0)$. Now assume that $x_{i-1} \in \delta_{\text{nd}}(x_{0,\text{nd}}, \sigma_0 \cdots \sigma_{i-1})$ for $x_{i-1} = \delta(x_0, u_0\sigma_0 \cdots u_{i-1}\sigma_{i-1})$. Then $x_i \in \delta_{\text{nd}}(x_{i-1}, \sigma_i)$ for $x_i = \delta(x_0, u_0\sigma_0 \cdots u_i\sigma_i)$ as $u_i \in (\Sigma - \Sigma^{\text{hi}})^*$ and $\delta(x_{i-1}, u_i\sigma_i) = x_i$ and consequently $x_i \in \delta_{\text{nd}}(x_{0,\text{nd}}, \sigma_0 \cdots \sigma_i)$ with the definition of G_{nd}. As this is valid for all $i = 1, \ldots, m$, $x_m \in \delta_{\text{nd}}(x_{0,\text{nd}}, s^{\text{hi}})$ follows. Hence $x_m \in f^{\text{hilo}}(x^{\text{hi}})$.

Now let $x_m \in f^{\text{hilo}}(x^{\text{hi}})$. Then $x_m \in \delta_{\text{nd}}(x_{0,\text{nd}}, s^{\text{hi}})$ and thus $\exists s = u_0\sigma_0 \cdots u_m\sigma_m$ as above s.t. $\delta(x_0, s) = x_m$. But as $p^{\text{hi}}(s) = s^{\text{hi}}$, it holds that $s \in L_{\text{en},s^{\text{hi}}}$ and thus $x_m \in X_{\text{en},x^{\text{hi}}}$. □

In view of the above lemma, the map f^{hilo} is directly obtained from the construction of the automaton G_{nd}. As an effect, the set of entry states for each high-level state can easily be determined. Knowing that the transition function δ^{hi} of the deterministic automaton G^{hi} is unique, it is readily observed, that for every string in $L(G^{\text{hi}})$ there is exactly one state $x^{\text{hi}} \in X^{\text{hi}}$. Thus, a finite representation of the set of entry strings $L_{\text{en},s^{\text{hi}}}$ of a high-level string $s^{\text{hi}} \in L(G^{\text{hi}})$ is obtained by computing $x^{\text{hi}} = \delta(x_0^{\text{hi}}, s^{\text{hi}})$ and determining the set of entry states via $X_{\text{en},s^{\text{hi}}} = f^{\text{hilo}}(x^{\text{hi}})$.

The subsequent example illustrates the computation of a projected system from an automaton G and also describes the concept of entry states.

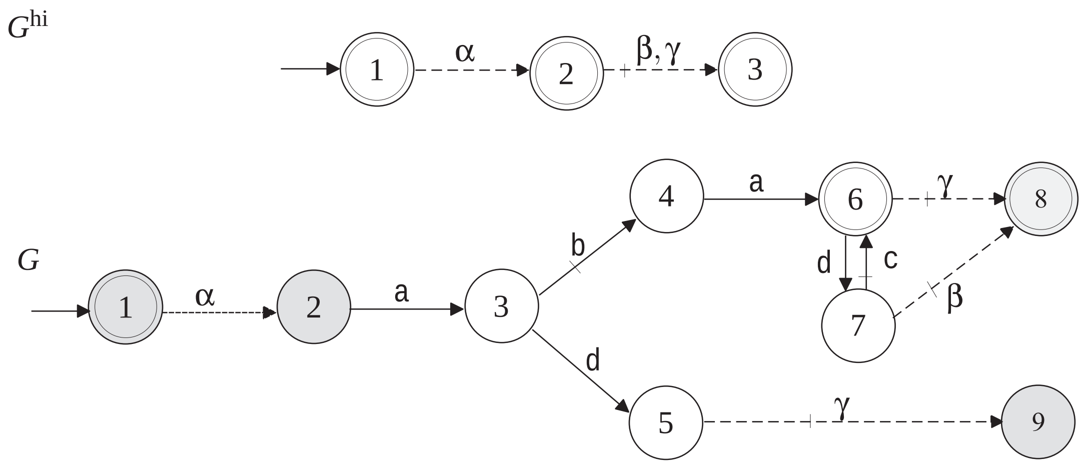


Figure 3.9: Projected control system with the low-level automaton G and the high-level automaton G^{hi}

Example 3.4
Let G in Figure 3.9 be the automata representation of the control system in Example 3.1.[14] For computing the abstracted automaton G^{hi}, all local paths have to be filtered out. The resulting high-level automaton is also depicted in Figure 3.9. All states of G^{hi} are marked, as there is always a low-level string in L_2 which is projected to the corresponding high-level string, e.g. $\varepsilon \in L_2$ is projected to $\varepsilon \in L_2^{\text{hi}}$ and $\alpha\mathtt{aba} \in L_2$ is projected to $\alpha \in L_2^{\text{hi}}$. The set of entry states of G is highlighted by the shaded nodes. It holds that $X_{\text{en},1} = \{1\}$, $X_{\text{en},2} = \{2\}$ and $X_{\text{en},3} = \{8,9\}$. □

The marked string controllability condition can be examined, as it is possible to represent entry strings as a finite set of entry states. It has to be verified for the automata implementation, that for every entry state of a marked high-level state, any local path must first reach a marked state before a high-level event is active. The corresponding algorithm is presented in Figure 3.10. The function "check_msa"is explained in Figure 3.11.

[14]Controllable transitions are labeled by a tick and high-level transitions are dashed.

```
/* Verify marked string acceptance: marked_string_acceptance */
marked_string_acceptance($G$, $G^{\mathrm{hi}}$, $f^{\mathrm{hilo}}$)

/* Investigate each marked high-level state */
$\tilde{X}_{\mathrm{m}}^{\mathrm{hi}} = X_{\mathrm{m}}^{\mathrm{hi}}$
while $\tilde{X}_{\mathrm{m}}^{\mathrm{hi}} \neq \emptyset$
    pick $x^{\mathrm{hi}} \in \tilde{X}_{\mathrm{m}}^{\mathrm{hi}}$, set $\tilde{X}_{\mathrm{m}}^{\mathrm{hi}} = \tilde{X}_{\mathrm{m}}^{\mathrm{hi}} - \{x^{\mathrm{hi}}\}$.
    /* Compute the set of entry states corresponding to $x^{\mathrm{hi}}$*/
    $\tilde{X}_{\mathrm{en}} = f^{\mathrm{hilo}}(x^{\mathrm{hi}})$

    /* Investigate all entry states */
    while $\tilde{X}_{\mathrm{en}} \neq \emptyset$
        pick $x_{\mathrm{en}} \in \tilde{X}_{\mathrm{en}}$, set $\tilde{X}_{\mathrm{en}} = \tilde{X}_{\mathrm{en}} - \{x_{\mathrm{en}}\}$.
        if !check_msa$(x_{\mathrm{en}}, X_{\mathrm{done}}, \Sigma^{\mathrm{hi}}, X_{\mathrm{m}})$
            return(false)
        end if
    end while
end while
return(true)
```

Figure 3.10: Verification of marked string acceptance

The function "check_msa"is a recursive function determining if the current low-level state is marked. If not, it is checked if a high-level event is active. If this is the case, marked string acceptance is violated as there exists a local path from an entry string to a high-level event without passing a marked state. The algorithmic description of "check_msa"is given in Figure 3.11.

For analyzing the computational complexity of the algorithm in Figure 3.10, the following items are considered.

(i) loop through all marked high-level states $x^{\mathrm{hi}} \in X_{\mathrm{m}}^{\mathrm{hi}}$. This set is bounded by n^{hi}.

(ii) loop through every entry state corresponding to x^{hi}. This set is bounded by n.

(ii) Perform the function "check_msa"for the entry state x corresponding to x^{hi}. The function is mainly a reachability computation on states which are locally reachable from x. This set is also bounded by n.

Consequently the complexity of the above algorithm is $\mathcal{O}(n^{\mathrm{hi}}n^2)$.

```
/* check_msa */
check_msa($x$, $X_{\mathrm{done}}$, $\Sigma^{\mathrm{hi}}$, $X_{\mathrm{m}}$)

/* Check if x is marked in the low level */
if $x \in X_{\mathrm{m}}$
    return(true)

/* Check the successor states and events of x */
else
    $\Sigma_x = \Lambda(x)$
    while $\Sigma_x \neq \emptyset$
        pick $\sigma \in \Sigma_x$, set $\Sigma_x = \Sigma_x - \{\sigma\}$

        /* if the successor event is a high-level event, marked string acceptance is violated */
        if $\sigma \in \Sigma^{\mathrm{hi}}$
            return(false)

        /* if the state has not been investigated yet */
        else if $\delta(x,\sigma) \notin X_{\mathrm{done}}$
            $x' = \delta(x,\sigma)$
            $X_{\mathrm{done}} = X_{\mathrm{done}} \cup \{x'\}$
            if $!\mathrm{check\_msa}(x', \Sigma^{\mathrm{hi}}, X_{\mathrm{done}}, X_{\mathrm{m}})$
                return(false)
            end if
        end if
    end while
end if

/* marked string acceptance is valid for the current state x */
return(true)
```

Figure 3.11: Function check_msa

3.4.3 Algorithmic Verification of the Locally Nonblocking Condition

For verifying if a projected system is locally nonblocking, local behaviors starting from entry strings are examined. It has to be determined if from any state in the local behavior corresponding to some high-level state and for each high-level event which is in the active event set of that high-level state, a local path terminating with the respective high-level event can be found.

For a projected control system $H, p^{\mathrm{hi}}, H^{\mathrm{hi}}$ and its automata realization G, G^{hi}, the locally nonblocking condition can be checked by the algorithm in Figure 3.12.

```
/* Verify Locally Nonblocking Condition: locally_nonblocking */
locally_nonblocking($G$, $G^{\mathrm{hi}}$, $f^{\mathrm{hilo}}$)

/* Investigate each high-level state */
$\tilde{X}^{\mathrm{hi}} = X^{\mathrm{hi}}$
while $\tilde{X}^{\mathrm{hi}} \neq \emptyset$
    pick $x^{\mathrm{hi}} \in \tilde{X}^{\mathrm{hi}}$, set $\tilde{X}^{\mathrm{hi}} = \tilde{X}^{\mathrm{hi}} - \{x^{\mathrm{hi}}\}$.
    /* Compute the set of low-level states corresponding to $x^{\mathrm{hi}}$ */
    $X^{\mathrm{lo}} = \mathrm{reach}(x^{\mathrm{hi}})$

    /* Investigate all low-level states */
    while $X^{\mathrm{lo}} \neq \emptyset$
        pick $x \in X^{\mathrm{lo}}$, set $X^{\mathrm{lo}} = X^{\mathrm{lo}} - \{x\}$.

        /* Investigate all forward paths (recursive function lnb) */
        var = lnb_reachability$(x, \Lambda^{\mathrm{hi}}(x^{\mathrm{hi}}), \Sigma_x^{\mathrm{hi}} = \emptyset, x_{\mathrm{done}} = \{x\})$
        if !var
            print("The hierarchical control system is not locally nonblocking")
            return(false)
    end while
end while

return(true)
```

Figure 3.12: Verification of the local nonblocking condition

For every high-level state $x^{\mathrm{hi}} \in X^{\mathrm{hi}}$, the set of corresponding low-level states is computed by a standard forward reachability computation $X^{\mathrm{lo}} = \mathrm{reach}(x^{\mathrm{hi}})$. The entry states $f^{\mathrm{hilo}}(x^{\mathrm{hi}})$ of x^{hi} are used as the seeds for this computation. For any state in X^{lo} it has to be checked if all high-level events $\Lambda^{\mathrm{hi}}(x^{\mathrm{hi}})$ can be reached via a local path starting from x. This is done by using the function "lnb_reachability", with the arguments $x \in X^{\mathrm{lo}}$ (current state), $\Lambda^{\mathrm{hi}}(x^{\mathrm{hi}})$ (high-level event set), Σ_x^{hi} (list of reachable high-level events) and X_{done} (list of states which have already been investigated). It returns true if all events in $\Lambda^{\mathrm{hi}}(x^{\mathrm{hi}})$ can be reached and false otherwise. If none of these computations returns "false", the projected control system is locally nonblocking. The function "lnb_reachability" is realized by the following algorithm.

```
/* Compute local reachability for the current low-level state: lnb_reachability */
lnb_reachability($x$, $\Lambda^{\mathrm{hi}}(x^{\mathrm{hi}})$, $\Sigma_x^{\mathrm{hi}}$, $X_{\mathrm{done}}$)

/* Compute and investigate the outgoing events of x */
$\Sigma_x = \Lambda(x)$

while $\Sigma_x \neq \emptyset$

    pick $\sigma \in \Sigma_x$, set $\Sigma_x = \Sigma_x - \{\sigma\}$.
    /* Check if the event is a high-level event */
    if $\sigma \in \Sigma^{\mathrm{hi}}$
        $\Sigma_x^{\mathrm{hi}} = \Sigma_x^{\mathrm{hi}} \cup \sigma$

        /* Check if all high-level events are already reached */
        if $\Sigma_x^{\mathrm{hi}} = \Lambda^{\mathrm{hi}}(x^{\mathrm{hi}})$
            return(true)
        end if

    /* apply "lnb_reachability"on $\delta(x,\sigma)$ if that state has not been investigated, yet */
    else if $\delta(x,\sigma) \notin X_{\mathrm{done}}$
        $x' = \delta(x,\sigma)$
        $X_{\mathrm{done}} = X_{\mathrm{done}} \cup \{x'\}$
        if lnb_reachability$(x', \Lambda^{\mathrm{hi}}(x^{\mathrm{hi}}), \Sigma_x^{\mathrm{hi}}, X_{\mathrm{done}})$
            return(true)
        end if
    end if
end while

/* the locally nonblocking condition is not fulfilled */
return(false)
```

Figure 3.13: Realization of the function "lnb_reachability"

For determining the computational complexity of the algorithm in Figure 3.12, the following list provides the necessary information.

(i) Loop through all states in $x^{\mathrm{hi}} \in X^{\mathrm{hi}}$. The cardinality of the set is $|X^{\mathrm{hi}}| = n^{\mathrm{hi}}$.

(ii) Investigate all low-level states corresponding to x^{hi}. This set comprises the low-level states which are locally reachable from the entry states of x^{hi}. It is bounded by the number of low-level states $|X| = n$.

(iii) Perform the function "lnb_reachability" for all x corresponding to x^{hi}. The function is a reachability computation of states which are locally reachable from x. This set is also bounded by n.

Evaluating the above enumeration, the complexity of the algorithm is $\mathcal{O}(n^{\text{hi}}n^2)$. It has to be mentioned that the bound on the state sizes of the local automata is very conservative. In practical applications, the number of states which are locally reachable from some low-level state is considerably smaller than n.

3.4.4 Algorithmic Verification of Liveness

In addition to the system properties verified in Section 3.4.2 and 3.4.3, liveness of the high-level closed-loop language is needed for nonblocking control according to Theorem 3.1. Every state of the high-level supervised automaton has to be checked for outgoing events. The system is live if all these states have successor events, and the complexity for computing this result only depends on the number of states of the high-level supervised system. This number is bounded by $n^{\text{hi}}m^{\text{hi}}$, where m^{hi} is the number of states for the canonical recognizer of the high-level specification. Thus the complexity is $\mathcal{O}(n^{\text{hi}}m^{\text{hi}})$.

3.4.5 Algorithmic Verification of Marked String Controllability

In Theorem 3.2 the liveness condition is replaced by marked string controllability. For checking this property, local control systems as in Section 3.3 are needed. Local control systems were derived by computing the local behavior after some entry string of a low-level control system. Analogously, *local automata* are defined in the automata framework.

Definition 3.13 (Local Automaton)
Let G and G^{hi} be given as above and let $x_{\text{en}} \in X_{\text{en},x^{\text{hi}}}$ be an entry state for some $x^{\text{hi}} \in X^{\text{hi}}$. The local automaton $G_{x_{\text{en}},x^{\text{hi}}} = (\Sigma_{x_{\text{en}}}, X_{x_{\text{en}}}, \delta_{x_{\text{en}}}, x_{0,x_{\text{en}}}, X_{\text{m},x_{\text{en}}})$ is defined as

- $\Sigma_{x_{\text{en}}} := \Sigma$,
- $X_{x_{\text{en}}} := \{x \in X | \exists u \in (\Sigma - \Sigma^{\text{hi}})^* \text{ s.t. } x = \delta(x_{\text{en}}, u)\}$,
- $$\begin{aligned} \delta_{x_{\text{en}}}(x_{\text{en}}, \varepsilon) &:= x_{\text{en}} \\ \delta_{x_{\text{en}}}(x, \sigma) &:= \begin{cases} \delta(x, \sigma) & \text{if } \exists u' \in (\Sigma - \Sigma^{\text{hi}})^* \text{ s.t. } x = \delta(x_{\text{en}}, u) \text{ and } \sigma \in \Sigma\} \\ \text{not defined} & \text{otherwise} \end{cases} , \end{aligned}$$
- $x_{0,x_{\text{en}}} := x_{\text{en}}$,

- $X_{\mathrm{m},x_{\mathrm{en}}} := \{x \in X_{x_{\mathrm{en}}} | x \in X_{\mathrm{m}}\}$.

□

Local control systems capture the local behavior of a control system after some entry string. Analogously, local automata represent the local behavior of an automaton after some entry state. We establish the link between local control systems and local automata in Lemma 3.12, using the observation that an entry state in an automaton represents a set of entry strings in the corresponding control system (see Lemma 3.11).

Lemma 3.12 (Automata Realization of a Local Control System)
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a projected control system with the automata realization (G, G^{hi}). Also let $H_{s_{\mathrm{en}}}$ be a local control system for $s_{\mathrm{en}} \in L_{\mathrm{en},s^{\mathrm{hi}}}$ with $s^{\mathrm{hi}} \in L_1^{\mathrm{hi}}$. The local subautomaton $G_{x_{\mathrm{en}},x^{\mathrm{hi}}}$ for $x_{\mathrm{en}} := \delta(x_0, s_{\mathrm{en}})$ is an automata representation of $H_{s_{\mathrm{en}}}$, i.e. $L(G_{x_{\mathrm{en}},x^{\mathrm{hi}}}) = L_{s_{\mathrm{en}},1}$ and $L_{\mathrm{m}}(G_{x_{\mathrm{en}},x^{\mathrm{hi}}}) = L_{s_{\mathrm{en}},2}$. □

Proof: First it is shown that $L(G_{x_{\mathrm{en}},x^{\mathrm{hi}}}) \subseteq L_{s_{\mathrm{en}},1}$ and $L_{\mathrm{m}}(G_{x_{\mathrm{en}},x^{\mathrm{hi}}}) \subseteq L_{s_{\mathrm{en}},2}$. Assume that $u \in L(G_{x_{\mathrm{en}},x^{\mathrm{hi}}})$. Then, considering Definition 3.13, $u \in (\Sigma - \Sigma^{\mathrm{hi}})^*$. Also $s_{\mathrm{en}}u \in L(G)$ and because of Lemma 2.9, with $L(G) = L_1$, it follows that $s_{\mathrm{en}}u \in L_1$. Observing that $u \in (\Sigma - \Sigma^{\mathrm{hi}})^*$, Definition 3.4 states that $u \in L_{s_{\mathrm{en}},1}$.

Now let $u \in L_{\mathrm{m}}(G_{x_{\mathrm{en}},x^{\mathrm{hi}}})$. As $\delta_{x_{\mathrm{en}}}(x_{\mathrm{en}}, u) \in X_{\mathrm{m},x_{\mathrm{en}}}$, then also $\delta(x_0, s_{\mathrm{en}}u) \in X_{\mathrm{m}}$ by definition of $G_{x_{\mathrm{en}},x^{\mathrm{hi}}}$. Then, because of Lemma 2.9, $s_{\mathrm{en}}u \in L_2$ and with Definition 3.4 $u \in L_{s_{\mathrm{en}},2}$.

For proving the reverse direction it has to be shown that $L_{s_{\mathrm{en}},1} \subseteq L(G_{x_{\mathrm{en}},x^{\mathrm{hi}}})$ and $L_{s_{\mathrm{en}},2} \subseteq L_{\mathrm{m}}(G_{x_{\mathrm{en}},x^{\mathrm{hi}}})$. Assume that $u \in L_{s_{\mathrm{en}},1}$. Then $s_{\mathrm{en}}u \in L_1 \cap s_{\mathrm{en}}(\Sigma - \Sigma^{\mathrm{hi}})^* = L(G) \cap s_{\mathrm{en}}(\Sigma - \Sigma^{\mathrm{hi}})^*$. As $u \in (\Sigma - \Sigma^{\mathrm{hi}})^*$, Definition 3.13 states that $\delta_{x_{\mathrm{en}}}(x_{\mathrm{en}}, u)!$ as $\delta(x_0, s_{\mathrm{en}}u)!$. Hence, $u \in L(G_{x_{\mathrm{en}},x^{\mathrm{hi}}})$.

For showing $u \in L_{s_{\mathrm{en}},2}$, assume that $s_{\mathrm{en}}u \in L_2$ for $u \in (\Sigma - \Sigma^{\mathrm{hi}})^*$. Then $s_{\mathrm{en}}u \in L_{\mathrm{m}}(G)$ because of Lemma 2.9. But then $\delta_{x_{\mathrm{en}}}(x_{\mathrm{en}}, u)!$ and thus $u \in L_{\mathrm{m}}(G_{x_{\mathrm{en}},x^{\mathrm{hi}}})$. □

Consequently, it is true that any property which can be established for the local automaton at an entry state x_{en} is also valid for the local control systems of all corresponding entry strings s_{en} with $\delta(x_0, s_{\mathrm{en}}) = x_{\mathrm{en}}$.[15]

With the above result, marked string controllability can be checked by computing a controllability result for the local automata of each marked high-level state x^{hi} with a specification automaton $D_{x_{\mathrm{en}},x^{\mathrm{hi}}}$ which has the marked language $L_m(D_{x_{\mathrm{en}},x^{\mathrm{hi}}}) = L_m(G_{x_{\mathrm{en}},x^{\mathrm{hi}}})$ and the closed language $L(D_{x_{\mathrm{en}},x^{\mathrm{hi}}}) = \overline{L_m(G_{x_{\mathrm{en}},x^{\mathrm{hi}}})}$. (G, G^{hi}) is marked string controllable, if the supremal controllable sublanguage $\kappa_{L_{\mathrm{m}}(G_{x_{\mathrm{en}},x^{\mathrm{hi}}})}\big(L_{\mathrm{m}}(D_{x_{\mathrm{en}},x^{\mathrm{hi}}})\big)$ is nonempty for the local automata of all marked high-level states and corresponding entry states.

The number of marked high-level states is bounded by n^{hi} and the number of states of the local automata is bounded by n. Furthermore, the controllability computation (with complexity $\mathcal{O}(n^2m^2)$)

[15]In particular, the local control systems of these entry strings are all equivalent.

is carried out with a specification automaton with one state, i.e. $m = 1$. Thus, the complexity of the algorithm is $\mathcal{O}(n^{\text{hi}}n^2)$.

Combining these results, the condition relying on the liveness of the high-level controlled behavior (Section 3.3.1) can be verified in $\mathcal{O}(n^{\text{hi}}n^2)$, and the purely structural condition in Section 3.3.2 can also be computed in polynomial time with $\mathcal{O}(n^{\text{hi}}n^2)$.

The following detailed example illustrates the automata-based concepts explained in this section.

Example 3.5
The automaton model G used in this example is equivalent to the model in Example 3.4, except for marking of the states 5 and 9 and the transition a from state 5 to state 7. Analogously, the high-level event set $\Sigma^{\text{hi}} = \{\alpha, \beta, \gamma\}$ is used and $\Sigma_{\text{uc}} = \{\alpha, \mathtt{a}, \mathtt{d}\}$ and $\Sigma_{\text{c}} = \{\mathtt{b}, \mathtt{c}, \beta, \gamma\}$. The high-level and low-level automata are shown in Figure 3.14.

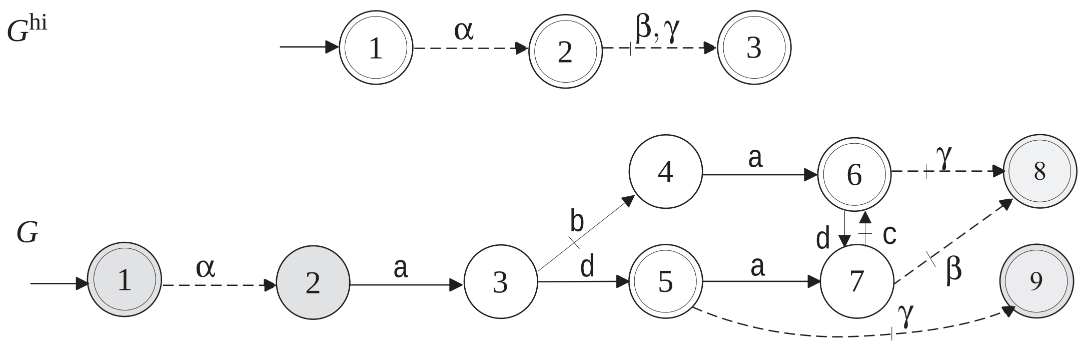


Figure 3.14: Low-level and high-level automata G and G^{hi}.

For verifying marked string acceptance and the locally nonblocking condition, a closer look at the local automaton $G_{2,2}$ is taken in Figure 3.15.[16] It is readily observed that from any local state of $G_{2,2}$, there is a local path to a state where any of the high-level events β or γ can occur. According to Definition 3.9, the locally nonblocking condition is fulfilled. Furthermore, all local paths from the entry state 2 to any of the exit states 5, 6 or 7 pass a marked state. Hence, with Definition 3.8, marked string acceptance is also true.

[16]The evaluation is trivial for the remaining local automata.

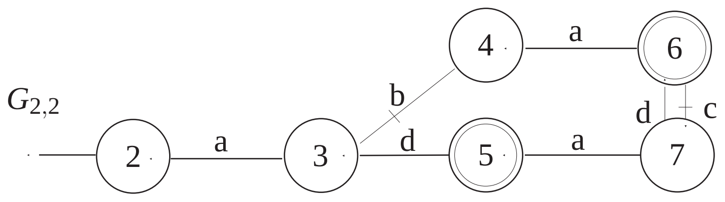


Figure 3.15: Local automaton $G_{2,2}$

It is readily observed that the high-level closed-loop system cannot be live as there are no cycles in the automaton graph. Thus, the structural condition in Theorem 3.2 has to be used. Checking for marked string controllability involves computing the supremal controllable sublanguage $\kappa_{L_m(G_{2,2})}\big(L_m(D_{2,2})\big)$ with $D_{2,2}$ as in Figure 3.16, which results in $\kappa_{L_m(G_{2,2})}\big(L_m(D_{2,2})\big) = aba(dc)^* + adac(dc)^* + ad$. It turns out, that this language is nonempty and thus the projected system is marked string controllable, too.

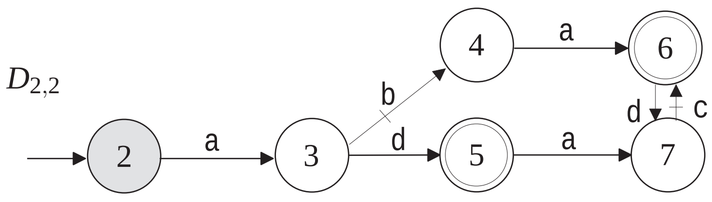


Figure 3.16: Specification automaton $D_{2,2}$ for marked string controllability in the high-level state 2

Considering the above results, it is possible to design a supervisor for the high-level model G^{hi} and translate it to the low level. As a high-level specification, it is desired that the event γ is disabled in the high-level state 2, i.e. $S^{hi}(2) = \{\alpha, \beta\}$. The resulting high-level closed-loop system $R^{hi}||G^{hi}$ with the automata realization R^{hi} of the supervisor S^{hi} is depicted in Figure 3.17.

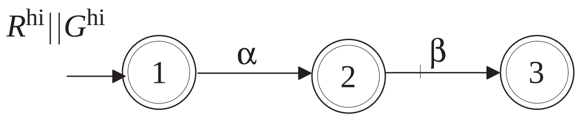


Figure 3.17: High-level supervised system

The supervisor implementing the low-level control is the same supervisor as shown in Equation 3.3. It disables the event γ in all states corresponding to the high-level state 2. The low-level closed-loop behavior is shown in Figure 3.18. Note that the corresponding hierarchical closed-loop system is hierarchically consistent as well as nonblocking.

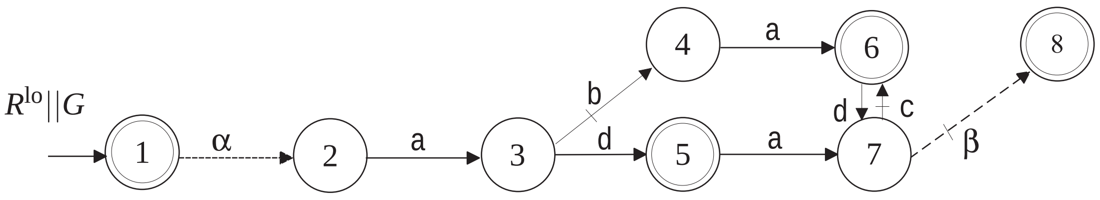


Figure 3.18: Low-level supervised system $R^{\mathrm{lo}}||G$

□

3.4.6 Evaluation of the Hierarchical Approach

We have shown that it is possible to verify all the conditions introduced in Section 3.3, i.e. the locally nonblocking condition, marked string acceptance, marked string controllability and liveness, algorithmically. The complete procedure for synthesizing a hierarchical supervisor is presented in the following list. G is the low-level system model.[17]

- Compute the projected automaton G^{hi} with $L_{\mathrm{m}}(G^{\mathrm{hi}}) = p^{\mathrm{hi}}(L_{\mathrm{m}}(G))$. For the systems under consideration, this computation can be done in polynomial time $\mathcal{O}(n^7(e^{\mathrm{hi}})^2)$ according to Section 3.4.1.
- Verify marked string acceptance. The complexity is $\mathcal{O}(n^{\mathrm{hi}}n^2)$.
- Check the locally nonblocking condition. This is done with complexity $\mathcal{O}(n^{\mathrm{hi}}n^2)$.
- Synthesize the high-level supervisor S^{hi} for a high-level specification automaton D^{hi} with m^{hi} states. The complexity is $\mathcal{O}((n^{\mathrm{hi}})^2(m^{\mathrm{hi}})^2)$.

Using the consistent implementation, nonblocking low-level control is guaranteed if either the high-level closed-loop system is live or the projected control system is marked string controllable.

[17] Recall that n is the number of states of G, e^{hi} is the number of high-level events, n^{hi} is the number of states of G^{hi} and m^{hi} is the number of states of a high-level specification automaton.

- Test for liveness. The complexity is $\mathcal{O}(n^{\mathrm{hi}}m^{\mathrm{hi}})$.
- The verification of marked string controllability is done in $\mathcal{O}(n^{\mathrm{hi}}n^2)$.

It is obvious that the complexity of the complete procedure is dominated by the natural projection. Evaluating the complexities of the subtasks, the overall complexity of the hierarchical approach is $\mathcal{O}\big(\max(n^7(e^{\mathrm{hi}})^2, (n^{\mathrm{hi}})^2(m^{\mathrm{hi}})^2)\big)$. The limiting factor is clearly the number of states of the low-level model. As pointed out in the beginning of this chapter, this number can be huge for composed systems.

However, composed systems have an inherent structure which is destroyed by the composition to the overall system. Because of this, it is worthwhile considering decentralized architectures where the low-level components need not be composed and thus the state explosion in the low-level model does not occur. The next chapter introduces a hierarchical and decentralized architecture which both makes use of this structural information and provides a method for applying the presented hierarchical control method in a decentralized setting.

Chapter 4

Hierarchical and Decentralized Control

As pointed out in the previous chapter, hierarchical architectures reduce the computational complexity of supervisor synthesis, by only taking into account the relevant behavior of the control system. The architecture in Chapter 3 guarantees that the automata representation of the high-level model always has less or equal states than the automata representation of the low-level model. It is pointed out that in applications, the number of states of the high-level model is always smaller. However, for computing the high-level model, the projection operation must be carried out for the low-level control system. To this end, a representation of the complete low-level automaton must be provided. Especially for composed systems, this automaton can have a very large number of states.

Addressing these issues, we provide an approach for the hierarchical and decentralized control of discrete event systems in this chapter. In particular, composed systems are investigated. They consist of several smaller components which have their own functionality. These smaller components interact to make up the behavior of the overall system. An instance of a composed system is presented in Chapter 5. It is important to be aware of the fact that the state sizes of the components multiply if they are put together to form the complete system. Therefore, it is highly desirable to preserve the decentralized structure for supervisor synthesis. This is indeed possible if the abstraction method and the consistent implementation of low-level supervisors outlined in the previous chapter is used. The decentralized low-level models can be projected to the high level, where they are composed to an overall high-level plant. Then, supervisory control is applied for the high-level model, and the resulting supervisor is implemented as a decentralized low-level supervisor for the subsystems. This hierarchical and decentralized architecture is based on the hierarchical architecture in Chapter 3 for monolithic systems. We extend these results to decentralized systems and provide sufficient conditions which guarantee nonblocking and hierarchically consistent behavior of the closed-loop system.

As in Chapter 3, we first elaborate a language-based description of the theoretical concepts, followed by an automata representation along with the algorithms which are necessary for implementation.

4.1 Hierarchical and Decentralized Control Architecture

An informal characterization of composed systems has been given above. The following definition formalizes this description in the form of *decentralized control systems*.

Definition 4.1 (Decentralized Control System)
A *decentralized control system* $||_{i=1}^{n} H_i$ (DCS) consists of subsystems, modeled by finite control systems $H_i = (L_{i,1}, L_{i,2})$, $i = 1, \dots, n$ over the respective alphabets Σ_i. The overall system is defined as $H := ||_{i=1}^{n} H_i =: (L_1, L_2)$ over the alphabet $\Sigma := \bigcup_{i=1}^{n} \Sigma_i$. The controllable and uncontrollable events are $\Sigma_{i,\mathrm{c}} := \Sigma_i \cap \Sigma_{\mathrm{c}}$ and $\Sigma_{i,\mathrm{uc}} := \Sigma_i \cap \Sigma_{\mathrm{uc}}$, respectively, where $\Sigma_{\mathrm{c}} \dot{\cup} \Sigma_{\mathrm{uc}} = \Sigma$. $\square$

A decentralized control system can be considered as a low-level model of a composed discrete event system as shown in Figure 4.1.

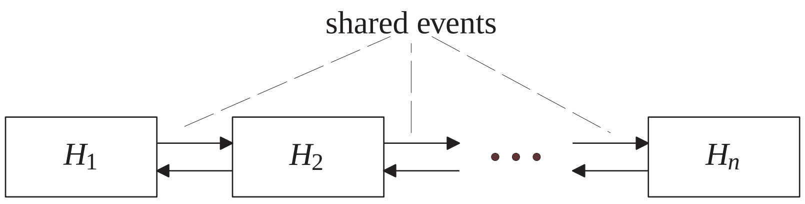


Figure 4.1: Decentralized control system

The different components of a DCS interact via shared events, i.e. events that are elements of the intersections of at least two alphabets. This interaction is crucial for the behavior of the overall system because the different components can block each other, which is clarified in the following example portrayed in Figure 4.2.

Example 4.1
The strings s_1, s_2 in $L_{1,2}$ and the string s_3 in $L_{2,2}$ contain the shared events α, β and γ. The strings s_1 and s_3 agree on the order of these events, such that the strings in $s_1||s_3$ in the synchronous composition $H_1||H_2$ are also elements of L_2 (i.e. they are marked strings of the control system $H = H_1||H_2$). However, the order of the high-level events is different for the strings s_2 and s_3. Thus, none of the strings in $s_2||s_3$ is a marked string in L_2. $\square$

This indicates that although the control systems H_1 and H_2 are nonblocking, the composed system $H_1||H_2$ can be blocking. If this is the case, then the decentralized components are *conflicting* (see also [RW87a, Won04, dQC00, QC00, LW02]).

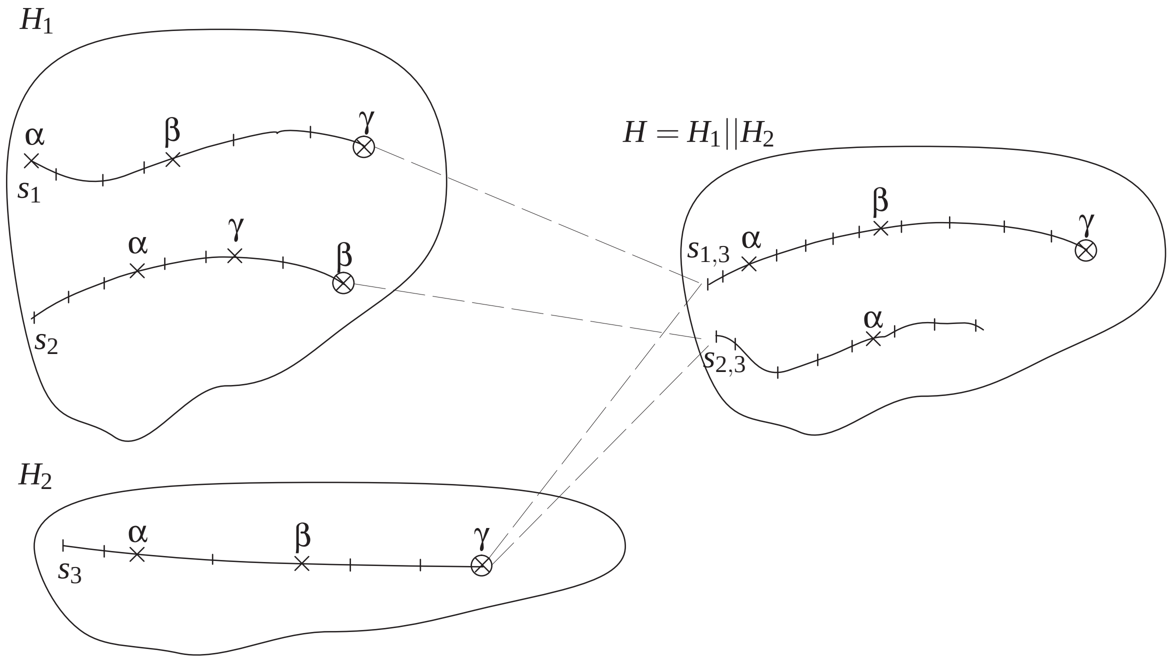


Figure 4.2: Blocking in a decentralized control system

This observation suggests that the abstraction of the decentralized components must always preserve the shared behavior and thus an abstraction alphabet must always contain the shared events. Accordingly, the high-level alphabet is chosen such that $\bigcup_{i,j=1,i\neq j}^{n} (\Sigma_i \cap \Sigma_j) \subseteq \Sigma^{\text{hi}} \subseteq \Sigma$. The abstraction of the DCS results in the *projected decentralized control system*.

Definition 4.2 (Projected Decentralized Control System)
Let $||_{i=1}^{n} H_i$ be a DCS, let Σ^{hi} s.t. $\bigcup_{i,j,i\neq j}^{n} (\Sigma_i \cap \Sigma_j) \subseteq \Sigma^{\text{hi}} \subseteq \Sigma$, $i = 1,\ldots,n$ and let $p^{\text{hi}} : \Sigma^* \to (\Sigma^{\text{hi}})^*$ be a natural projection. Also define the decentralized high-level alphabets as $\Sigma_i^{\text{hi}} := \Sigma^{\text{hi}} \cap \Sigma_i$ with the corresponding decentralized natural projections $p_i^{\text{dec}} : \Sigma_i^* \to (\Sigma_i^{\text{hi}})^*$ for $i = 1,\ldots,n$. A *projected decentralized control system* $(||_{i=1}^{n} H_i, p^{\text{hi}}, ||_{i=1}^{n} H_i^{\text{hi}})$ (PDCS) is composed of finite control systems $H_i^{\text{hi}} := p_i^{\text{dec}}(H_i)$, $i = 1,\ldots,n$. The overall high-level model is $H^{\text{hi}} = ||_{i=1}^{n} H_i^{\text{hi}}$. [1] High-level controllable and uncontrollable events are defined as $\Sigma_c^{\text{hi}} := \Sigma_{\text{c}} \cap \Sigma^{\text{hi}}$ and $\Sigma_{\text{uc}}^{\text{hi}} := \Sigma_{\text{uc}} \cap \Sigma^{\text{hi}}$, respectively. □

[1]For $H_1 = (L_{1,1}, L_{1,2})$ and $H_2 = (L_{2,1}, L_{2,2})$, the notation $H_1||H_2 = (L_{1,1}||L_{2,1}, L_{1,2}||L_{2,2})$ is introduced.

Figure 4.3 illustrates the concept of the projected decentralized control system with the decentralized control system $||_{i=1}^{n} H_i$ on the low level, the projections p_i^{dec}, $i = 1, \ldots, n$ and the projected decentralized high-level system $||_{i=1}^{n} H_i^{\mathrm{hi}}$.

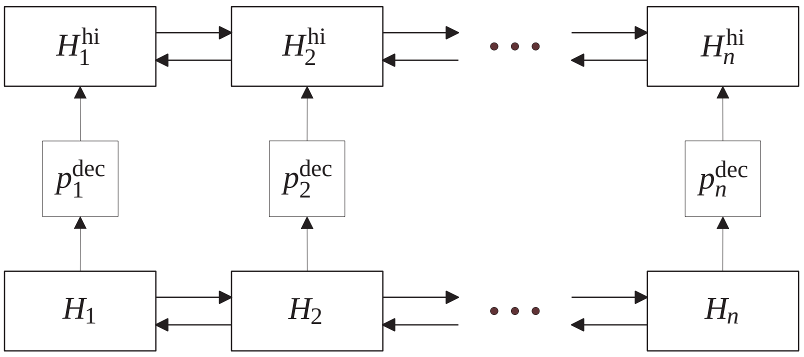


Figure 4.3: Projected decentralized control system

Definition 4.2 suggests the computation of the overall high-level model as $H^{\mathrm{hi}} = ||_{i=1}^{n} H_i^{\mathrm{hi}}$. Now, the question arises if this high-level model equals the model derived from projecting the overall low-level system, i.e. if $||_{i=1}^{n} p_i^{\mathrm{dec}}(H_i) = p^{\mathrm{hi}}(||_{i=1}^{n} H_i)$. For our particular choice of the high-level event set, this equality indeed holds, as established in our work in [SRM04, SMP05]. Proposition 4.1 states the respective result.

Proposition 4.1 (High Level Plant [SRM04, SMP05])
Let $(||_{i=1}^{n} H_i, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi}})$ be a projected decentralized control system. Then the high level control system is $H^{\mathrm{hi}} = p^{\mathrm{hi}}(||_{i=1}^{n} H_i) = ||_{i=1}^{n} p_i^{\mathrm{dec}}(H_i)$. □

The proof of Proposition 4.1 is based on a result in [Won04, dQ00]. It is given in Appendix A.3.

Proposition 4.1 provides an important result which reduces the computational complexity of the projection operation for decentralized systems tremendously. Now, it is no longer necessary to compute the overall low-level control system and then project it to the high level, but it is possible to project the decentralized subsystems to the high level first and then compose the projected systems to form the high-level control system.

Due to the fact that all shared events are contained in the high-level alphabet, the complete shared behavior is preserved in the high-level control system. It is possible that the feasible shared behavior of each subsystem is different from their independent behavior, i.e. it can happen that $p_i^{\mathrm{hi}}(L_1^{\mathrm{hi}}) \subset L_{i,1}^{\mathrm{hi}}$. This means that there are strings which are feasible in an independent system but which don't agree with the strings in the other systems. Furthermore, as already seen in Example 2.3, if a string is marked in one subsystem, a corresponding string in another subsystem need not be marked. This means that the string in the composed system is not marked, either.

The *feasible projected decentralized control system* of a decentralized control system represents its possible behavior after the synchronization.

Definition 4.3 (Feasible Projected Decentralized Control System)
Let $(||_{i=1}^{n} H_i, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi}})$ be a PDCS and let $p_i^{\mathrm{hi}} : (\Sigma^{\mathrm{hi}})^* \to (\Sigma_i^{\mathrm{hi}})^*$ be a natural projection.[2] The feasible projected decentralized control system (FPDCS) $(H_i^{\mathrm{f}}, p^{\mathrm{hi}}, H_i^{\mathrm{hi,f}})$, $i = 1, \ldots, n$, is defined as

(i) $H_i^{\mathrm{hi,f}} = (L_{i,1}^{\mathrm{hi,f}}, L_{i,2}^{\mathrm{hi,f}}) := p_i^{\mathrm{hi}}(H^{\mathrm{hi}})$

(ii) $H_i^{\mathrm{f}} := (L_{i,1}^{\mathrm{f}}, L_{i,2}^{\mathrm{f}})$ with $L_{i,1}^{\mathrm{f}} := \{s \in L_{i,1} | p_i^{\mathrm{dec}}(s) \in L_{i,1}^{\mathrm{hi,f}}\}$ and $L_{i,2}^{\mathrm{f}} := \{s \in L_{i,2} | p_i^{\mathrm{dec}}(s) \in L_{i,2}^{\mathrm{hi,f}}\}$.

$\square$

It is clear that the feasible projected decentralized control system for a projected decentralized control system exactly represents the possible shared behavior for the interacting decentralized control systems. We give a formal statement of this result in the following lemma.

Lemma 4.1
Let H_i, H_i^{f}, H_i^{hi} and $H_i^{\mathrm{hi,f}}$, $i = 1, \ldots, n$ be defined as in Definition 4.3. Then

$$||_{i=1}^{n} H_i^{\mathrm{hi}} = ||_{i=1}^{n} H_i^{\mathrm{hi,f}} \quad \text{and} \quad ||_{i=1}^{n} H_i = ||_{i=1}^{n} H_i^{\mathrm{f}}.$$

$\square$

The proof of Lemma 4.1 uses the fact that the composed system must agree with the shared behavior of the decentralized subsystems. It is provided in Appendix A.4.

Because of the above equivalence, the feasible projected subsystems $(||_{i=1}^{n} H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi,f}})$ are considered in the sequel, instead of the projected high-level subsystems $(||_{i=1}^{n} H_i, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi}})$.

After establishing the connection between the high level and the low level of a decentralized control system, supervisors for both the high level and the low level are added to the architecture. This leads to the formal definition of the *hierarchical and decentralized closed-loop system*.

Definition 4.4 (Hierarchical and Decentralized Closed Loop System [SPM05, SMP05])
A hierarchical and decentralized closed-loop system (HDCLS) $(||_{i=1}^{n} H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi,f}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ consists of the following entities

[2]The natural projections p^{hi} and p_i^{dec} are given in Definition 4.2.

- a FPDCS $(||_{i=1}^{n} H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi,f}})$ according to Definition 4.3,
- a high-level supervisor $S^{\mathrm{hi}}: L^{\mathrm{hi}} \to \Gamma^{\mathrm{hi}}$ with the high-level closed-loop control system $S^{\mathrm{hi}}/H^{\mathrm{hi}}$,
- a valid low-level supervisor $S^{\mathrm{lo}}: L_1 \to \Gamma$ s.t. $p^{\mathrm{hi}}(S^{\mathrm{lo}}/H) \subseteq S^{\mathrm{hi}}/H^{\mathrm{hi}}$.

$\square$

That is, the low-level model of a hierarchical and decentralized control system is a decentralized subsystem which is abstracted by projecting to a superset of the shared events of its components. Because of the decentralized nature of the system, the controllability properties of the low-level events are directly transferred to the high-level in this approach, i.e. a high-level event is controllable if it is controllable in the low-level, and it is uncontrollable if it is uncontrollable in the low level.[3] On the high level, standard supervisory control is applied, yielding the high-level supervisor. The translation of the high-level control action to the low level is considered to be valid if the low-level control achieves the desired behavior in the high level.

In this approach, the low-level control is defined by using a decentralized implementation of the high-level supervisor which is similar to the consistent implementation introduced in Section 3.2. Definition 4.5 introduces the *decentralized consistent implementation*.

Definition 4.5 (Decentralized Consistent Implementation)
Let $(||_{i=1}^{n} H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi,f}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ be a hierarchical and decentralized closed-loop system. Also implement

(i) decentralized high-level supervisors $S_i^{\mathrm{hi}} : (\Sigma_i^{\mathrm{hi}})^* \to \Gamma_i^{\mathrm{hi}}$ s.t. $S_i^{\mathrm{hi}}/H_i^{\mathrm{hi,f}} = p_i^{\mathrm{hi}}(S^{\mathrm{hi}}/H^{\mathrm{hi}})$,[4]

(ii) decentralized low-level supervisors $S_i^{\mathrm{lo}} : \Sigma_i^* \to \Gamma_i$ for the projected control systems $(H_i^{\mathrm{f}}, p_i^{\mathrm{dec}}, H_i^{\mathrm{hi,f}})$ as consistent implementations of S_i^{hi} for $i = 1, \dots, n$.

If the low-level supervisor $S^{\mathrm{lo}} : \Sigma^* \to \Gamma$ is defined s.t.

$$S^{\mathrm{lo}}/H = S^{\mathrm{hi}}/H^{\mathrm{hi}} || \left(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i^{\mathrm{f}} \right),$$

then $(||_{i=1}^{n} H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi,f}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ is called a HDCLS with a decentralized consistent implementation. $\square$

The HDCLS with a decentralized consistent implementation is illustrated in Figure 4.4.

[3]This choice of the high-level controllable and uncontrollable events is called control delay freedom in [Zho92, WW96].

[4]Conditions for the existence of such supervisors are given in the next lemma.

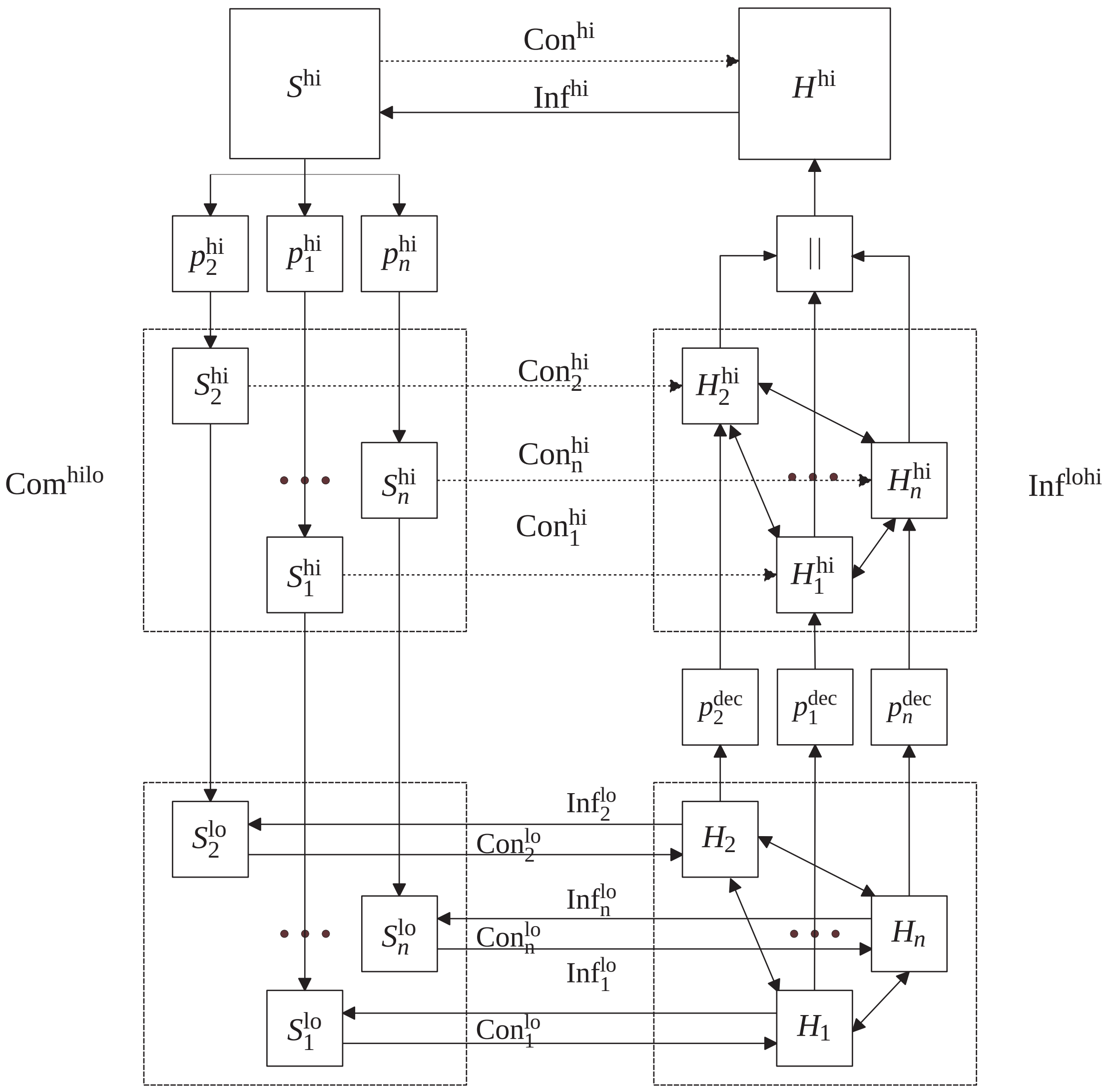


Figure 4.4: Hierarchical architecture

A decentralized high-level supervisor, implementing the projection of the high-level closed-loop behavior to the event set of the decentralized component, is computed for each of the decentralized projected control systems. The control action of this high-level supervisor is then translated to a decentralized low-level supervisor as a consistent implementation according to Definition 3.5. The joint action of the decentralized low-level supervisors S_i^{lo} and the high-level supervisor S^{hi} yields the overall low-level supervisor S^{lo}.

From Section 3.2, it is known how a consistent low-level supervisor can be computed if the corresponding high-level supervisor exists. However, the existence of the high-level supervisors in

Definition 4.5 is not automatically guaranteed. There need not be admissible high-level supervisors implementing the projection of the overall high-level closed-loop behavior to the event set of the respective decentralized subsystem. Yet, if the feasible projected control systems are required to be *mutually controllable*, then it is true that these supervisors always exist. Mutual controllability ensures that the decentralized subsystems agree on the control action to be executed. It was established by Lee and Wong in [LW02].

Lemma 4.2
Let $(||_{i=1}^{n} H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi,f}})$ be a feasible projected decentralized control system and let S^{hi} be a high-level supervisor for the overall high-level system H^{hi}. Also the natural projections $p_{i,j} : (\Sigma_i^{\mathrm{hi}})^* \to (\Sigma_i^{\mathrm{hi}} \cap \Sigma_j^{\mathrm{hi}})^*$ for $i, j = 1, \ldots, n$ are defined. If the high-level subsystems $H_i^{\mathrm{hi,f}}$, $i = 1, \ldots, n$ are mutually controllable, i.e. $\forall i, j = 1, \ldots, n$, $i \neq j$

$$L_{j,1}^{\mathrm{hi,f}}(\Sigma_{i,\mathrm{uc}} \cap \Sigma_{j,\mathrm{uc}}) \cap (p_{j,i})^{-1}\big(p_{i,j}(L_i^{\mathrm{hi,f}})\big) \subseteq L_j^{\mathrm{hi,f}},$$

then $p_i^{\mathrm{hi}}(L_1^{\mathrm{hi,c}})$ is controllable w.r.t. $L_{i,1}^{\mathrm{hi,f}}$ for all $i = 1, \ldots, n$. □

Lemma 4.2 is proven in Appendix A.5. A short illustration of the concept of mutual controllability is given in the subsequent example.

Example 4.2
Let L_1 and L_2 be two languages with the controllable shared event α and the uncontrollable shared event σ_{uc}. Also let $\mathtt{a} \in \Sigma_1$ and $\mathtt{b} \in \Sigma_2$ be non-shared events. Figure 4.5 illustrates some strings from L_1 and L_2.The string $s_2 \in L_2$ is chosen for verifying mutual controllability. It holds that $s_2\mathtt{b}\sigma_{\mathrm{uc}} \in L_2\Sigma_{\mathrm{uc}}$ and $s_2\mathtt{b}\sigma_{\mathrm{uc}} \in (p_{2,1})^{-1}(p_{1,2}(L_1))$ as $p_{1,2}(s_1) = p_{2,1}(s_2) = \alpha\alpha$. Thus, $s_2\mathtt{b}\sigma_{\mathrm{uc}} \in L_2 \cap (p_{2,1})^{-1}(p_{1,2}(L_1))$ but $s_2\mathtt{b}\sigma_{\mathrm{uc}} \notin L_2$. This indicates that mutual controllability fails for the example.

The problem is that the uncontrollable event σ_{uc} can either happen or not after different low-level strings (e.g. s_2 and $s_2\mathtt{b}$) which are perceived as the same string from the other system. (L_1 just perceives the shared string $\alpha\alpha$). Mutual controllability prevents this ambiguity. □

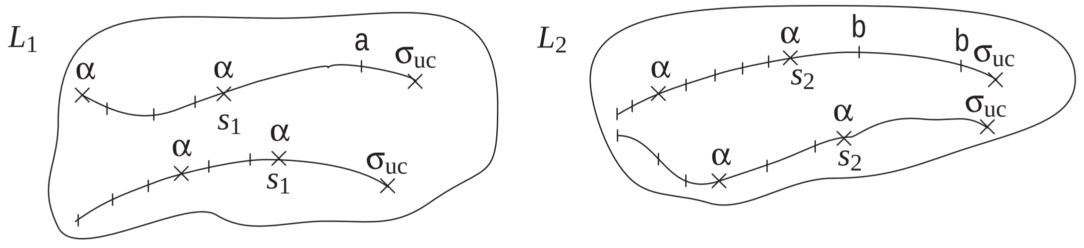


Figure 4.5: Illustration of mutual controllability

To sum up, Lemma 4.2 ensures that the projection of the high-level closed-loop behavior to the decentralized event sets is controllable w.r.t. the respective feasible high-level language. Consequently, the decentralized high-level supervisors in Definition 4.5 can be implemented. The application of the hierarchical and decentralized architecture is shown in the following example.

Example 4.3
Let H_1 and H_2 be control systems with the languages $L_{1,1} = \overline{\alpha\mathtt{b}(\alpha + \mathtt{c}\alpha)}$, $L_{1,2} = \alpha\mathtt{b} + \alpha\mathtt{b}\alpha + \alpha\mathtt{bc}\alpha$ and $L_{2,1} = \overline{\alpha\mathtt{d}\alpha}$, $L_{2,2} = \alpha\mathtt{d} + \alpha\mathtt{d}\alpha$, respectively. The shared event α and the event $\mathtt{c}$ are controllable and for the high-level projection $H^{\text{hi}} = p^{\text{hi}}(H_1)||p^{\text{hi}}(H_2)$, the languages are $L_1^{\text{hi}} = \overline{\alpha\alpha}$ and $L_2^{\text{hi}} = \alpha$. Note that both decentralized projected control systems $(H_1, p^{\text{hi}}, H_1^{\text{hi}})$ and $(H_2, p^{\text{hi}}, H_2^{\text{hi}})$ are feasible as $p_1(H^{\text{hi}}) = H_1^{\text{hi}}$ and $p_2(H^{\text{hi}}) = H_2^{\text{hi}}$. The resulting hierarchical and decentralized structure is denoted in Figure 4.6. The high-level supervisor S^{hi} is defined such that $S^{\text{hi}}(\alpha) = \emptyset$ and $S^{\text{hi}}(s^{\text{hi}}) = \{\alpha\}$ for all other $s^{\text{hi}} \in L_1^{\text{hi}}$. Mutual controllability of the high-level subsystems can be verified. As the two high-level systems H_1^{hi} and H_2^{hi} have the same event set as H^{hi}, the corresponding decentralized high-level supervisors are equal to S^{hi}.[5] Applying the consistent implementation, the supervisors S_1^{lo} and S_2^{lo} are

$$S_1^{\text{lo}}(s_1) = \begin{cases} \{\mathtt{b}\} & \text{if } s_1 \in \alpha\mathtt{b} \\ \{\alpha, \mathtt{b}, \mathtt{c}\} & \text{else} \end{cases} \quad S_2^{\text{lo}}(s_2) = \begin{cases} \{\mathtt{d}\} & \text{if } s_2 = \alpha\mathtt{d} \\ \{\alpha, \mathtt{d}\} & \text{else} \end{cases} \tag{4.1}$$

The resulting low-level closed-loop behaviors are $S_1^{\text{lo}}/H_1 = (\overline{\alpha\mathtt{b}}, \alpha\mathtt{b})$ and $S_2^{\text{lo}}/H_2 = (\overline{\alpha\mathtt{d}}, \alpha\mathtt{d})$. □

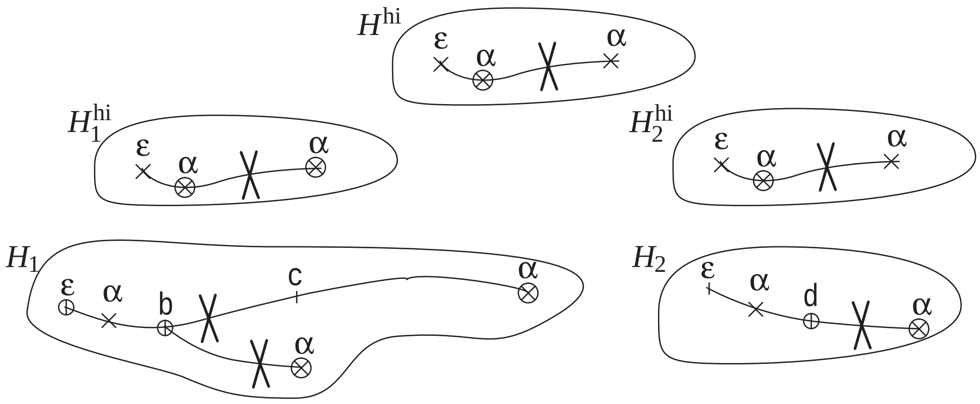


Figure 4.6: Illustration of the decentralized supervisor implementation

[5] Note that in this case, the projections p_1^{hi} and p_2^{hi} are the identity map.

4.2 Hierarchical Consistency

The decentralized consistent implementation is based on the consistent implementation in Section 3.2. As this low-level supervisor realization is sufficient for hierarchical consistency of a HCLS, a similar result is expected for the decentralized case. Proposition 4.2 confirms this conjecture.

Proposition 4.2 (Hierarchical Consistency)
Let Let $(||_{i=1}^{n} H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi,f}})$ be a feasible projected decentralized control system with a high-level supervisor S^{hi} and the high-level closed-loop system $S^{\mathrm{hi}}/H^{\mathrm{hi}} = (L_1^{\mathrm{hi,c}}, L_2^{\mathrm{hi,c}})$. If S^{lo} is a decentralized consistent implementation of S^{hi} and the feasible high-level subsystems $H_i^{\mathrm{hi,f}}$ are mutually controllable, then $(||_{i=1}^{n} H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi,f}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ is a hierarchically consistent HDCLS. $\square$

Proof: Considering Lemma 4.2, it holds for all $i = 1, \dots, n$ that $p_i^{\mathrm{hi}}(L_1^{\mathrm{hi,c}})$ is controllable w.r.t. $L_{i,1}^{\mathrm{f}}$. Thus, for all i, there exists a $S_i^{\mathrm{hi}} : (\Sigma_i^{\mathrm{hi}})^* \to \Gamma_i^{\mathrm{hi}}$ s.t. $S_i^{\mathrm{hi}}/H_i^{\mathrm{hi,f}} = p_i^{\mathrm{hi}}(S^{\mathrm{hi}}/H^{\mathrm{hi}})$.[6] Using consistent implementations S_i^{lo} for the high-level supervisors S_i^{hi} and the projected control systems $(H_i^{\mathrm{f}}, p_i^{\mathrm{dec}}, H_i^{\mathrm{hi,f}})$, the hierarchical control systems $(H_i^{\mathrm{f}}, p_i^{\mathrm{dec}}, H_i^{\mathrm{hi,f}}, S_i^{\mathrm{hi}}, S_i^{\mathrm{lo}})$ are hierarchically consistent because of Proposition 3.1, and hence $p_i^{\mathrm{dec}}(S_i^{\mathrm{lo}}/H_i^{\mathrm{f}}) = S_i^{\mathrm{hi}}/H_i^{\mathrm{hi,f}} = p_i^{\mathrm{hi}}(S^{\mathrm{hi}}/H^{\mathrm{hi}})$.

Implementing the low-level supervisor as in Definition 4.5 results in $p^{\mathrm{hi}}(S^{\mathrm{lo}}/H) = p^{\mathrm{hi}}\big(S^{\mathrm{hi}}/H^{\mathrm{hi}}||(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i)\big) = p^{\mathrm{hi}}(S^{\mathrm{hi}}/H^{\mathrm{hi}})||p^{\mathrm{hi}}(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i) = S^{\mathrm{hi}}/H^{\mathrm{hi}}||p_i^{\mathrm{dec}}(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i)$ because of Lemma 4.1. Also with Lemma 4.1 and with the above observation, it is true that $p^{\mathrm{hi}}(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i^{\mathrm{f}}) = ||_{i=1}^{n} p_i^{\mathrm{dec}}(S_i^{\mathrm{lo}}/H_i^{\mathrm{f}}) = ||_{i=1}^{n} p_i^{\mathrm{hi}}(S^{\mathrm{hi}}/H^{\mathrm{hi}})$. Noting that $S^{\mathrm{hi}}/H^{\mathrm{hi}} \subseteq ||_{i=1}^{n} p_i^{\mathrm{hi}}(S^{\mathrm{hi}}/H^{\mathrm{hi}})$ (Lemma A.6), it holds that $S^{\mathrm{hi}}/H^{\mathrm{hi}}||p_i^{\mathrm{dec}}(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i) = S^{\mathrm{hi}}/H^{\mathrm{hi}}||(||_{i=1}^{n} p_i^{\mathrm{hi}}(S^{\mathrm{hi}}/H^{\mathrm{hi}})) = S^{\mathrm{hi}}/H^{\mathrm{hi}}$. Consequently, the hierarchical and decentralized control system is hierarchically consistent. $\square$

The above result makes use of the fact that the joint behavior of the decentralized high-level supervisors S_i^{hi} results in consistent behavior on the high-level, i.e. the high-level controlled decentralized systems combined with the overall high-level supervisor yield the same behavior as the overall high-level plant. Further on, the fact that the low-level control of the decentralized projected control systems is achieved by consistent implementations makes sure that the high-level closed-loop behavior of the decentralized subsystems can be implemented by low-level supervisors.

Example 4.4
The HDCLS in Example 4.3 is hierarchically consistent. There is a corresponding string in the low-level closed-loop behavior for all strings in the high-level closed-loop behavior. $\square$

After showing that the behavior implemented by the low-level supervisor is hierarchically consistent, it has also to be verified if the specified language can really be implemented. The subsequent lemma states the required result.

[6]Also observe that $\overline{p_i^{\mathrm{hi}}(L_2^{\mathrm{hi,c}})} = p_i^{\mathrm{hi}}(\overline{L_2^{\mathrm{hi,c}}}) = p_i^{\mathrm{hi}}(L_1^{\mathrm{hi,c}})$ as $S^{\mathrm{hi}}/H^{\mathrm{hi}}$ is nonblocking.

Lemma 4.3
Let $(||_{i=1}^{n} H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi,f}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ be a HDCLS with a decentralized consistent implementation. Then $S^{\mathrm{hi}}/H^{\mathrm{hi}}||\big(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i^{\mathrm{f}}\big)$ is controllable w.r.t. L_1. $\square$

Proof: Assume that $S^{\mathrm{hi}}/H^{\mathrm{hi}}||\big(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i^{\mathrm{f}}\big)$ is not controllable w.r.t. L_1. Then it holds that $\exists s \in L_1 \cap S^{\mathrm{hi}}/H^{\mathrm{hi}}||\big(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i^{\mathrm{f}}\big)$, $\sigma \in \Sigma_{\mathrm{uc}}$ s.t. $s\sigma \in L_1$ but $s\sigma \notin S^{\mathrm{hi}}/H^{\mathrm{hi}}||\big(||_{i=1}^{n} S_i^{\mathrm{lo}}/H_i^{\mathrm{f}}\big)$. Let $\sigma \in \Sigma^{\mathrm{hi}}$. Then $p^{\mathrm{hi}}(s\sigma) \in L_1^{\mathrm{hi}}$ and $p^{\mathrm{hi}}(s\sigma) \notin p^{\mathrm{hi}}(L_1^{\mathrm{c}}) = L_1^{\mathrm{hi,c}}$ because of Proposition 4.2. As $p^{\mathrm{hi}}(s\sigma) \in L_1^{\mathrm{hi}}$, this means that $L^{\mathrm{hi,c}}$ is not controllable w.r.t. L^{hi} which leads to contradicion.

Now assume $\sigma \in \Sigma - \Sigma^{\mathrm{hi}}$. Then $\exists i$ such that $\sigma \in \Sigma_i$ and i is unique (otherwise $\sigma \in \Sigma^{\mathrm{hi}}$). Then $p_i(s\sigma) \in L_{i,1}$ and $p_i(s\sigma) \notin L_{i,1}^{\mathrm{f,c}}$ (otherwise $s\sigma \in L_1^{\mathrm{c}}$). Hence $L_{i,1}^{\mathrm{f,c}}$ is not controllable w.r.t. $L_{i,1}^{\mathrm{f}}$ which contradicts the admissibility of S_i^{lo}. $\square$

4.3 Nonblocking Control

Looking at the decentralized consistent implementation in Definition 4.5, it is readily observed that the supervised system consists of decentralized hierarchical closed-loop systems. The *locally nonblocking* and *marked string acceptance* conditions are needed to guarantee that the hierarchical closed-loop systems are nonblocking. Consequently, these conditions are also required for feasible decentralized projected control systems.

In addition to the above mentioned properties, two different requirements were formulated in Section 3.3. Analogous to the approach in Chapter 3, two alternative versions of the main theorem are elaborated in the sequel. The first version provides a condition on the high-level closed-loop subsystems, and the second condition requires marked string controllability as a further structural condition for the feasible decentralized projected control systems.

4.3.1 Condition on the High-Level Closed Loop Subsystems

The main theorem of this section is similar to Theorem 3.1. Instead of requiring the overall feasible projected control system to be locally nonblocking and marked string accepting, only the decentralized components need to fulfill these conditions. The decentralized consistent implementation ensures hierarchical consistency and nonblocking behavior of the overall hierarchical and decentralized closed-loop system in combination with liveness of the decentralized high-level closed-loop systems.

Theorem 4.1 (Main Result)
Let $(\|_{i=1}^n H_i^{\mathrm{f}}, p^{\mathrm{hi}}, \|_{i=1}^n H_i^{\mathrm{hi,f}})$ be a feasible decentralized projected control system[7] with a nonblocking high-level supervisor S^{hi}. Assume that all projected control systems $(H_i^{\mathrm{f}}, p_i^{\mathrm{dec}}, H_i^{\mathrm{hi,f}})$, $i = 1,\dots,n$ are marked string accepting, locally nonblocking and that the decentralized high-level languages $L_{i,1}^{\mathrm{hi,f}}$ are mutually controllable. Also let S^{lo} be a decentralized consistent implementation of S^{hi}. If all languages $L_{i,1}^{\mathrm{hi,f,c}}$ are live, then the HDCLS $(\|_{i=1}^n H_i^{\mathrm{f}}, p^{\mathrm{hi}}, \|_{i=1}^n H_i^{\mathrm{hi,f}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ is nonblocking and hierarchically consistent. $\square$

The subsequent lemma provides a property of feasible projected decentralized control systems with live high-level subsystems. It says that any high-level string in the overall behavior can be extended to a marked high-level string such that the extension contains symbols from all alphabets in a specified set of alphabets.

Lemma 4.4
Let $(\|_{i=1}^n H_i, p^{\mathrm{hi}}, \|_{i=1}^n H_i^{\mathrm{hi}})$, $i = 1,\dots,n$ be a projected decentralized system with a nonblocking high-level control system H^{hi} and assume that $L_{i,1}^{\mathrm{hi}}$ is live for $i = 1,\dots,n$. Also let $\mathcal{I}$ be an index set with $\mathcal{I} = \{i_1,\dots,i_m\} \subseteq \{1,\dots,n\}$. Then for all $s^{\mathrm{hi}} \in L_1^{\mathrm{hi}}$, there exists a $t \in (\Sigma^{\mathrm{hi}})^*$ s.t. $s^{\mathrm{hi}}t \in L_2^{\mathrm{hi}}$ and for all $j \in \mathcal{I}$, $p_j^{\mathrm{hi}}(t) \neq \varepsilon$. $\square$

Proof: The following algorithm for constructing a suitable string t is proposed for proving Lemma 4.4. Assume $s^{\mathrm{hi}} \in L_1^{\mathrm{hi}}$ and $\mathcal{I}$ are given.

1. $k := 1$, $\tilde{\mathcal{I}} = \mathcal{I}$
2. choose $i_k \in \tilde{\mathcal{I}}$
3. find $t_k \in (\Sigma^{\mathrm{hi}})^*$ s.t. $s^{\mathrm{hi}}t_1 \cdots t_k \in L_2^{\mathrm{hi}}$ and $p_{i_k}^{\mathrm{hi}}(t_k) \neq \varepsilon$
4. remove all j with $p_j^{\mathrm{hi}}(t_k) \neq \varepsilon$ from $\tilde{\mathcal{I}}$
5. **if** $\tilde{\mathcal{I}} = \emptyset$, set $k^* := k$ and **terminate**
   else $k := k+1$ and **go to** 2.

First note that t_k as in item 3. exists for each k. Observing that $p_{i_k}^{\mathrm{hi}}(s^{\mathrm{hi}}t_1\dots t_{k-1}) \in L_{i_k,1}^{\mathrm{hi}}$ and with $L_{i_k,1}^{\mathrm{hi}}$ being live, there exists a $\sigma_{i_k} \in \Sigma_{i_i}$ s.t. $p_{i_k}^{\mathrm{hi}}(s^{\mathrm{hi}}t_1\dots t_{k-1})\sigma_{i_k} \in L_{i_k,1}^{\mathrm{hi}}$. As $L_{i_k,1}^{\mathrm{hi}} = p_{i_k}^{\mathrm{hi}}(L_1^{\mathrm{hi}})$, there is a $\hat{t}_k$ s.t. $s^{\mathrm{hi}}t_1\dots t_{k-1}\hat{t}_k \in L_1^{\mathrm{hi}}$ and $p_{i_k}^{\mathrm{hi}}(\hat{t}_k) = \sigma_{i_k}$. But as H^{hi} is nonblocking, there exists a $\tilde{t}_k$ such that $s^{\mathrm{hi}}t_1\dots t_{k-1}\hat{t}_k\tilde{t}_k \in L_2^{\mathrm{hi}}$. Thus, $t_k := \hat{t}_k\tilde{t}_k$ fulfills the condition in item 3. of the algorithm. Secondly, observe that the algorithm terminates as the set $\mathcal{I}$ is finite and in each loop through the algorithm at least one element is removed from $\mathcal{I}$. After termination of the algorithm, the string $t = t_1 \cdots t_{k^*}$ fulfills Lemma 4.4. It holds that $s^{\mathrm{hi}}t \in L_2^{\mathrm{hi}}$ and $p_i^{\mathrm{hi}}(t) \neq \varepsilon$ for all $i \in \mathcal{I}$ by construction of t. $\square$

[7]Note that the theorem can also be stated for the original decentralized projected control system $(\|_{i=1}^n H_i, p^{\mathrm{hi}}, \|_{i=1}^n H_i^{\mathrm{hi}})$. However, as this DPCS contains redundant behavior, conditions would be more restrictive.

The proof of Theorem 4.1 is supported by Lemma 4.4.

Proof: Hierarchical consistency is provided by Proposition 4.2.

The strings $s \in L_1^c$ are considered for showing nonblocking behavior. Note that $s^{\mathrm{hi}} := p^{\mathrm{hi}}(s) \in L_1^{\mathrm{hi,c}}$, $s_i^{\mathrm{hi}} := p_i^{\mathrm{hi}}(s^{\mathrm{hi}}) \in p_i^{\mathrm{hi}}(L_1^{\mathrm{hi,c}})$ and $s_i := p_i(s) \in p_i(L_1^c)$. Now let $\mathcal{I} := \{i \in \{1,\ldots,n\} | \nexists u_i \in (\Sigma_i - \Sigma^{\mathrm{hi}})^*$ s.t. $s_i u_i \in L_{i,2}^{\mathrm{f,c}}\}$. Because of Lemma 4.4, it is possible to find a $t \in (\Sigma^{\mathrm{hi}})^*$ with $s^{\mathrm{hi}} t \in L_1^{\mathrm{hi,c}}$ s.t. $\forall i \in \mathcal{I}$, $p_i^{\mathrm{hi}}(t) \neq \varepsilon$.

Applying Lemma 3.7, $(S_i^{\mathrm{lo}}/H_i, p^{\mathrm{hi}}, S_i^{\mathrm{hi}}/H_i^{\mathrm{hi}})$ is a locally nonblocking projected control system for all $i = 1,\ldots,n$. With this observation, Lemma 3.8 states that for all i with $p_i^{\mathrm{hi}}(t) \neq \varepsilon$, there exists $u_i \in \Sigma_i^*$ s.t. $s_i u_i \in L_{i,1}^{\mathrm{f,c}} \cap L_{\mathrm{en}, s_i^{\mathrm{hi}} p_i(t)}$. Furthermore, because of Lemma 3.9, there is a $\hat{u}_i \in (\Sigma_i - \Sigma^{\mathrm{hi}})^*$ s.t. $s_i u_i \hat{u}_i \in L_{i,2}^{\mathrm{f,c}}$ for i with $p_i^{\mathrm{hi}}(t) \neq \varepsilon$.

For i with $p_i^{\mathrm{hi}}(t) = \varepsilon$, define $u_i = \varepsilon$ and note that there also exists $\hat{u}_i \in (\Sigma_i - \Sigma^{\mathrm{hi}})^*$ s.t. $s_i u_i \hat{u}_i \in L_{i,2}^{\mathrm{f,c}}$ as $i \notin \mathcal{I}$.

Consequently, for any $u \in ||_{i=1}^n u_i \hat{u}_i$ it holds that $su \in ||_{i=1}^n L_{i,1}^{\mathrm{f,c}}$. Just as well, $p_i(su) \in L_{i,2}^{\mathrm{f,c}}$, for all $i = 1,\ldots,n$ and $p^{\mathrm{hi}}(su) = s^{\mathrm{hi}} t \in L_2^{\mathrm{hi,c}}$. Thus $su \in L_2^{\mathrm{hi,c}} || (||_{i=1}^n L_{i,2}^{\mathrm{f,c}})$ which means $s \in \overline{L_2^{\mathrm{hi,c}} || (||_{i=1}^n L_{i,2}^{\mathrm{f,c}})}$. Thus, $(||_{i=1}^n H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^n H_i^{\mathrm{hi,f}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ is nonblocking. □

Looking at Theorem 4.1, it turns out that the conditions required for nonblocking control are very similar to the monolithic case. The reason for this is that the decentralized consistent implementation is chosen for the low-level supervisor. It uses the concept of a consistent implementation for each decentralized high-level supervisor. In this framework, the decentralized high-level supervisors exist, because the high-level languages are mutually controllable. Together, the existence of the high-level supervisors and the decentralized consistent implementation guarantee hierarchically consistent and nonblocking control.

4.3.2 Structural Condition

The main result in the previous section is dependent on the fact that the high-level closed-loop subsystems are live, that is a condition on the closed-loop system is imposed. In the sequel, this condition is replaced by a structural condition which covers the case that liveness is not given for the high-level closed-loop subsystems, i.e. there are high-level strings which cannot be extended further. The second condition in the consistent implementation in Definition 3.5 covers this case if marked string controllability is required.

Theorem 4.2 (Main Result)
Let $(||_{i=1}^n H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^n H_i^{\mathrm{hi,f}})$ be a feasible decentralized projected control system with a nonblocking high-level supervisor S^{hi}. Assume that all projected control systems $(H_i^{\mathrm{f}}, p_i^{\mathrm{dec}}, H_i^{\mathrm{hi,f}})$, $i = 1,\ldots,n$

are marked string accepting and locally nonblocking. Also let the decentralized high-level languages $L_{i,1}^{\mathrm{hi,f}}$ be mutually controllable and let S^{lo} be a decentralized consistent implementation of S^{hi}. If all projected systems $(H_i^{\mathrm{f}}, p_i^{\mathrm{dec}}, H_i^{\mathrm{hi,f}})$, $i = 1, \dots, n$ are marked string controllable, then the HDCLS $(||_{i=1}^n H_i^{\mathrm{f}}, p^{\mathrm{hi}}, ||_{i=1}^n H_i^{\mathrm{hi,f}}, S^{\mathrm{hi}}, S^{\mathrm{lo}})$ is nonblocking and hierarchically consistent. $\square$

Proof: Hierarchical consistency follows from Proposition 4.2.

For proving nonblocking behavior, first note that $L_1^c \neq \emptyset$ as $L_1^{\mathrm{hi,c}} \neq \emptyset$ and $p^{\mathrm{hi}}(L_1^c) = L_1^{\mathrm{hi,c}}$. Now assume that $s \in L_1^c$ and $s^{\mathrm{hi}} = p^{\mathrm{hi}}(s) \in L_1^{\mathrm{hi,c}}$. It has to be shown that $s \in \overline{L_2^c}$.

Because of Definition 4.5, $s \in L_1^{\mathrm{hi,c}} || \left(||_{i=1}^n L_{i,1}^{\mathrm{f,c}} \right)$. Thus $s_i := p_i(s) \in L_{i,1}^{\mathrm{f,c}}$ and $s_i^{\mathrm{hi}} := p_i^{\mathrm{dec}}(s_i) \in L_{i,1}^{\mathrm{hi,f,c}}$. Let $\mathfrak{I} := \{i | 1 \leq i \leq n \wedge \nexists u_i \in (\Sigma_i - \Sigma^{\mathrm{hi}})^* \text{ s.t. } s_i u_i \in L_{i,2}^{\mathrm{f,c}}\}$. The following algorithm is performed to find an appropriate string leading to a marked sring in the high-level.

1. $k = 1, \tilde{\mathfrak{I}} = \mathfrak{I}$.
2. choose $i_k \in \tilde{\mathfrak{I}}$.
3. find $t_k \in (\Sigma^{\mathrm{hi}})^*$ s.t. $s^{\mathrm{hi}} t_1 \cdots t_k \in L_2^{\mathrm{hi,c}}$ and $p_{i_k}^{\mathrm{hi}}(t_k) \neq \varepsilon$.
4. remove all j with $p_j^{\mathrm{hi}}(t_k) \neq \varepsilon$ from $\tilde{\mathfrak{I}}$.
5. **if** $\tilde{\mathfrak{I}} = \emptyset$: set $k^* := k$ and **terminate**
   else $k := k + 1$ and **go to** 2.

First note that the string t_k in 3. always exists. There are two possible cases. In case that $\Sigma_i^{\mathrm{hi}}(s_i^{\mathrm{hi}}) \cap S_i^{\mathrm{hi}}(s_i^{\mathrm{hi}}) = \emptyset$ for some i, it holds that $i \notin \mathfrak{I}$, as there must be $u_i \in (\Sigma_i - \Sigma^{\mathrm{hi}})^*$ s.t. $s_i u_i \in L_{i,2}^{\mathrm{f,c}}$ because $H_i^{\mathrm{f,c}}$ is nonblocking according to Theorem 3.2. Thus, for all $i \in \mathfrak{I}$, it holds that $\Sigma_i^{\mathrm{hi}}(s_i^{\mathrm{hi}}) \cap S_i^{\mathrm{hi}}(s_i^{\mathrm{hi}}) \neq \emptyset$. For this case, there exists a $t_{i_k} \neq \varepsilon$ s.t. $p_{i_k}^{\mathrm{hi}}(s^{\mathrm{hi}} t_1 \dots t_{k-1}) t_{i_k} \in L_{i,2}^{\mathrm{hi,f,c}}$ because $H_i^{\mathrm{hi,f,c}}$ is nonblocking. With $L_{i,2}^{\mathrm{hi,f,c}} = p_i^{\mathrm{hi}}(L_2^{\mathrm{hi,c}})$ it is readily observed that $\exists t_k \in (\Sigma^{\mathrm{hi}})^*$ with $p_i^{\mathrm{hi}}(t_k) = t_{i_k} \neq \varepsilon$ and $s^{\mathrm{hi}} t_1 \dots t_{k-1} t_k \in L_2^{\mathrm{hi,c}}$.

Secondly, note that the algorithm terminates as $\mathfrak{I}$ is a finite index set which is reduced in every step.

Hence, the above algorithm provides a high-level string $t := t_1 \cdots t_{k^*}$ s.t. $s^{\mathrm{hi}} t \in L_2^{\mathrm{hi,c}}$ and $p_i^{\mathrm{hi}}(t) \neq \emptyset$ for all $i \in \mathfrak{I}$. It holds that $s_i^{\mathrm{hi}} t_i := p_i^{\mathrm{hi}}(s^{\mathrm{hi}} t) \in L_{i,1}^{\mathrm{hi,f,c}}$ as $L_{i,1}^{\mathrm{hi,c}} = p_i^{\mathrm{hi}}(L_1^{\mathrm{hi,c}})$. Then, because of Lemma 3.8, $\forall i \in \mathfrak{I}$, $\exists u_i \in \Sigma_i^*$ s.t. $s_i u_i \in L_{i,1}^{\mathrm{f,c}} \cap L_{\mathrm{en}, s_i^{\mathrm{hi}} t_i}$. Further on, because of Lemma 3.9, $\exists \hat{u}_i \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ s.t. $s_i u_i \hat{u}_i \in L_{i,2}^{\mathrm{c}}$.

For $i \notin \mathfrak{I}$, define $u_i := \varepsilon$ and note that there is a $\hat{u}_i \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ s.t. $s_i u_i \hat{u}_i \in L_{i,2}^{\mathrm{f,c}}$ by definition of $\mathfrak{I}$. Then $\forall u \in ||_{i=1}^n u_i \hat{u}_i$, it holds that $su \in ||_{i=1}^n s_i u_i \hat{u}_i \subseteq ||_{i=1}^n L_{i,2}^{\mathrm{f,c}}$ and $p^{\mathrm{hi}}(su) = s^{\mathrm{hi}} t \in L_2^{\mathrm{hi,c}}$. Thus $su \in L_2^{\mathrm{hi,c}} || \left(||_{i=1}^n L_{i,2}^{\mathrm{f,c}} \right) = L_2^{\mathrm{c}}$, which proves that $s \in \overline{L_2^{\mathrm{c}}}$. $\square$

The proof of Theorem 4.2 is very similar to the proof of Theorem 4.1. The difference comes in, if high-level strings do not have any extension in the respective controlled high-level subsystem. Then, analogous to Section 3.3, the combination of the consistent implementation and the marked string controllability of the feasible decentralized projected systems results in nonblocking behavior of the hierarchical and decentralized control system.

4.4 Automata Implementation for the Decentralized Case

After elaborating the theoretical concepts of the hierarchical and decentralized control method, an algorithmic implementation of the presented results is given in the automata framework.

4.4.1 Notation

At first, the notation used for evaluating the computational complexity of the algorithms presented in this section is introduced.

- G^{hi}: number of states: n^{hi}.
- D^{hi}: number of states: m^{hi}.
- G_i: bound on the number of states: n_i.
- G_i^{hi}: bound on the number of states: n_i^{hi} and bound on the number of events: e_i^{hi}.
- G_i^{f}: bound on the number of states: n_i^{f}.
- $G_i^{\mathrm{hi,f}}$: bound on the number of states: $n_i^{\mathrm{hi,f}}$.

4.4.2 Feasible Projected Decentralized Control Systems

According to Definition 4.1, a decentralized control system consists of finite local control systems H_i. With Corollary 3.1, these control systems can be represented as finite automata $G_i = (\Sigma, X_i, \delta_i, x_{0,i}, X_{\mathrm{m},i})$ s.t. $L(G_i) = L_{i,1}$ and $L_m(G_i) = L_{i,2}$, and also the overall control system $H = ||_{i=1}^n H_i$ corresponds to the finite automaton $G = ||_{i=1}^n G_i$ with $L(G) = L_1$ and $L_m(G) = L_2$. Thus the automata representation of a decentralized control system $||_{i=1}^n H_i$ is given by $||_{i=1}^n G_i$.

Applying the natural projection to the high-level events, also projected decentralized control systems $(||_{i=1}^n H_i, p^{\mathrm{hi}}, ||_{i=1}^n H_i^{\mathrm{hi}})$ can be formulated as a set of finite automata, with the automata representations G_i^{hi} of the high-level control systems H_i^{hi}. The algorithm providing this result is shown in the following figure.

```
/* Compute a projected decentralized control system: compute_pdcs */
compute_pdcs($G_i, \dots, G_n$, $\Sigma^{\text{hi}}$)

/* project all automata to their high-level event set */
for($1 \leq i \leq n$)
    $\Sigma_i^{\text{hi}} = \Sigma^{\text{hi}} \cap \Sigma_i$
    $(G_i^{\text{hi}}, f_i^{\text{hilo}})$ = projection($G_i$,$\Sigma_i^{\text{hi}}$)
end for

return($G_1^{\text{hi}}, \dots, G_n^{\text{hi}}, f_1^{\text{hilo}}, \dots, f_n^{\text{hilo}}$)
```

Figure 4.7: Computation of a projected decentralized control system

The natural projection has to be executed for all n control systems of the decentralized control system $||_{i=1}^{n} G_i$. Considering Section 3.4.1, the complexity of the algorithm is $\mathcal{O}(n\, n_i^7 (e_i^{\text{hi}})^2)$.

With the finite automata representation $(||_{i=1}^{n} G_i, p^{\text{hi}}, ||_{i=1}^{n} G_i^{\text{hi}})$ of a projected decentralized control system $(||_{i=1}^{n} H_i, p^{\text{hi}}, ||_{i=1}^{n} H_i^{\text{hi}})$, it is possible to compute the high-level plant G^{hi}. According to Lemma 4.1, the high-level subautomata have to be composed, i.e. $G^{\text{hi}} = ||_{i=1}^{n} G_i^{\text{hi}}$. This is done in the subsequent algorithm.

```
/* Compute the high-level automaton: compute_ghi */
compute_ghi($G_1^{\text{hi}}, \dots, G_n^{\text{hi}}$)

/* Initialize $G^{\text{hi}}$ with the first decentralized high-level automaton */
$G^{\text{hi}} = G_1^{\text{hi}}$

/* Loop through all high-level subautomata and compute the synchronous composition */
for($2 \leq i \leq n$)
    $G^{\text{hi}} = G^{\text{hi}} || G_i^{\text{hi}}$
end for
return($G^{\text{hi}}$)
```

Figure 4.8: Computation of the high-level automaton

The algorithm evaluates the synchronous composition for all n high-level subautomata. Recalling the complexity of the synchronous composition, the computational effort for the algorithm is $\mathcal{O}((n_i^{\text{hi}})^n)$. Here, the positive effect of Lemma 4.1 can be recognized. Tthe computational effort is exponential in the number of states of the high-level components of the decentralized control system instead of being exponential in the number of states of the low-level subautomata. As the number of states of the high-level subautomata is expected to be smaller than the number of states of the low-level automata, (Theorem 3.3), this denotes a considerable computational gain.

The feasible projected decentralized control system can be determined according to Definition 4.3 with the projected decentralized control system $(||_{i=1}^{n} H_i, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi}})$ and the overall high-level control system H^{hi}. The following Lemma states how this is done in an automata representation.

Lemma 4.5
Let (G_i, G_i^{hi}) be an automata representation of a projected control system $(H_i, p^{\mathrm{hi}}, H_i^{\mathrm{hi}})$ and let G^{hi} be the automata representation of the high-level control system H^{hi}. Then $G_i^{\mathrm{hi,f}}$ is an automata representation of $p_i^{\mathrm{hi}}(H^{\mathrm{hi}})$, and the automaton for H_i^{f} is

$$G_i^{\mathrm{f}} = G_i^{\mathrm{hi,f}} || G_i.$$

$\square$

Proof: $p_i^{\mathrm{hi}}(H^{\mathrm{hi}})$ is a control system because of Lemma 3.1. With Corollary 3.1, $G_i^{\mathrm{hi,f}}$ is the minimal recognizer of $p_i^{\mathrm{hi}}(H^{\mathrm{hi}})$.

Let $s \in L(G_i^{\mathrm{hi,f}} || G_i)$. Then $p^{\mathrm{hi}}(s) \in L(G_i^{\mathrm{hi,f}})$ and $s \in L(G_i)$. Thus, $s \in L(G_i^{\mathrm{f}})$ according to Definition 4.3. Now let $s \in L(G_i^{\mathrm{f}})$. Then $s \in L(G_i)$ and $p^{\mathrm{hi}}(s) \in L(G_i^{\mathrm{hi,f}})$. This means that $s \in L(G_i^{\mathrm{hi,f}} || G_i)$. $\square$

It is sufficient to evaluate the projection of the high-level automaton G^{hi} to the event sets $(\Sigma^{\mathrm{hi}} \cap \Sigma_i)$ for $i = 1, \ldots, n$ and to compose the high-level feasible control systems $G_i^{\mathrm{hi,f}}$ with the low-level systems G_i to compute the automata representation $(G_i^{\mathrm{f}}, G_i^{\mathrm{hi,f}})$. The algorithm in Figure 4.9 illustrates this procedure.

```
/* Compute the feasible projected decentralized control system: compute_fpdcs */
compute_fpdcs($G_1, \ldots, G_n, G^{\mathrm{hi}}, f_1^{\mathrm{hilo,f}}, \ldots, f_n^{\mathrm{hilo,f}}$)

/* Project $G^{\mathrm{hi}}$ on the different alphabets */
for($1 \le i \le n$)
    $G_i^{\mathrm{hi,f}} = \mathrm{projection}\big(G^{\mathrm{hi}}, (\Sigma^{\mathrm{hi}} \cap \Sigma_i)\big)$
    $G_i^{\mathrm{f}} = G_i^{\mathrm{hi,f}} || G_i$
    $f_i^{\mathrm{hilo,f}} = \mathrm{f}_{\mathrm{hilo}}(G_i^{\mathrm{f}}, f_i^{\mathrm{hilo}})$
end for
return($G_1^{\mathrm{hi,f}}, \ldots, G_n^{\mathrm{hi,f}}, G_1^{\mathrm{f}}, \ldots, G_n^{\mathrm{f}}, f_1^{\mathrm{hilo,f}}, \ldots, f_n^{\mathrm{hilo,f}}$)
```

Figure 4.9: Computation of the feasible projected decentralized control system

The function $\mathrm{f}_{\mathrm{i}}^{\mathrm{hilo,f}}$ determines the new mapping of high-level states to entry states. From f_i^{hilo}, it gets the information about the entry states in G_i. Every state in the synchronous composition $G_i||G_i^{\mathrm{hi,f}}$ is associated to the corresponding high-level state in $G_i^{\mathrm{hi,f}}$ by inspecting the state name pairs $(x_i, x_i^{\mathrm{hi,f}})$ in X_i^{f}.

The natural projection of the high-level automaton G^{hi} has to be carried out n times to compute the feasible projected decentralized control system. The complexity of this computation is $\mathcal{O}(n\,(n^{\mathrm{hi}})^7(e_i^{\mathrm{hi}})^2)$. The subsequent synchronous composition is done with $\mathcal{O}(n\,n_i\,n_i^{\mathrm{hi,f}})$.

Combining the above algorithms, determining the feasible projected decentralized control system from a decentralized control system and a high-level alphabet is of complexity $\mathcal{O}\big(\max(n\,n_i^7(e_i^{\mathrm{hi}})^2, (n_i^{\mathrm{hi}})^n)\big)$.

4.4.3 Marked String Acceptance and Locally Nonblocking Condition

Marked string acceptance as well as the locally nonblocking condition have to be checked for all feasible projected control systems $(H_i^{\mathrm{f}}, p^{\mathrm{hi}}, H_i^{\mathrm{hi,f}})$ by examining the corresponding automata representations $(G_i^{\mathrm{f}}, G_i^{\mathrm{hi,f}})$. For verifying marked string acceptance, the algorithm in Figure 3.10 has to be carried out n times, while the locally nonblocking condition is checked by applying the algorithm in Figure 3.12 n times. Consequently, the complexity of both computations is $\mathcal{O}(n\,(n_i^{\mathrm{f}})^2 n_i^{\mathrm{hi,f}})$.

4.4.4 Liveness and Marked String Controllability

Similar to the previous section, liveness and marked string controllability have to be evaluated for each feasible projected decentralized control system. For checking liveness, it has to be verified if every state of the natural projections of the high-level closed-loop behavior to the decentralized alphabets has successor events. This can be done by checking if any entry state (for the respective decentralized alphabet) has a path to a successor event. The complexity is $\mathcal{O}(((n_i^{\mathrm{hi,f}})^n m^{\mathrm{hi}})^2\, n)$, where $(n_i^{\mathrm{hi,f}})^n m^{\mathrm{hi}}$ is a bound for the number of states of the high-level closed-loop automaton. The complexity for verifying marked string controllability is just multiplied by n, i.e. $\mathcal{O}(n\,n_i^2 n_i^{\mathrm{hi}})$ in the decentralized case.

4.4.5 Supervisor Computation

All conditions for the hierarchical and decentralized approach presented in this chapter can be verified computationally with the above algorithms. The complete procedure for synthesizing a hierarchical and decentralized supervisor for a decentralized control system with a given high-level alphabet is presented in the following list.

- Compute the feasible projected decentralized automata representations $(G_i^{\mathrm{f}}, G^{\mathrm{hi},\mathrm{f}_i})$. For the systems under consideration, this computation can be done in polynomial time $\mathcal{O}(n\, n_i^7 (e_i^{\mathrm{hi}})^2)$ according to Section 4.4.2.
- Verify marked string acceptance. The complexity is $\mathcal{O}(n\, n_i^{\mathrm{hi,f}} (n_i^{\mathrm{f}})^2)$.
- Check the locally nonblocking condition. This is done with complexity $\mathcal{O}(n\, n_i^{\mathrm{hi,f}} (n_i^{\mathrm{f}})^2)$.
- Synthesize the high-level supervisor S^{hi} for a high-level specification automaton D^{hi} with m^{hi} states. The complexity is $\mathcal{O}((n_i^{\mathrm{hi,f}})^{2n} (m^{\mathrm{hi}})^2)$.

Nonblocking low-level control is guaranteed if either the decentralized high-level closed-loop systems are live or the feasible decentralized projected control systems are marked string controllable, using the decentralized consistent implementation.

- Test for liveness. The complexity is $\mathcal{O}((n_i^{\mathrm{hi,f}})^{2n} (m^{\mathrm{hi}})^2\, n)$.
- Verify marked string controllability. The complexity is $\mathcal{O}(n\, n_i^{\mathrm{hi,f}} (n^{\mathrm{f}})^2)$.

Putting together the complexities of the different algorithms needed in this approach, it turns out that the overall complexity for verification of the structural properties and synthesis of the high-level supervisor is $\mathcal{O}(\max(n\, n_i^7 (e_i^{\mathrm{hi}})^2, (n_i^{\mathrm{hi,f}})^{2n} (m^{\mathrm{hi}})^2\, n))$. Thus, the main contributions to the computational effort are the natural projection of the decentralized components to their respective high-level alphabets ($\mathcal{O}(n\, n_i^7 (e_i^{\mathrm{hi}})^2)$) and the composition of the decentralized high-level subsystems to the overall high-level system in combination with the check for liveness ($\mathcal{O}((n_i^{\mathrm{hi,f}})^{2n} (m^{\mathrm{hi}})^2\, n)$). Considering the fact that the state size of the abstracted systems is supposed to be smaller than the state size of the original low-level models, the computational gain is evident (from $\mathcal{O}(n_i^n)$ to $\mathcal{O}((n_i^{\mathrm{hi}})^n)$, where $n_i^{\mathrm{hi}} << n_i$).

The fact, that the computational complexity of the approach is still exponential in the number of states of the decentralized high-level components, is not surprising, if the result in [RL02] is considered. It is stated that "the verification of large discrete event systems modeled as interacting sets of finite automata will not lead to computationally tractable results unless we make more assumptions about the models themselves".

These additional assumptions are exactly what is exploited in our method. The structural properties of *marked string acceptance* and *locally nonblocking DES* guarantee that systems can be projected to models with a smaller number of states in the high level.

In addition to the reduced computational effort, it is also worth mentioning, that the implementation of the decentralized supervisors is manageable. Small decentralized supervisors which can directly be implemented for their respective component are designed instead of synthesizing one low-level supervisor implementation for the overall system. A detailed description of the procedure with an extension to a multi-level hierarchy is given in Chapter 5.

Chapter 5

Manufacturing System Case Study

An example for a large-scale composed discrete event system is the Fischertechnik Simulation Model of the "Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg", that represents a distributed manufacturing system (see Figure 5.1). The system comprises 28 components, and a monolithic automaton model reaches an estimated number of 10^{24} states.[1] Due to the size of the manufacturing system, monolithic supervisor synthesis is not advisable.

Nevertheless, the decentralized nature of the system suggests the applicability of the hierarchical and decentralized approach in Chapter 4. To this end, the example system is divided into structural components, and 4 hierarchical levels are used.

The chapter is organized as follows. At first, a thorough description of the the example system (called the "manufacturing system"in the sequel) is given in Section 5.1. Section 5.3 focuses on the supervisor synthesis for a part of the manufacturing system. For the *distribution system*, which embodies the entrance area of the system, the hierarchical abstraction and decentralized supervisor design are worked out in detail. Finally, the complete system is considered, and the performance of the method is evaluated for a representative supervisor computation in Section 5.4.

A bird's eye view of the manufacturing system is shown in Figure 5.1. It consists of a stack feeder, conveyor belts, pushers, rotary tables, production cells and a rail transport system. A schematic overview is given in Figure 5.2. The purpose of the manufacturing system is to process workpieces (symbolized by wooden blocks) which enter the system from a stack feeder (sf). From there, the workpieces are distributed by the long conveyor belt (cb1). There are two pushers pu1 and pu2 attached to this conveyor belt, that transport workpieces to the actual production part of the system. Also there is a reject depot (dep) at the end of the long conveyor belt.

[1]For implementation reasons, control systems will be represented by the corresponding finite automata throughout the whole chapter.

5.1 Manufacturing System Overview

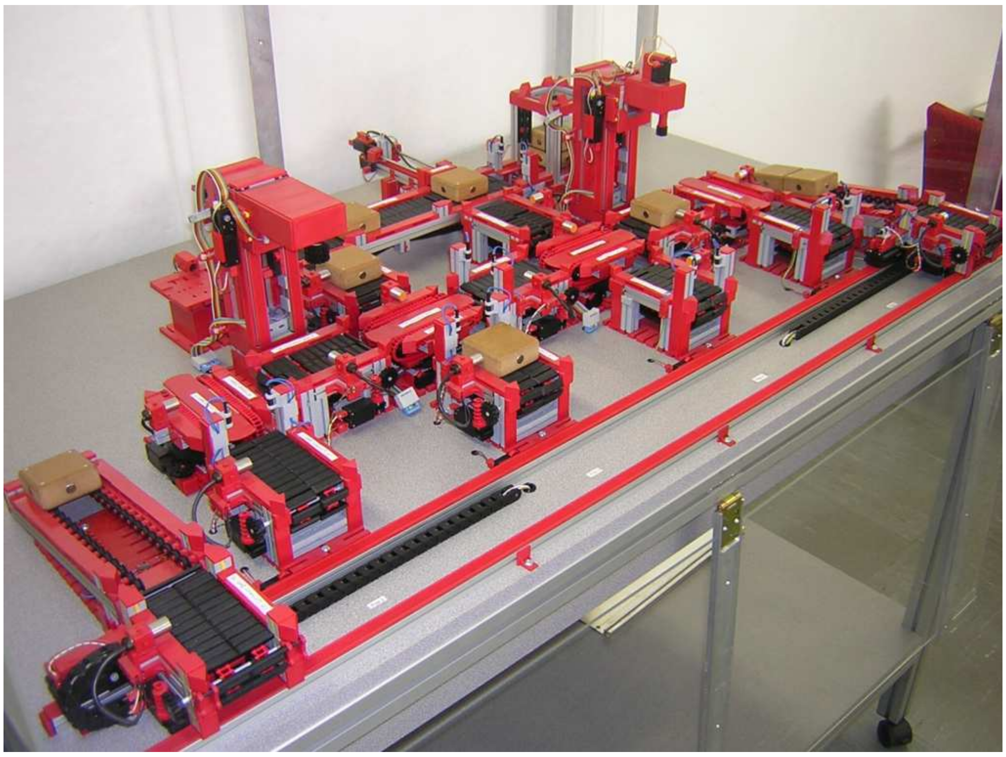


Figure 5.1: Fischertechnik simulation model

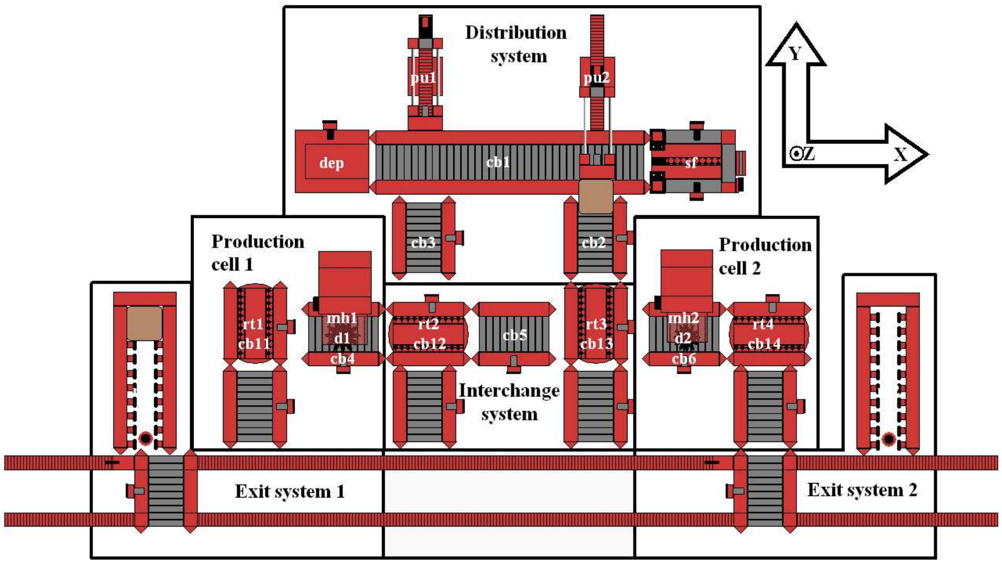


Figure 5.2: Schematic overview of the Fischertechnik simulation model

The production part is entered via two conveyor belts (cb2 and cb3) which serve as buffers for workpieces waiting for processing in one of the production cells (mh1, mh2). The rotary tables rt2 and rt3 (with conveyor belts cb12 and cb13) move workpieces to the respective machine (mh1 and mh2), where they are drilled (d1 and d2). From the machines, workpieces either maneuver back by using the rotary tables rt2 or rt3 again or move forward via rt1 or rt4 (with conveyor belts cb11 and cb14).

For leaving the manufacturing system, the conveyor belts cb7, cb8, cb9 and cb10 deliver workpieces to the rail transport systems rts1 and rts2, which can unload to the roll conveyors rc1 and rc2. Each of the roll conveyors is able to store up to four workpieces. The rail transport system rts1 can serve the conveyor belts cb7, cb8 and cb9, while rts2 can deliver workpieces to the conveyor belts cb8, cb9 and cb10.

The manufacturing system is equipped with different sensors which detect logical signals[2]. Also, the motors driving the conveyor belts, rotary tables, machines, etc. only assume discrete values, i.e. moving into one of two directions or stopping. In this setting, sensor signals can be imagined as uncontrollable events (there is no direct influence on the occurrence of the signal), while actuator signals are considered as controllable events (these signals can directly be set or reset). Using these event definitions, the behavior of the manufacturing system can be described as sequences of actuator and sensor signals, i.e. sequences of controllable and uncontrollable events. Regarding the definition of a discrete event system in Chapter 2, the manufacturing system belongs to this class of systems.

Furthermore, it is readily observed, that the manufacturing system is composed of different components which interact[3]. It is possible to apply the hierarchical and decentralized method developed in Chapter 4 for supervisory control of the manufacturing system. At first, the design and synthesis procedure is performed for the distribution system for facility of inspection. It comprises the stack feeder sf, the long conveyor belt cb1, the two pushers pu1 and pu2, the deposit dep and the conveyor belts cb2 and cb3. The same approach is then used for the rest of the system, and performance details are discussed in Section 5.4.

5.2 Notation

The following notation is used for modeling system components on different hierarchical levels.

- plant automata are written $G_j^{(i)}$, where i denotes the level of the hierarchy where the automaton model resides and j is the name of the component, according to the schematic overview in Figure 5.2.

[2]A detailed description will be given in Section 5.3.

[3]For example consider one conveyor belt transporting a workpiece to another conveyor belt

- if $\Sigma_j^{(i)}$ is the alphabet of $G_j^{(i)}$ and $\Sigma_j^{(i+1)} \subseteq \Sigma_j^{(i)}$ is an abstraction alphabet, then the corresponding natural projection is $p_j^{(i,i+1)} : (\Sigma_j^{(i)})^* \rightarrow (\Sigma_j^{(i+1)})^*$.
- assume that one component $G_j^{(i)}$ of the system is composed of smaller subcomponents $G_{j_k}^{(i)}$, $k = 1,\ldots,n$ and $j_k \neq j$, i.e. $G_j^{(i)} = ||_{k=1}^{n} G_{j_k}^{(i)}$. Then, the alphabet of $G_j^{(i)}$ is $\Sigma_j^{(i)} := \bigcup_{k=1}^{n} \Sigma_{j_k}^{(i)}$.
- let $G_j^{(i)} = ||_{k=1}^{n} G_{j_k}^{(i)}$ be as above and let $\Sigma_j^{(i+1)} \supseteq \bigcup_{k,l=1;k\neq l}^{n} (\Sigma_{j_k}^{(i)} \cap \Sigma_{j_l}^{(i)})$ be an abstraction alphabet for the level $i+1$. Then the natural projection for the subsystems is defined as $p_{j_k}^{(i,i+1)} : (\Sigma_{j_k}^{(i)})^* \rightarrow (\Sigma_j^{(i+1)} \cap \Sigma_{j_k}^{(i)})^*$.
- specification automata for an automaton model $G_j^{(i)}$ are written as $D_j^{(i)}$.
- the automaton realizing the supervised behavior $\kappa_{L_m(D_j^{(i)})}\big(L_m(G_j^{(i)})\big)$ is written as $R_j^{(i)}$.
- event names indicate the component where the event occurs and the action which is related to the event. For example, the event `sfwpar` means that a workpiece arrives at the stack feeder.
- for better orientation, a coordinate system is defined in Figure 5.2.
- in the automata graphs, dashed arrows indicate high-level transitions, and a tick on an arrow denotes a controllable event.

5.3 Supervisor Synthesis for the Distribution System

The distribution system as shown in Figure 5.3 has several components. The stack feeder, the depot and the two conveyor belts cb2 and cb3 are modeled as single components, whereas the long conveyor belt cb1 with the pushers pu1 and pu2 is split into three parts for modeling convenience. The first part (cb1apu2) describes cb1 between the stack feeder and the pusher pu2. The second part (cb1bpu1) models cb1 from the pusher pu1 to the depot, and the third part (cb1$_{\text{connect}}$) accounts for the physical connection between cb1apu2 and cb1bpu1 via the belt. In the following sections, the hierarchical architecture for the distributionsystem with 4 levels is constructed, starting from the lowest (sensor and actuator) level. The automaton models of the system components were derived in [Ers02, Per04].

5.3.1 Stack Feeder

The stack feeder consists of a stack which can hold maximally four workpieces and a belt with a small block which can shove workpieces to the conveyor belt cb1. The belt's motion and end of

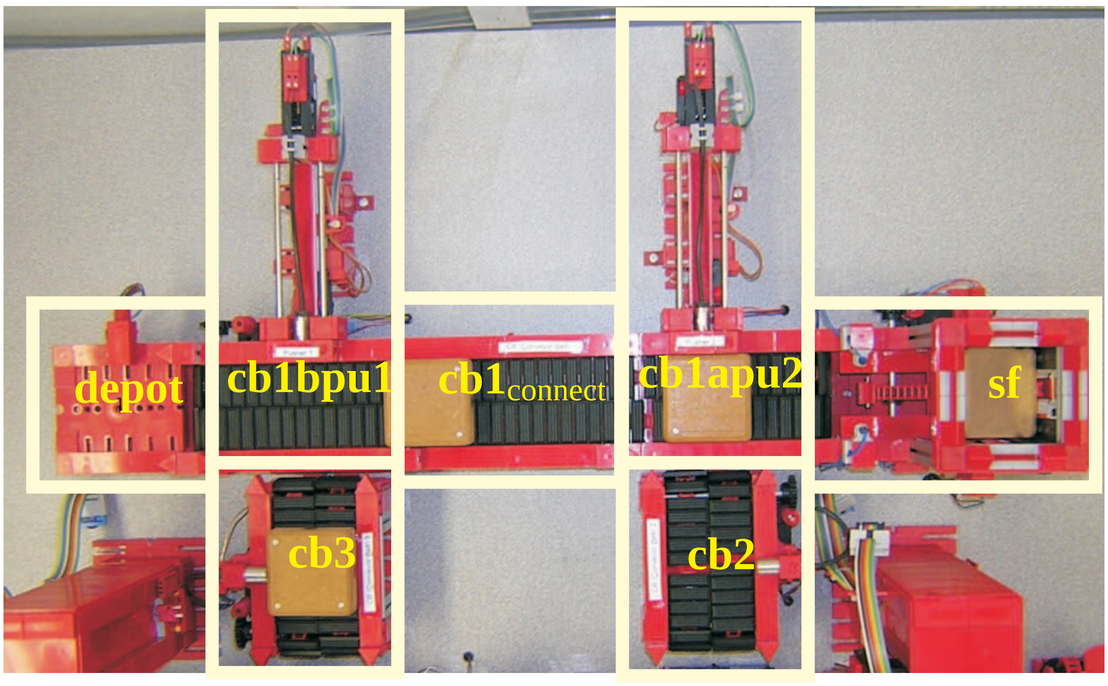


Figure 5.3: Components of the distributionsystem

motion is triggered by the events `sfmv` and `sfstp`, respectively. The stack feeder is equipped with a photoelectric barrier which detects if a workpiece is present. Arrival of a workpiece generates the event `sfwpar` and the event `sfwplv` occurs if a workpiece leaves the stack feeder. The rest position of the small block is detected by a magnetic sensor which triggers the events `sfr` (rest position) and `sfnr` (not in the rest position).

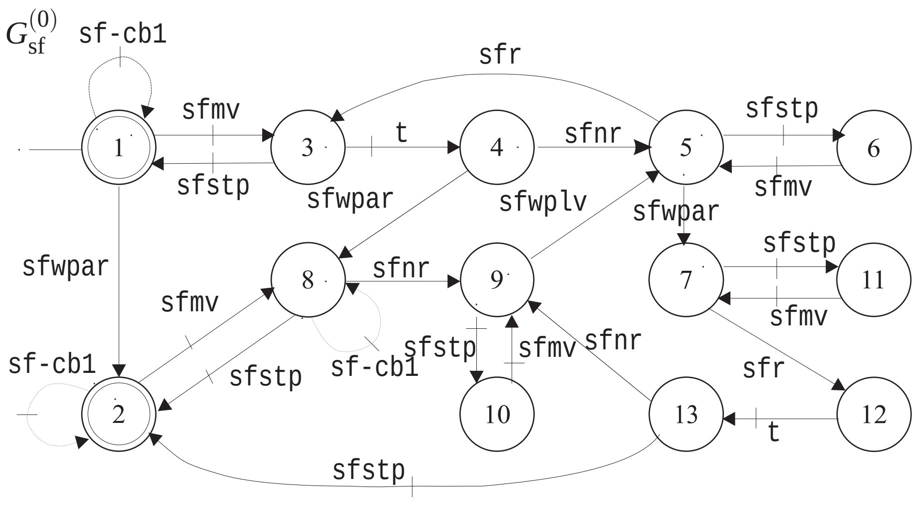


Figure 5.4: Stack feeder $G_{\text{sf}}^{(0)}$

The stack feeder is a control system. Referring to Lemma 2.9, an automaton model of the uncontrolled behavior of the stack feeder is shown in Figure 5.4. The additional event `sf-cb1` indicates that interaction with the neighboring component is possible in the respective state, i.e. a workpiece can be transported to the conveyor belt cb1. The event `t` represents the elapse of a nonzero time period until the occurrence of the next event. In this model, `t` captures the physical property that when the small block arrives at the rest position, the belt can be stopped before the rest position is left.

For the controlled stack feeder, it is desired that it only moves if a workpiece is detected at the sensor (`sfwpar`) and if cooperation with the long conveyor belt cb1 is possible (`sf-cb1` is feasible). Also, the belt has to stop if the small block reaches the rest position and only then. Figure 5.5 shows the corresponding specification automaton $D_{\text{sf}}^{(0)}$. Observing that $L_m(D_{\text{sf}}^{(0)})$ is $L_m(G_{\text{sf}}^{(0)})$-closed, Corollary 2.2 guarantees that the maximally permissive behavior fulfilling the specification can be determined. The resulting supervised behavior $\kappa_{L_m(D_{\text{sf}}^{(0)})}\big(L_m(G_{\text{sf}}^{(0)})\big)$ is implemented by the finite automaton $R_{\text{sf}}^{(0)}$ in Figure 5.6 using Lemma 2.11.

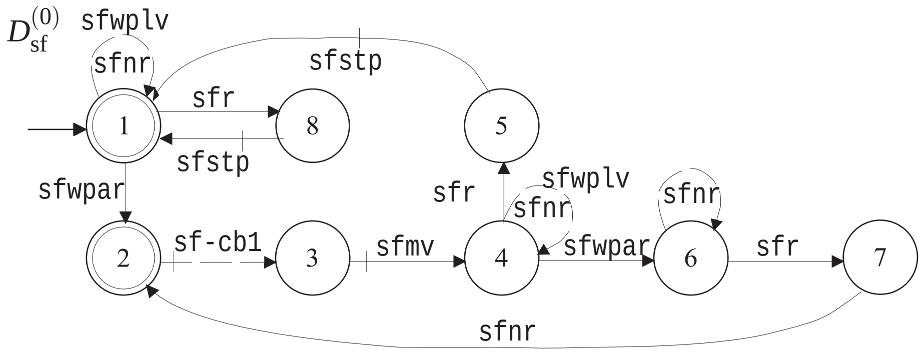


Figure 5.5: Specification automaton $D_{\text{sf}}^{(0)}$ for the stack feeder

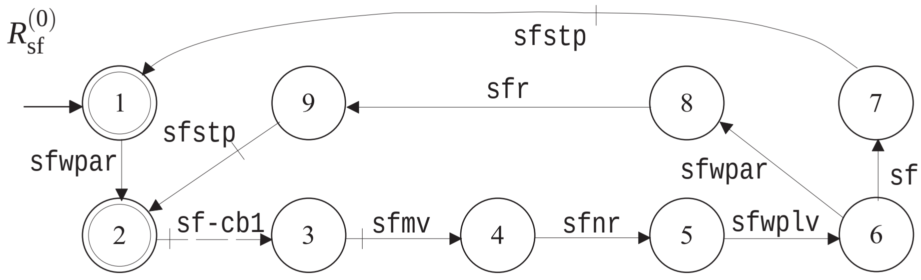


Figure 5.6: Supervised automaton R_{sf} for the stack feeder

The stack feeder is connected to the rest of the distribution system via the conveyor belt cb1. The only shared event which has to be considered for hierarchical abstraction is the event `sf-cb1`.

Thus, $\Sigma_{\mathrm{sf}}^{(1)} = \{$`cb1-sf`$\}$ is chosen. The projection $p_{\mathrm{sf}}^{(0,1)}$ yields the abstracted automaton model $G_{\mathrm{sf}}^{(1)}$ depicted in Figure 5.7. Applying the algorithms presented in the Sections 3.4.2 and 3.4.6, it can be verified that the projected system $(R_{\mathrm{sf}}^{(0)}, p_{\mathrm{sf}}^{(0,1)}, G_{\mathrm{sf}}^{(1)})$ is locally nonblocking, marked string accepting and marked string controllable. Hence, it is possible to use $G_{\mathrm{sf}}^{(1)}$ as a component in a composed system on level 1.

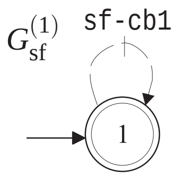


Figure 5.7: Abstracted automaton $G_{\mathrm{sf}}^{(1)}$ of the stack feeder on level 1

5.3.2 Conveyor Belt cb1

For modeling, the long conveyor is divided into three parts as shown in Figure 5.3.

5.3.2.1 Conveyor Belt cb1a and Pusher pu2

Level 0: The first part of cb1 combined with the pusher pu2 has three sensors and two actuators. The conveyor belt transports workpieces while moving into the negative x-direction (`cb1-x`). The event `sf-cb1`, which can occur in state 1 and 2 represents the possible shared behavior with the neighboring stack feeder. Arrival of a workpiece is detected by the capacitive sensor attached to the pusher pu2 (`pu2wpar` for arrival and `pu2wplv` for departure of a workpiece). The pusher pu2 pushes workpieces from cb1 to the conveyor belt cb2 (event `sf-2`). It has two push-buttons, which indicate if the pusher is in the extended (`pu2ar-y`, `pu2lv-y`) or retracted position (`pu2ar+y`, `pu2lv+y`). In general, the pusher can move forward (`pu2mv-y`), backward (`pu2mv+y`) or stop (`pu2stp`).

Low-level models for the pusher pu2 ($G_{\mathrm{pu2}}^{(0)}$) and the conveyor belt cb1a ($G_{\mathrm{cb1a}}^{(0)}$) have 20 and 8 states, respectively. It is desired that the conveyor belt moves to the negative x-direction if a workpiece is delivered from the stack feeder and until it reaches the pusher. From there, it is either further transported on the conveyor belt or pushed away by the pusher. It is required that if a workpiece is present, then the pusher either moves forward until it is completely extended or until the workpiece leaves the sensor. After that it is retracted to its rest position. The corresponding specification automata $D_{\mathrm{pu2}}^{(0)}$ and $D_{\mathrm{cb1a}}^{(0)}$ have 11 and 9 states. Locally supervised models of pu2 and cb1a are shown in Figure 5.8.

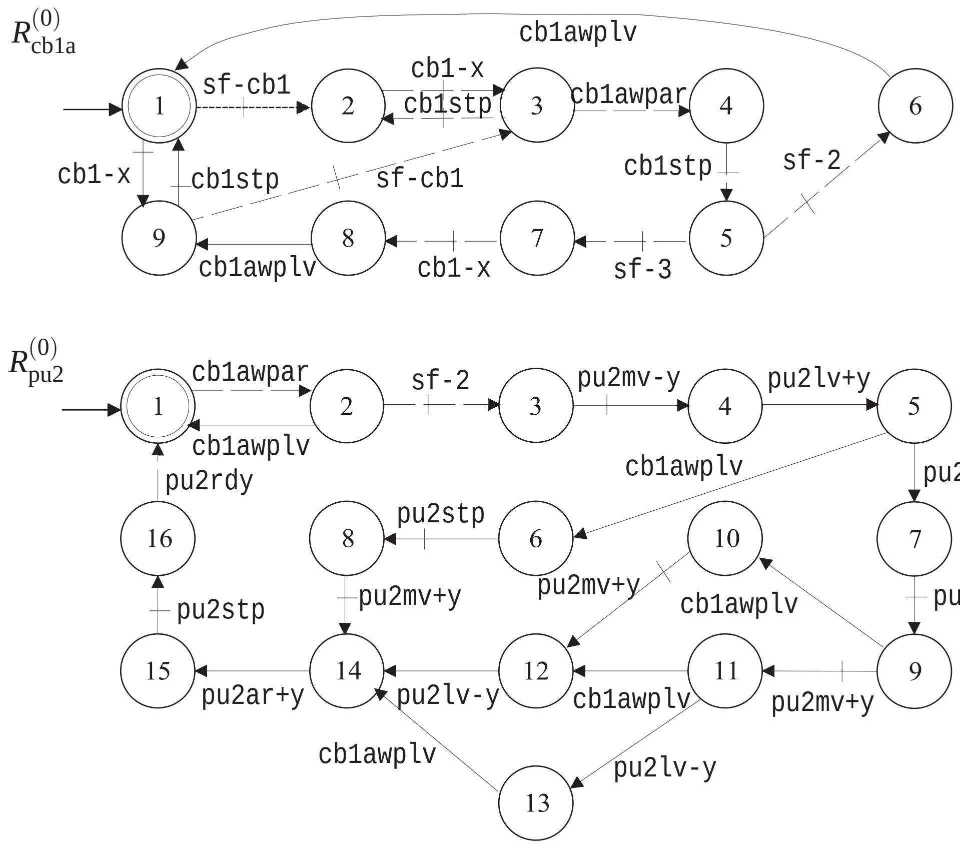


Figure 5.8: Low-level supervised models of pu2 ($R^{(0)}_{\text{pu2}}$) and cb1a ($R^{(0)}_{\text{cb1a}}$)

The automaton $G^{(0)}_{\text{cb1apu2}} := R^{(0)}_{\text{cb1a}}||R^{(0)}_{\text{pu2}}$ has 43 states and describes the composed behavior of cb1a and pu2 synchronized by the shared events `cb1awpar` and `cb1awplv`. As a requirement on this shared behavior, it is desired, that the conveyor belt does not move if the pusher is extending or retracting and that the pusher stays in its rest position as long as the conveyor belt is moving. The corresponding specification automaton $D^{(0)}_{\text{cb1apu2}}$ is depicted in Figure 5.9. The event sets Φ_1 and Φ_2 are defined as $\Phi_1 =${`cb1awpar`, `cb1awplv`, `cb1stp`, `sf-3`, `cb1-x`, `sf-cb1`} and $\Phi_2 =${`pu2stp`, `pu2mv+y`, `pu2mv-y`, `pu2ar+y`, `pu2ar-y`, `pu2lv+y`, `pu2lv-y`}.

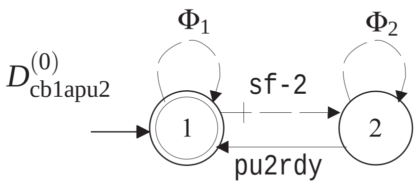


Figure 5.9: Specification automaton $D^{(0)}_{\text{cb1apu2}}$ for cb1a and pu2

Evaluating the supervisor computation $\kappa_{L_m(D^{(0)}_{\text{cb1apu2}})}\left(L_m(G^{(0)}_{\text{cb1apu2}})\right)$ yields the automaton $R^{(0)}_{\text{cb1apu2}}$

for the supervised plant as shown in Figure 5.10.

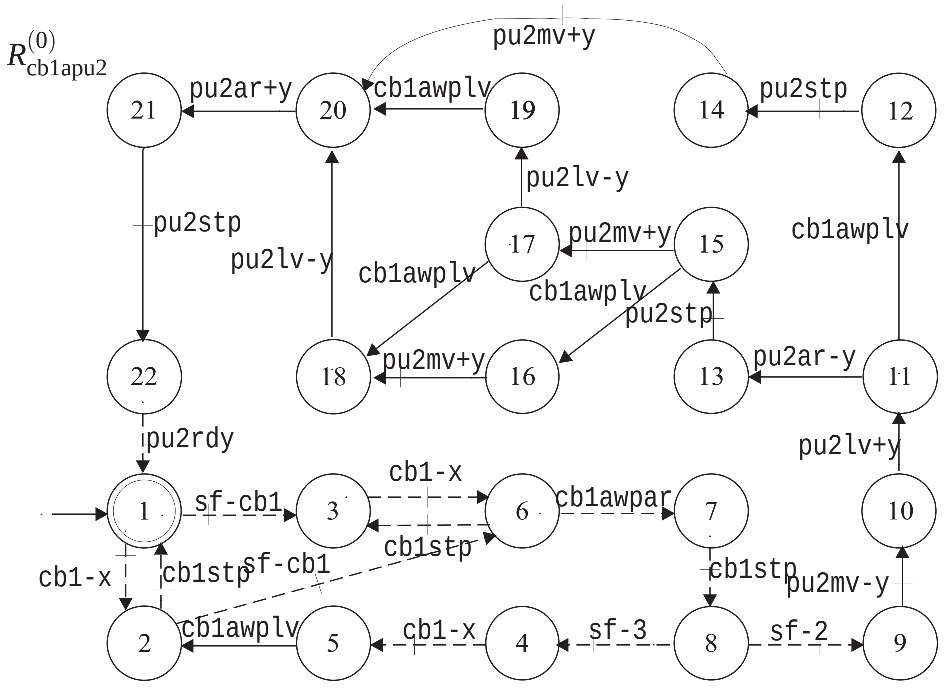


Figure 5.10: Supervised behavior $R^{(0)}_{\text{cb1apu2}}$ of cb1a and pu2

Level 1: The shared events with other components are `sf-cb1` (shared with the stack feeder), `sf-2` (shared with cb2), `sf-3` and `sf-dep` (connecting to cb1bpu1), `cb1-x` and `cb1stp` (shared with cb1bpu1). Adding `cb1awpar` and `pu2rdy` to the high-level event set [4], we arrive at $\Sigma^{(1)}_{\text{cb1apu2}} = \{$`cb1awpar`, `pu2rdy`, `sf-cb1`, `sf-2`,`sf-3`,`sf-dep`,`cb1-x`,`cb1stp`$\}$. The automaton $G^{(1)}_{\text{cb1apu2}}$ in Figure 5.11 represents the abstracted behavior of $R^{(0)}_{\text{cb1apu2}}$ on level 1. Again, the projected control system
$(R^{(0)}_{\text{cb1apu2}}, p^{(0,1)}_{\text{cb1apu2}}, G^{(1)}_{\text{cb1apu2}})$ is locally nonblocking, marked string accepting and marked string controllable.

[4]`cb1awpar` has to be reported to the high level, as the belt cb1 might be stopped in reaction to `cb1awpar` and `pu2rdy` terminates the operation of the pusher.

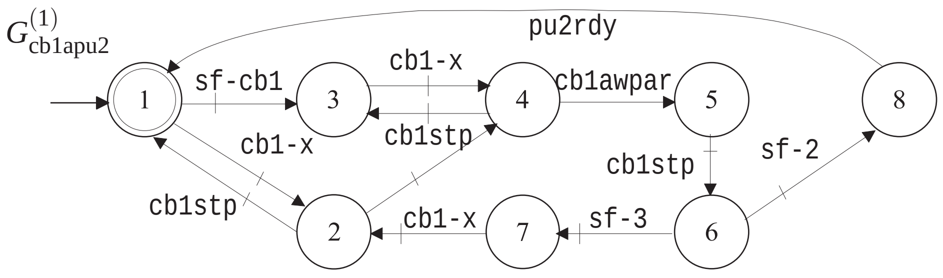


Figure 5.11: Level 1 automaton $G^{(1)}_{\text{cb1apu2}}$ of cb1apu2

5.3.2.2 Conveyor Belt cb1b and Pusher pu1

The design process for the combination of cb1b and pu1 is analogous to the synthesis in section 5.3.2.1. To sum up, this component is able to detect workpieces at the sensor of pusher pu1 and push the workpieces to the conveyor belt cb3. It is important to mention that the component cb1bpu1 is physically connected to cb1apu2 via the long belt. Because of this reason, the events `cb1-x` and `cb1stp` are shared events of these components. Also note that in this case, transport of workpieces to the depot is not allowed by the specified behavior.

The resulting model on level 1 is depicted in Figure 5.12. It can easily be verified that the PCS $(R^{(0)}_{\text{cb1bpu1}}, p^{(0,1)}_{\text{cb1bpu1}}, G^{(1)}_{\text{cb1bpu1}})$ is also locally nonblocking, marked string accepting and marked string controllable.

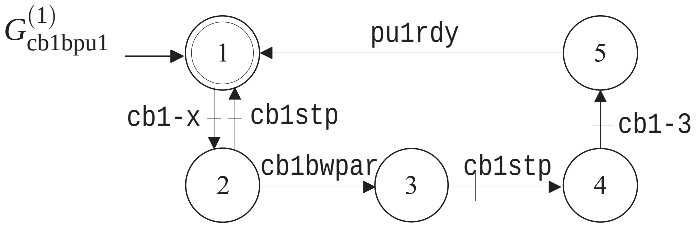


Figure 5.12: Level 1 automaton $G^{(1)}_{\text{cb1bpu1}}$ of cb1bpu1

5.3.2.3 Connection between cb1apu2 and cb1bpu1

Up to now, the two components cb1apu2 and cb1bpu1 have been modeled independently except for the shared events `cb1-x` and `cb1stp`. However, certain temporal conditions, which restrict the synchronized behavior of the components, apply. It is valid, that a workpiece can only arrive at the sensor of pusher pu1 (`cb1bwpar`) if it has passed the sensor of pusher pu2 (`cb1awpar`). In addition to that, there cannot be more than 3 workpieces between the two sensors due to physical limitations. These constraints are captured in the automaton $G^{(1)}_{\text{connect}}$ as shown in Figure 5.13.

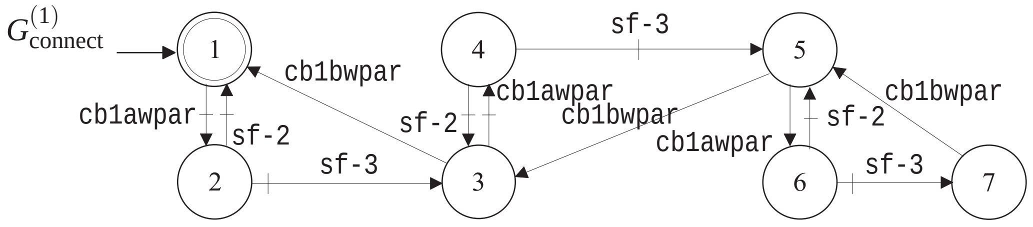


Figure 5.13: Level 1 automaton $G^{(1)}_{\text{connect}}$ connecting cb1apu2 and cb1bpu1

$G^{(1)}_{\text{connect}}$ can be seen as a buffer accepting up to 3 workpieces between the sensors of pu1 and pu2. The event `sf-3` increments the number of workpieces, and the event `cb1bwpar` decreases this number.

5.3.2.4 Complete Conveyor Belt cb1

Composing the above automata, the model of the complete conveyor belt cb1 is obtained with $G^{(1)}_{\text{cb1}} = G^{(1)}_{\text{cb1apu2}}||G^{(1)}_{\text{connect}}||G^{(1)}_{\text{cb1bpu1}}$. The resulting automaton has 67 states.

There are three specifications for cb1 (see Figure 5.14). The first two specifications address security aspects, while the third specification determines and (arbitrary) desired manufacturing routine.

- $D^{(1)}_{\text{cb1,1}}$: If the conveyor belt is empty, it is only allowed to start moving (`cb1-x`) if a workpiece is delivered from the stack feeder (`sf-cb1`). On the other hand, if cb1 is not empty, a workpiece can only be delivered from the stack feeder, if the conveyor belt is moving.

- $D^{(1)}_{\text{cb1,2}}$: The conveyor belt has to stop (`cb1stp`) if a workpiece arrives at one of the pushers (`cb1awpar` or `cb1bwpar`).

- $D^{(1)}_{\text{cb1,3}}$: There must always be two workpieces pushed by pu1 and one workpiece pushed by pu2.

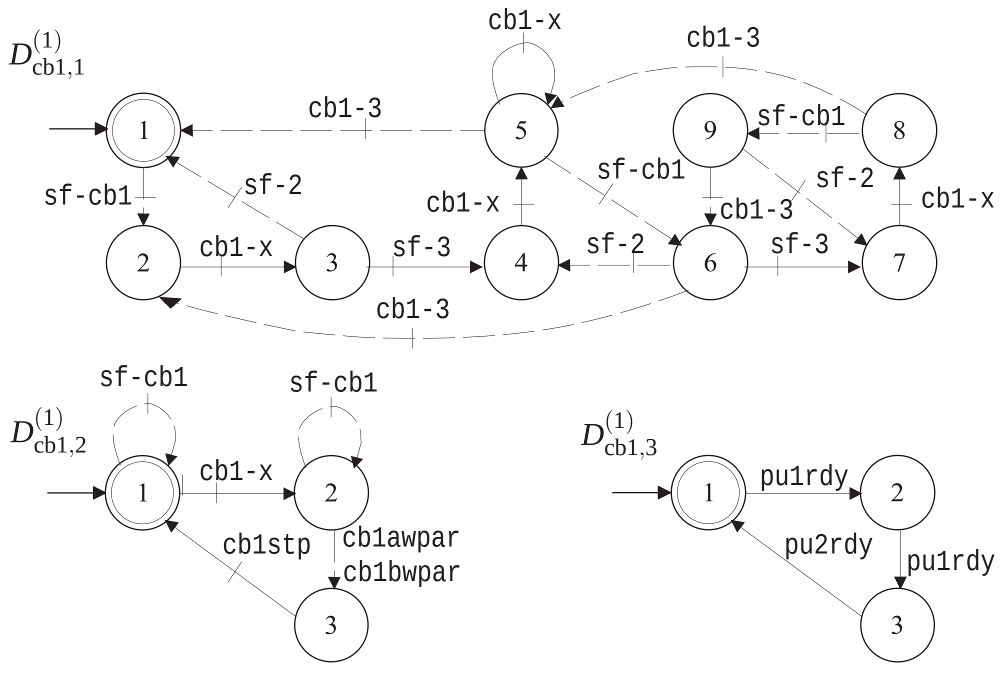


Figure 5.14: Specifications for the conveyor belt cb1 on level 1

The overall specification is computed as the synchronous composition of the above specifications $D^{(1)}_{\text{cb1}} = ||_{i=1}^{3} D^{(1)}_{\text{cb1},i}$. It has 78 states. Evaluating the supervisor computation, the controlled behavior of cb1 is represented by the automaton $R^{(1)}_{\text{cb1}}$ which recognizes the language $\kappa_{L_m(G^{(1)}_{\text{cb1}})}\left(L_m(D^{(1)}_{\text{cb1}})\right)$ and has 39 states. For hierarchical abstraction, the shared events `sf-cb1`, `cb1-3` and `sf-2` as well as the events `cb1awpar` and `cb1bwpar` are contained in the high-level alphabet $\Sigma^{(2)}_{\text{cb1}}$ ={`sf-cb1`, `cb1-3`, `sf-2`, `cb1awpar`, `cb1bwpar`}. The automaton $G^{(2)}_{\text{cb1}}$, representing the abstracted behavior is shown in Figure 5.15.

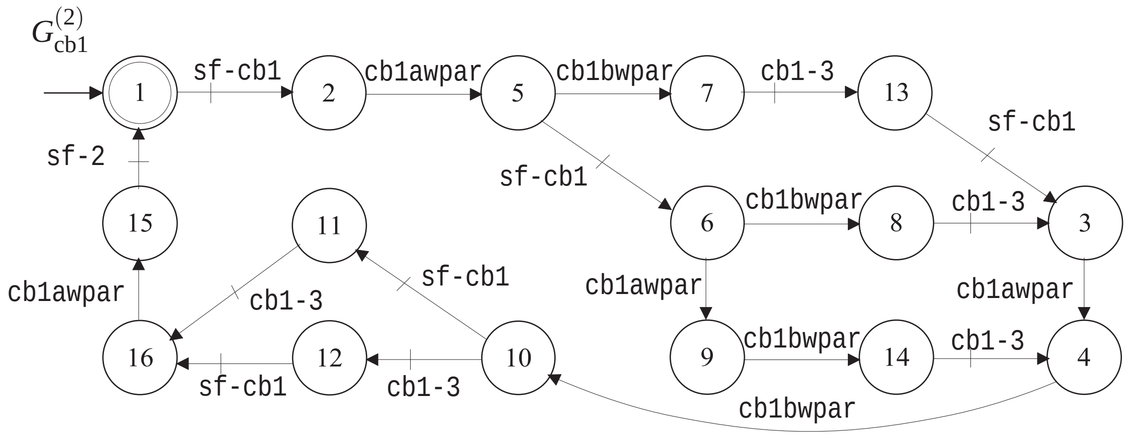


Figure 5.15: Automaton model $G^{(2)}_{\text{cb1}}$ of the conveyor belt cb1 on level 2

5.3.3 Conveyor Belts cb2 and cb3

The conveyor belts cb2 and cb3 are used in the same mode of operation. Both conveyor belts receive workpieces from the respective pusher and transport the workpieces to the remaining components of the manufacturing system. Because of this reason, only cb2 is explained in detail.

The conveyor belt can move in the negative y direction (`cb2-y`) and the sensor which is attached to it detects arrival of workpieces (`cb2wpar`). The events `wp2-13`, `sf-2` and `cb2-13` are shared events with neighboring components (see Figure 5.2). They can occur as specified in Figure 5.16.

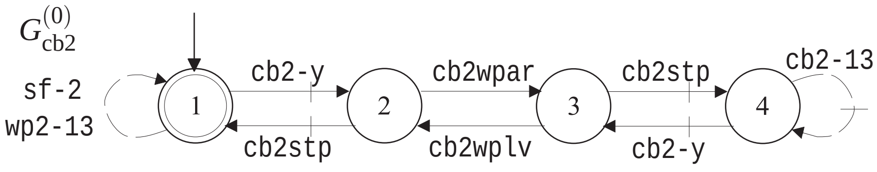


Figure 5.16: Automaton model $G^{(0)}_{\text{cb2}}$ of the conveyor belt cb2

The following requirements are specified for this conveyor belt.

- $D^{(0)}_{\text{cb2},1}$: cb2 is only allowed to move (`cb2-y`) if the events `sf-2` or `cb2-13` occurred.
- $D^{(0)}_{\text{cb2},2}$: cb2 must stop (`cb2stp`) when a workpiece arrives at its sensor (`cb2wpar`).
- $D^{(0)}_{\text{cb2},3}$: The event `wp2-13` happens if a workpiece left (`cb2wplv`) and the conveyor belt stopped.
- $D^{(0)}_{\text{cb2},4}$: The conveyor belt has to stop if a workpiece arrives or leaves.

Composing the four specifications as $D^{(0)}_{\text{cb2}} = ||_{i=1}^{4} D^{(0)}_{\text{cb2},i}$, an overall specification automaton with 7 states is derived as shown in Figure 5.17.

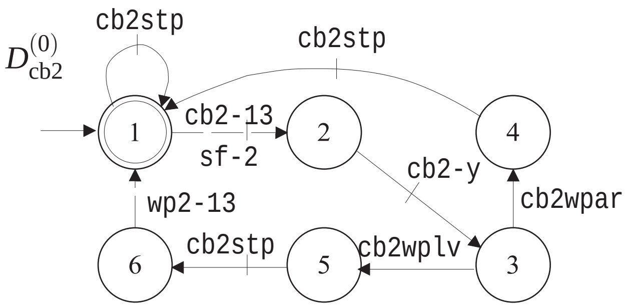


Figure 5.17: Specification automaton $D^{(0)}_{\text{cb2}}$

The maximally permissive supervisor for this specification implements the supremal controllable sublanguage $\kappa_{L_m(D^{(0)}_{\text{cb2}})}\big(L_m(G^{(0)}_{\text{cb2}})\big)$. It is recognized by the automaton $R^{(0)}_{\text{cb2}}$ in Figure 5.18.

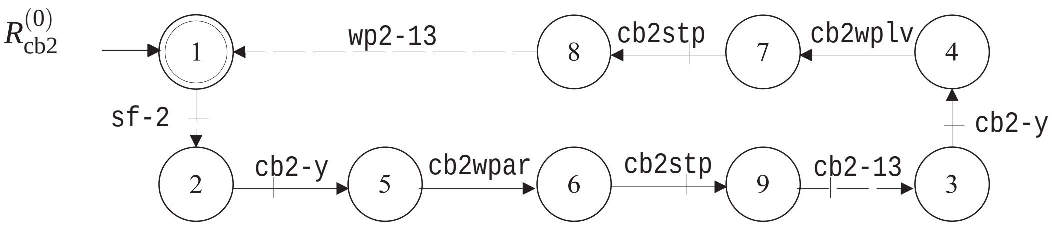


Figure 5.18: Supervised behavior of cb2 represented by $R^{(0)}_{\text{cb2}}$

The shared events of cb2 with the other components of the distribution system are $\Sigma^{(1)}_{\text{cb2}}$ ={`sf-2`, `cb2-13`, `wp2-13`}. Abstracting with the projection $p^{(0,1)}_{\text{cb2}} : (\Sigma^{(0)}_{\text{cb2}})^* \rightarrow (\Sigma^{(1)}_{\text{cb2}})^*$, the automaton representation $G^{(1)}_{\text{cb2}}$ on level 1 can be computed as shown in Figure 5.19. It is readily observed that the PCS $(R^{(0)}_{\text{cb2}}, p^{(0,1)}_{\text{cb2}}, G^{(1)}_{\text{cb2}})$ is locally nonblocking, marked string accepting and marked string controllable.

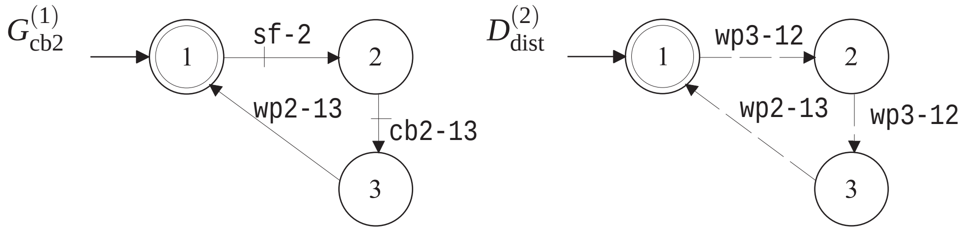


Figure 5.19: Level 2 automaton $G^{(1)}_{\text{cb2}}$ for cb2 and specification $D^{(2)}_{\text{dist}}$ for the distribution system

5.3.4 Overall Distribution System

Having applied local control to the components of the distribution system, the overall system is constructed on level 2 of the hierarchy. To this end, let the level 2 models of the stack feeder and the conveyor belts cb2 and cb3 be equal to the level 1 models, i.e. $G^{(2)}_{\text{dist}} = G^{(1)}_{\text{sf}}$, $G^{(2)}_{\text{cb2}} = G^{(1)}_{\text{cb2}}$ and $G^{(2)}_{\text{cb3}} = G^{(1)}_{\text{cb3}}$. Then, the distribution system is $G^{(2)}_{\text{dist}} = G^{(2)}_{\text{sf}}||G^{(2)}_{\text{cb1}}||G^{(2)}_{\text{cb2}}||G^{(2)}_{\text{cb3}}$. The automata representation has 144 states. We now specify a desired behavior requiring that always two workpieces leave the conveyor belt cb3 before one workpiece can leave cb2 (see Figure 5.19).

The supervisor automaton $R^{(2)}_{\text{dist}}$ on level 2 has 69 states. The shared events with the rest of the manufacturing system are `cb3-12`, `wp3-12`, `cb2-13` and `wp2-13`. Thus, the high-level event set $\Sigma^{(3)}_{\text{dist}}$ ={`cb3-12`, `wp3-12`, `cb2-13`, `wp2-13`} is chosen. Abstracting the distribution system to the third level results in the automaton $G^{(3)}_{\text{dist}}$ as depicted in Figure 5.20. The projected control system

$(R_{\text{dist}}^{(2)}, p_{\text{dist}}^{(2,3)}, G_{\text{dist}}^{(3)})$ is locally nonblocking, marked string accepting and marked string controllable. Thus the level-3 model of the distribution system can be used as a component in the overall model of the manufacturing system.

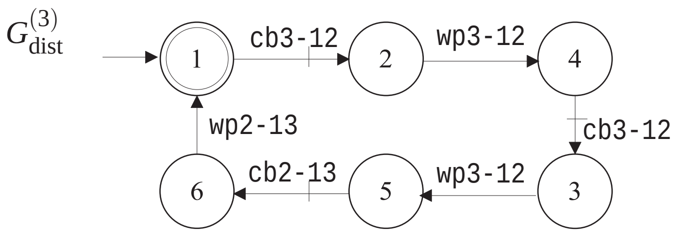


Figure 5.20: Automaton $G_{\text{dist}}^{(3)}$ for the abstracted distribution system on level 3

The complete hierarchy is shown in Figure 5.21. It is interesting to take a closer look at the high-level supervisor automata (highlighted by the shaded boxes) which have to be implemented in the low level (for example $R_{\text{dist}}^{(2)}$). All of them are live. Hence, because of Lemma 3.10, the corresponding low-level supervisors do not have to be computed extra. The high-level supervisors can directly be used for implementing the low-level control.

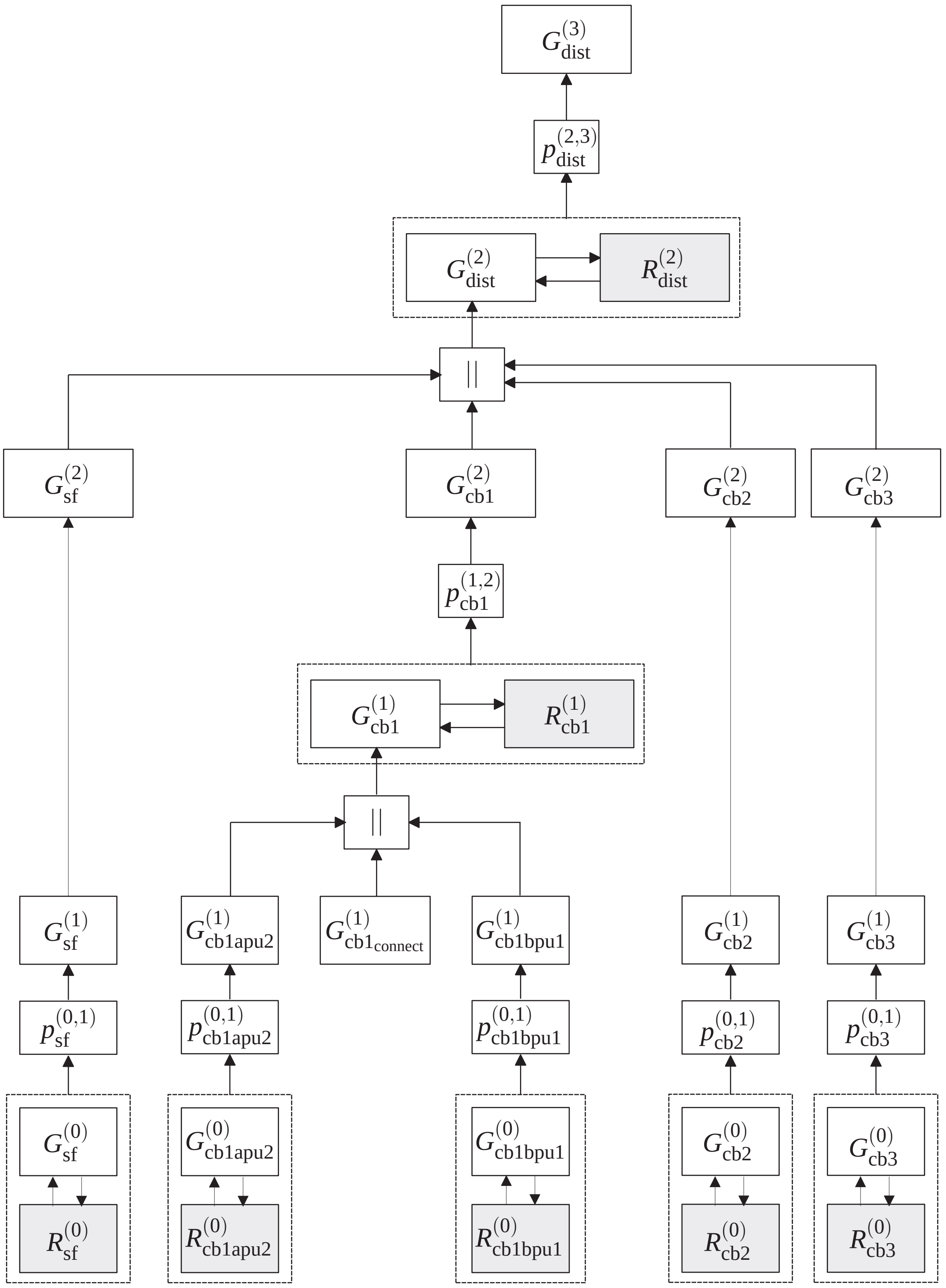


Figure 5.21: Hierarchical architecture for the distribution system

5.3.5 Performance Evaluation

The level 3 model of the distribution system has been constructed starting from low-level models of the different components sf, cb1apu2, cb1bpu1, cb1$_{\text{connect}}$, cb2 and cb3. An overview of the different automata with their respective state counts is given in Table 5.1.

Level 0

$G^{(0)}_{\text{sf}}$	$D^{(0)}_{\text{sf}}$	$R^{(0)}_{\text{sf}}$	$G^{(0)}_{\text{cb1apu2}}$	$D^{(0)}_{\text{cb1apu2}}$	$R^{(0)}_{\text{cb1apu2}}$	$G^{(0)}_{\text{cb1bpu1}}$	$D^{(0)}_{\text{cb1bpu1}}$	$R^{(0)}_{\text{cb1bpu1}}$
13	8	9	43	2	22	27	2	20
$G^{(0)}_{\text{cb2}}$	$D^{(0)}_{\text{cb2}}$	$R^{(0)}_{\text{cb2}}$	$G^{(0)}_{\text{cb3}}$	$D^{(0)}_{\text{cb3}}$	$R^{(0)}_{\text{cb3}}$			
4	7	9	4	7	9			

Level 1

$G^{(1)}_{\text{sf}}$	$G^{(1)}_{\text{cb1apu2}}$	$G^{(1)}_{\text{cb1bpu1}}$	$G^{(1)}_{\text{cb1}_{\text{connect}}}$	$G^{(1)}_{\text{cb2}}$	$G^{(1)}_{\text{cb3}}$	$G^{(1)}_{\text{cb1}}$	$D^{(1)}_{\text{cb1}}$	$R^{(1)}_{\text{cb1}}$
1	8	6	7	3	3	67	78	39

Level 2

$G^{(2)}_{\text{sf}}$	$G^{(2)}_{\text{cb1}}$	$G^{(2)}_{\text{cb2}}$	$G^{(2)}_{\text{cb3}}$	$G^{(2)}_{\text{dist}}$	$D^{(2)}_{\text{dist}}$	$R^{(2)}_{\text{dist}}$		
1	16	3	3	144	9	138		

Level 3

$G^{(3)}_{\text{dist}}$								
6								

Table 5.1: State quantities of the automata forming the distribution system

The monolithic approach is compared with the hierarchical and decentralized method for classifying the computational effort of synthesis and implementation. The composite plant automaton $G^{(0)}_{\text{dist}} = G^{(0)}_{\text{sf}}||G^{(0)}_{\text{cb1a}}||G^{(0)}_{\text{cb1b}}||G^{(0)}_{\text{cb1}_{\text{connect}}}||G^{(0)}_{\text{pu1}}||G^{(0)}_{\text{pu2}}||G^{(0)}_{\text{cb2}}||G^{(0)}_{\text{cb3}}$ has 360 000 states and the monolithic specification is represented by an automaton $D^{(0)}_{\text{dist}}$ with $3 \cdot 10^6$ states. Applying standard supervisory control, the closed-loop automaton $R^{(0)}_{\text{dist}}$ has 400 000 states.

It is evident that the large number of states is caused by the fact that the state sizes of the components are multiplied when the overall automaton is computed. Different from that, the decentralized approach avoids computing the complete low-level model as it makes use of the decentralized nature of the system.

On the low level, there are 5 decentralized supervisors with a sum of 71 states, on level 1 there is one supervisor with 39 states and on the second level, there is also one supervisor with 69 states. Together, the hierarchical and decentralized supervisors have 179 states. The considerable discrepancy in the state sizes of the different supervisor implementations originates from the fact that the state sizes of the components have to be multiplied for the monolithic approach while they are just added for the hierarchical and decentralized method. It is also important to note, that not only the number of states of the decentralized supervisors is smaller, but also the complexity for computing the supervisors is lower than for the monolithic approach.

Additionally, PLC code has been generated from the automata representations of the hierarchical and decentralized supervisors (see Figure 5.22 as an example) with a tool, that was developed in a student project at our institute ([Fig05]). The automata representation is converted into PLC functions, and practical issues such as *concurrency* and the *sequence of commands* are addressed. The PLC running the code generated from the hierarchical and decentralized supervisors of the distribution system operates the Fischertechnik simulation model correctly.

```
L       "state"
JL      END
JU      A001
JU      A002
         :
JU      A009
END     BEU

// automata realization
a001:   L    1
        T    "state".sf[0]
A001:   A    "event":sfwpar
        R    "event":sfwpar
        JC   a002
        JU   END
```

```
a002:   L    2
        T    "state".sf[0]
A002:   S    "event".sf-cb1
        JU   a002
a003:   L    3
        T    "state".sf[0]
A003:   S    "event".sfmv
        JU   a004
a004:   L    4
        T    "state".sf[0]
a004:   A    "event".sfnr
        R    "event".sfnr
        JC   a005
        JU   END
```

```
a005:   L    5
        T    "state".sf[0]
A005:   A    "event".sfwplv
        R    "event".sfwplv
        JC   a006
        JU   END
a006:   L    6
        T    "state".sf[0]
A006:   A    "event".sfwpar
        R    "event".sfwpar
        JC   a008
        A    "event".sfr
        R    "event".sfr
        JC   a007
        JU   END
```

```
a007:   L    7
        T    "state".sf[0]
A007:   S    "event".sfstp
        JU   a001
a008:   L    8
        T    "state".sf[0]
A008:   A    "event".sfr
        R    "event".sfr
        JC   a009
        JU   END
a009:   L    9
        T    "state".sf[0]
A009:   S    "event".sfstp
        JU   a002
```

Figure 5.22: PLC code for the level 0 supervisor $R_{\mathrm{sf}}^{(0)}$ of the stack feeder

5.4 Hierarchical and Decentralized Control for the Manufacturing System

After the detailed description of the hierarchical and decentralized supervisor synthesis for the distribution system, a controller for the overall Fischertechnik model is designed. To this end, the manufacturing system is divided into the 6 structural entities shown in Figure 5.2. The *distribution system* is adopted from Section 5.3. The production cells *pc1* and *pc2* are composed of cb4, mh1, d1, rt1, cb11, cb7 and cb6, mh2, d2, rt4, cb14, cb10, respectively. The components rt2, cb12, cb8, cb5, rt3, cb13 and cb9 form a system component which allows of exchanging workpieces between the different parts of the manufacturing system. It is called the interchange system *ics*. Also, there are two rail transport systems (rts1 and rts2) with conveyor belts (cb15 and cb16), each combined with a roll conveyor (rc1 and rc2). As the main purpose of these components is to remove workpieces from the manufacturing system, they are denoted *exit1* (rts1, cb15, rc1) and *exit2* (rts2, cb16, rc2).

First, an overview of the functionality and supervisor synthesis for these components is worked out. All locally controlled components are then composed to form the overall manufacturing system and finally, a supervisor for a high-level progress specification is synthesized.

5.4.1 Production Cell pc1

The production cell pc1 consists of the conveyor belt cb4, the machine with drill mh1d1, the rotary table rt1 with the conveyor belt cb11 and the conveyor belt cb7. Low-level models for these components are derived analogously to Section 5.3.

cb4: Applying a similar specification as for cb2 in Section 5.3.3, the low-level closed-loop automaton $R^{(0)}_{\text{cb4}}$ has 18 states and the projection on level 1 is $G^{(1)}_{\text{cb4}}$ as depicted in Figure 5.23.

mh1d1: The machine head mh1 can move up and down, and the drill that is attached to it can start its operation any time. The rest position of mh1 is the "upper" (+z) position, while workpieces can be drilled when it is in the "lower" (-z) position. A supervisor, guaranteeing that the drill only works in the down position and that the machine head only leaves its rest position if a workpiece is to be processed, results in a closed-loop automaton $R^{(0)}_{\text{mh1d1}}$ with 12 states. The level 1 model $G^{(1)}_{\text{mh1d1}}$ has 2 states (see Figure 5.23).

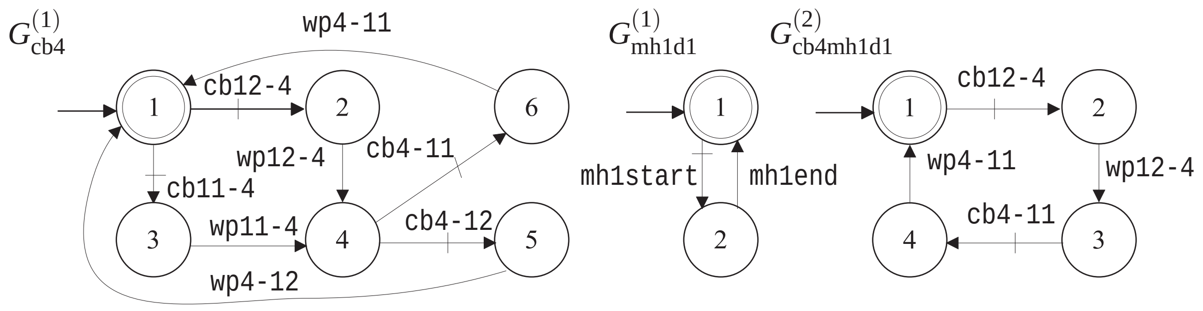


Figure 5.23: Conveyor belt cb4 with machine mh1d1 on level 1 and 2

cb4mh1d1: On level 1, the models of cb4 and mh1d1 are composed to $G^{(1)}_{\text{cb4mh1d1}} = G^{(1)}_{\text{cb4}} || G^{(1)}_{\text{mh1d1}}$ (12 states). For this component, it is specified that the conveyor belt cb4 is not allowed to move if a workpiece is currently processed by the machine and that the machine is not allowed to move as long as the conveyor belt is transporting workpieces. In addition to that, workpieces shall only move from right to left. The maximal permissive supervisor automaton $R^{(1)}_{\text{cb4mh1d1}}$ implementing this specification has 6 states. The projection $p^{(1,2)}_{\text{cb4mh1d1}}$ on the shared events with the neighboring components yields the automaton $G^{(2)}_{\text{cb4mh1d1}}$ with 4 states as shown in Figure 5.23.

rt1: The rotary table rt1 is equipped with 2 sensors, which indicate if rt1 is in the x- or in the y-position[5]. The rotary table rt1 is able to turn clockwise from the x- to the y- position (event `rt1xy`) and counterclockwise (`rt1yx`). A local controller guaranteeing that the rotary table only stops at the sensors and that it always turns in the correct direction (minimal angle for reaching the next sensor) is implemented with the supervisor automaton $R^{(0)}_{\text{rt1}}$. The projected level 1 automaton $G^{(1)}_{\text{rt1}}$ is presented in Figure 5.24.

cb11: The functionality of the conveyor belt cb11 is equivalent to cb4 (see Figure 5.24).

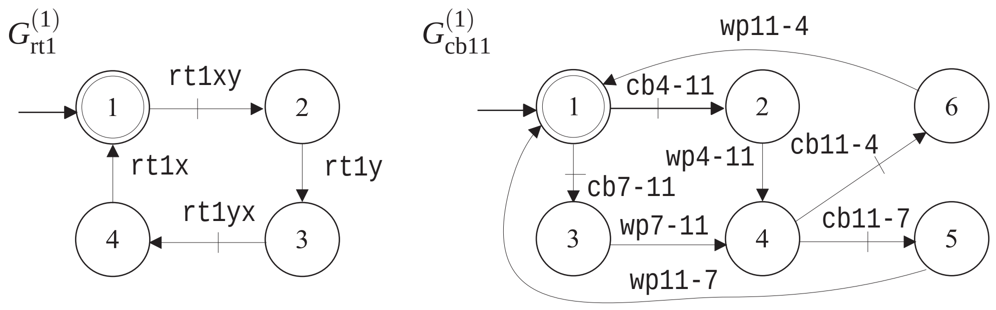


Figure 5.24: Rotary table rt1 with conveyor belt cb11

rt1cb11: The automaton $G^{(1)}_{\text{rt1cb11}} = G^{(1)}_{\text{rt1}}||G^{(1)}_{\text{cb11}}$ representing the behavior of the composition of rt1 and cb11 has 14 states. It is desired that the rotary table does not turn if the conveyor belt moves and vice versa. Also, it is required that workpieces are always transported from the machine to the next conveyor belt cb7. The closed-loop automaton $R^{(1)}_{\text{rt1cb11}}$ on level 1 has 8 states and its projection $G^{(2)}_{\text{rt1cb11}}$ is depicted in Figure 5.25.

cb7: The conveyor belt cb7 is just required to transport workpieces coming from the rotary table rt1 to the rail transport system rts1. The level 2 abstraction $G^{(2)}_{\text{cb7}}$ of the locally controlled system has 4 states (see Figure 5.25).

pc1: With the locally controlled and abstracted components $G^{(2)}_{\text{cb4mh1d1}}$, $G^{(2)}_{\text{rt1cb11}}$ and $G^{(2)}_{\text{cb7}}$, the level 2 model of the production cell pc1 is evaluated to $G^{(2)}_{\text{pc1}} = G^{(2)}_{\text{cb4mh1d1}}||G^{(2)}_{\text{rt1cb11}}||G^{(2)}_{\text{cb7}}$. This

[5]Figure 5.2 shows rt1 in the y-position and rt4 in the x-position.

automaton has 24 states. Applying the specification demanding that there be at most 2 workpieces allowed in the production cell yields the supervisor automaton $R^{(2)}_{\text{pc1}}$ with 20 states. The projection on the shared events $\Sigma^{(3)}_{\text{pc1}} = \{$`cb12-4`, `wp12-4`, `cb7-15`$\}$ results in the level 3 automaton $G^{(3)}_{\text{pc1}}$ with 5 states as presented in Figure 5.25.

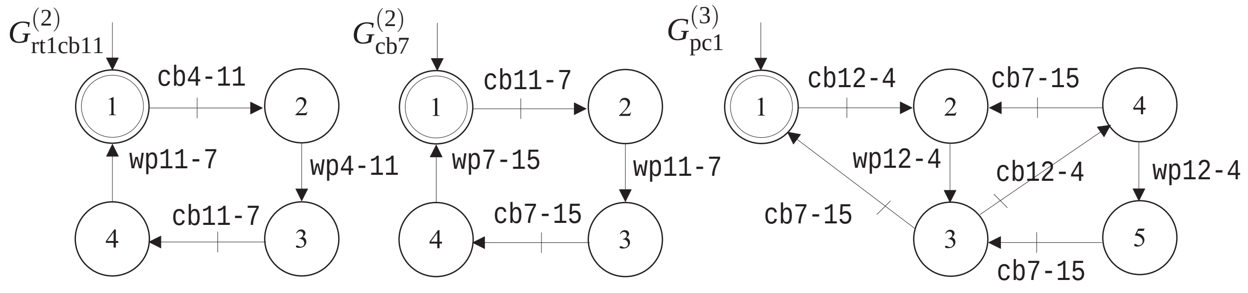


Figure 5.25: Level 2 models of rt1cb11 and cb7 and level 3 model of the production cell pc1

The automaton $G^{(3)}_{\text{pc1}}$ in the above figure is the model of the complete component pc1 on level 3. Note that all projected control systems involved in the hierarchy of pc1 are locally nonblocking and marked string accepting. In addition to that, all low-level supervisors are consistent implementations of live high-level supervisors. Thus, it is guaranteed that the overall production cell is nonblocking according to Theorem 3.1. The complete hierarchical architecture of the production cell pc1 is shown in Figure 5.26, and the sizes of the different automata are listed in Table 5.2.

Level 0

$G^{(0)}_{\text{cb4}}$	$D^{(0)}_{\text{cb4}}$	$R^{(0)}_{\text{cb4}}$	$G^{(0)}_{\text{mh1d1}}$	$D^{(0)}_{\text{mh1d1}}$	$R^{(0)}_{\text{mh1d1}}$	$G^{(0)}_{\text{cb11}}$	$D^{(0)}_{\text{cb11}}$	$R^{(0)}_{\text{cb11}}$
18	18	18	20	12	12	18	18	18
$G^{(0)}_{\text{rt1}}$	$D^{(0)}_{\text{rt1}}$	$R^{(0)}_{\text{rt1}}$	$G^{(0)}_{\text{cb7}}$	$D^{(0)}_{\text{cb7}}$	$R^{(0)}_{\text{cb7}}$			
10	12	12	18	10	10			

Level 1

$G^{(1)}_{\text{cb4}}$	$G^{(1)}_{\text{mh1d1}}$	$G^{(1)}_{\text{cb11}}$	$G^{(1)}_{\text{rt1}}$	$G^{(1)}_{\text{cb7}}$	$G^{(1)}_{\text{cb4mh1d1}}$	$D^{(1)}_{\text{cb4mh1d1}}$	$R^{(1)}_{\text{cb4mh1d1}}$	$G^{(1)}_{\text{rt1cb11}}$
6	2	6	4	4	12	4	6	16
$D^{(1)}_{\text{rt1cb11}}$	$R^{(1)}_{\text{rt1cb11}}$							
8	8							

Level 2 **Level 3**

$G^{(2)}_{\text{cb4mh1d1}}$	$G^{(2)}_{\text{rt1cb11}}$	$G^{(2)}_{\text{cb7}}$	$G^{(2)}_{\text{pc1}}$	$D^{(2)}_{\text{pc1}}$	$R^{(2)}_{\text{pc1}}$	$G^{(3)}_{\text{pc1}}$		
4	4	4	24	5	20	3		

Table 5.2: State quantities of the automata forming the production cell pc1

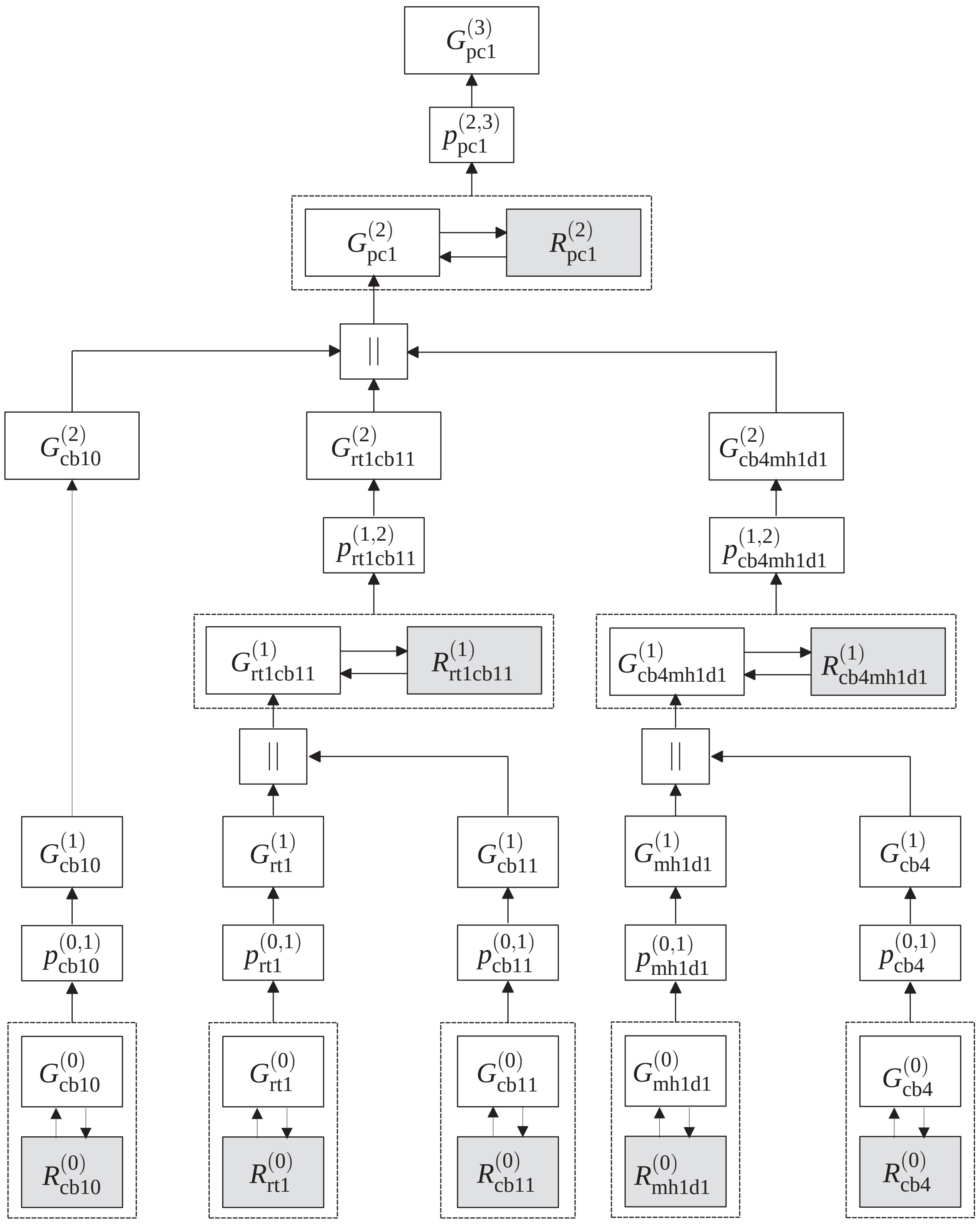


Figure 5.26: Hierarchical architecture for the production cell pc1

5.4.2 exit1

The component exit1 consists of the subsystems rts1, cb15 and rc1. The rail transport system rts1 can move between the roll conveyor rc1 and the conveyor belts cb7, cb8 and cb9, and workpieces are either loaded on or unloaded from the conveyor belt cb15.

rts1: The behavior of the rail transport system shall be restricted to either moving to rc1 or cb9 or coming back to the rest position at cb7. The supervisor $R^{(0)}_{\text{rts1}}$ guarantees the specified behavior. The abstracted automaton $G^{(1)}_{\text{rts1}}$ on level 1 has 10 states.

cb15: Similar to cb6, it is required that cb15 only accept workpieces from cb7 and deliver to either rc1 or cb9 (automaton $R^{(0)}_{\text{cb15}}$). The projected automaton $G^{(1)}_{\text{cb15}}$ on level 1 has 5 states.

rc1: The roll conveyor rc1 has space for 4 workpieces. It only detects the arrival and departure of workpieces. The level 1 model $G^{(1)}_{\text{rc1}}$ is depicted in Figure 5.27.

exit1: Composing the exit system $G^{(1)}_{\text{exit1}} = G^{(1)}_{\text{rts1}}||G^{(1)}_{\text{cb15}}||G^{(1)}_{\text{rc1}}$ leads to a level 1 model with 58 states. The supervised system $R^{(1)}_{\text{exit1}}$ must fulfill the following requirements:

- cb15 must not move while rts1 is moving and vice versa.
- rts1 must wait at cb7 until a workpiece arrives
- if rts1 moves to rc1 or cb9, it must deliver a workpiece

The supervisor automaton has 28 states. Figure 5.27 presents the level 2 automaton $G^{(2)}_{\text{exit1}}$ after the projection $p^{(1,2)}_{\text{exit1}}$ to the level 2 event set $\Sigma^{(2)}_{\text{exit1}} = \{\texttt{cb7-15}, \texttt{cb15-rc1}, \texttt{cb15-9}, \texttt{wp15-9}\}$.

All projected control systems involved in the hierarchy of exit1 are locally nonblocking and marked string accepting. In addition to that, all low-level supervisors are consistent implementations of live high-level supervisors. Thus, it is guaranteed that the overall exit system is nonblocking according to Theorem 3.1. The complete hierarchical architecture of exit1 is shown in Figure 5.28 and the sizes of the different automata are listed in Table 5.3.

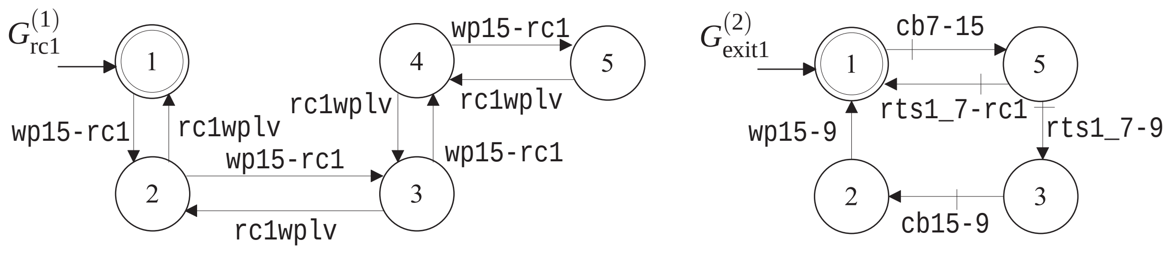


Figure 5.27: Roll conveyor rc1 on level 1 and exit system exit1 on level 2

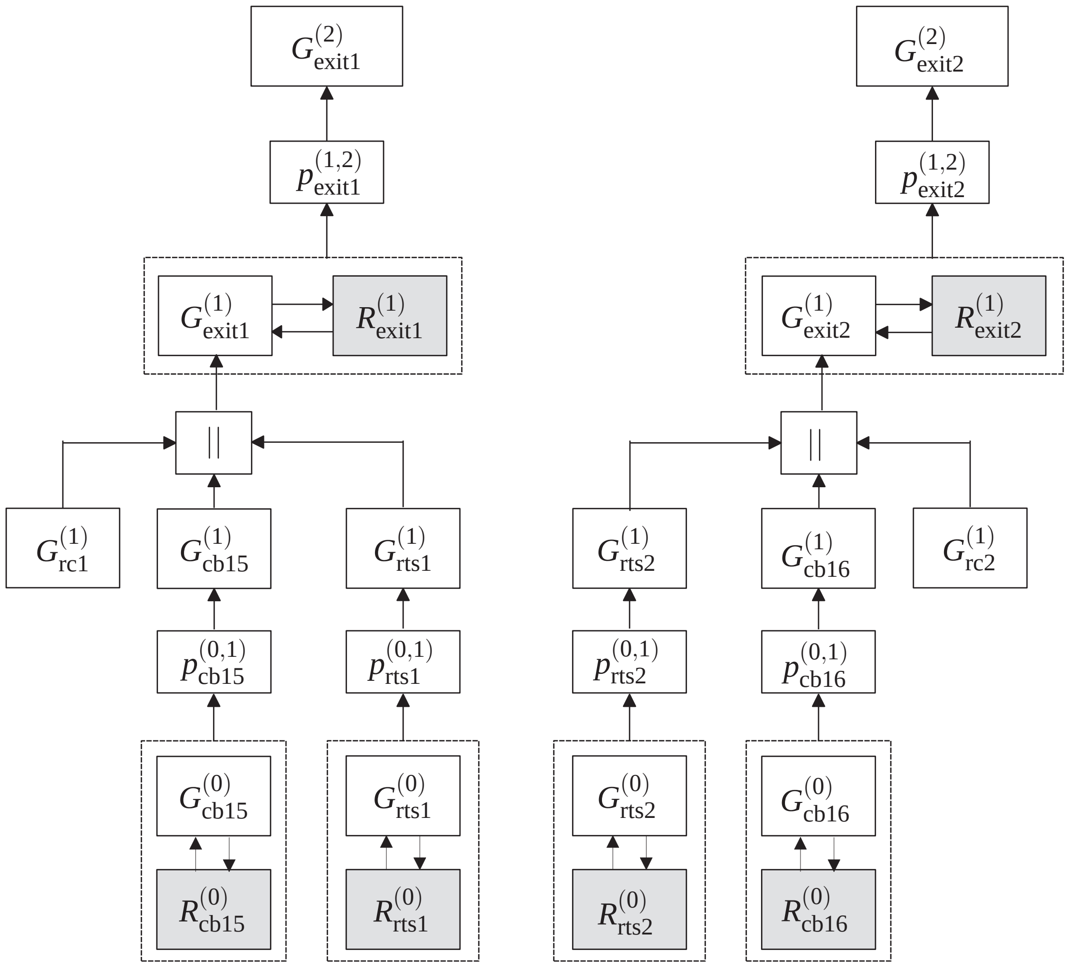

$G^{(2)}_{\mathrm{exit1}}$
$p^{(1,2)}_{\mathrm{exit1}}$
$G^{(1)}_{\mathrm{exit1}}$
$R^{(1)}_{\mathrm{exit1}}$
$||$
$G^{(1)}_{\mathrm{rc1}}$
$G^{(1)}_{\mathrm{cb15}}$
$G^{(1)}_{\mathrm{rts1}}$
$p^{(0,1)}_{\mathrm{cb15}}$
$p^{(0,1)}_{\mathrm{rts1}}$
$G^{(0)}_{\mathrm{cb15}}$
$R^{(0)}_{\mathrm{cb15}}$
$G^{(0)}_{\mathrm{rts1}}$
$R^{(0)}_{\mathrm{rts1}}$
$G^{(2)}_{\mathrm{exit2}}$
$p^{(1,2)}_{\mathrm{exit2}}$
$G^{(1)}_{\mathrm{exit2}}$
$R^{(1)}_{\mathrm{exit2}}$
$||$
$G^{(1)}_{\mathrm{rts2}}$
$G^{(1)}_{\mathrm{cb16}}$
$G^{(1)}_{\mathrm{rc2}}$
$p^{(0,1)}_{\mathrm{rts2}}$
$p^{(0,1)}_{\mathrm{cb16}}$
$G^{(0)}_{\mathrm{rts2}}$
$R^{(0)}_{\mathrm{rts2}}$
$G^{(0)}_{\mathrm{cb16}}$
$R^{(0)}_{\mathrm{cb16}}$

Figure 5.28: Hierarchical architecture for the exit systems exit1 and exit2

Level 0

$G^{(0)}_{\text{rts1}}$	$D^{(0)}_{\text{rts1}}$	$R^{(0)}_{\text{rts1}}$	$G^{(0)}_{\text{cb15}}$	$D^{(0)}_{\text{cb15}}$	$R^{(0)}_{\text{cb15}}$
28	5	32	30	14	14

Level 1

$G^{(1)}_{\text{exit1}}$	$D^{(1)}_{\text{exit1}}$	$R^{(1)}_{\text{exit1}}$	$G^{(1)}_{\text{rts1}}$	$G^{(1)}_{\text{cb15}}$	$G^{(1)}_{\text{rc1}}$
30	9	14	5	10	3

Level 2

$G^{(2)}_{\text{exit1}}$					
5					

Table 5.3: State numbers of the automata forming the exit system exit1

5.4.3 exit2

The component exit2 consists of the subsystems rts2, cb16 and rc2. The rail transport system rts2 can move between the roll conveyor rc2 and the conveyor belt cb8 and workpieces are either loaded on or unloaded from the conveyor belt cb16.

rts2: The rail transport system rts2 is only allowed to move from its rest position at the conveyor belt cb10 to the roll conveyor rc2. The supervisor $R^{(0)}_{\text{rts2}}$ guarantees the specified behavior. The abstracted automaton $G^{(1)}_{\text{rts2}}$ on level 1 has 4 states.

cb16: The conveyor belt cb16, is required to deliver workpieces from cb10 to rc2. (automaton $R^{(0)}_{\text{cb16}}$). The projected automaton $G^{(1)}_{\text{cb16}}$ on level 1 has 4 states.

rc2: The roll conveyor rc2 has the same behavior as rc1.

exit2: Computing the overall exit system $G^{(1)}_{\text{exit2}} = G^{(1)}_{\text{rts2}}||G^{(1)}_{\text{cb16}}||G^{(1)}_{\text{rc2}}$, leads to a level 1 model with 40 states.[6] The supervised system $R^{(1)}_{\text{exit2}}$ must fulfill the following requirements:

- cb16 must not move if rts2 is moving and vice versa.
- rts2 must wait at cb10 until a workpiece arrives

[6]exit1 and exit2 are of symmetric structure. The difference between $G^{(1)}_{\text{exit1}}$ and $G^{(1)}_{\text{exit2}}$ results from the restriction that exit2 is not allowed to unload workpieces to cb8.

- if rts1 moves to rc2, it must deliver a workpiece

The supervisor automaton $R^{(2)}_{\text{exit2}}$ has 30 states. Figure 5.29 presents the level 2 automaton $G^{(2)}_{\text{exit2}}$. As this component works like a sink (accepting workpieces), it has just one state.

All projected control systems involved in the hierarchy of exit2 are locally nonblocking and marked string accepting. In addition to that, all low-level supervisors are consistent implementations of live high-level supervisors. Thus, it is guaranteed that the overall exit system is nonblocking according to Theorem 3.1. The complete hierarchical architecture of exit2 is shown in Figure 5.28, and the sizes of the different automata are listed in Table 5.4.

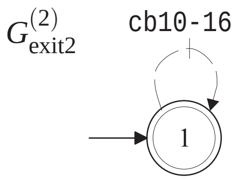


Figure 5.29: Exit system exit2 on level 2

Level 0

$G^{(0)}_{\text{rts2}}$	$D^{(0)}_{\text{rts2}}$	$R^{(0)}_{\text{rts2}}$	$G^{(0)}_{\text{cb16}}$	$D^{(0)}_{\text{cb16}}$	$R^{(0)}_{\text{cb16}}$
28	2	12	30	10	10

Level 1

$G^{(1)}_{\text{rts2}}$	$G^{(1)}_{\text{cb16}}$	$G^{(1)}_{\text{rc2}}$	$G^{(1)}_{\text{exit2}}$	$D^{(1)}_{\text{exit2}}$	$R^{(1)}_{\text{exit2}}$
4	2	5	40	6	30

Level 2

$G^{(2)}_{\text{exit2}}$					
1					

Table 5.4: State numbers of the automata forming the exit system exit2

5.4.4 Production Cell pc2

The production cell pc2 is composed of the conveyor belt cb6, the machine with drill mh2d2, the rotary table rt4 with the conveyor belt cb14 ,the conveyor belt cb10 and the exit system as described in Section 5.4.3. The exit system exit2 is treated as part of the production cell pc2, as it only interacts with this system component. Interaction with other parts of the manufacturing system is not specified. The supervisor synthesis for pc2 is similar to the approach in Section 5.4.1.

cb6: The conveyor belt cb6 has the same functionality as cb4.

mh2d2: The same supervisor as for the machine mh1d1 is synthesized for the machine mh2d2, and the projection of the closed-loop behavior to level 1 is represented by the automaton $G^{(1)}_{\text{mh2d2}}$ with 2 states analogously to $G^{(1)}_{\text{mh1d1}}$.

cb6mh2d2: For the composed system $G^{(1)}_{\text{cb6mh2d2}} = G^{(1)}_{\text{cb6}}||G^{(1)}_{\text{mh2d2}}$, it is specified that the conveyor belt cb6 is not allowed to move if a workpiece is processed by the machine and that the machine is not allowed to move as long as the conveyor belt is transporting workpieces. In addition to that, workpieces are only allowed to enter cb6 from the conveyor belt cb13 (i.e. from -x). The supervisor automaton $R^{(1)}_{\text{cb6mh2d2}}$ implementing this specification has 7 states and the projection $p^{(1,2)}_{\text{cb6mh2d2}}$ on the events shared with the neighboring components yields the automaton $G^{(2)}_{\text{cb6mh2d2}}$ with 5 states as shown in Figure 5.30.

rt4cb14 and cb10: The components $G^{(2)}_{\text{rt4cb14}}$ and $G^{(2)}_{\text{cb10}}$ are constructed like the components $G^{(2)}_{\text{rt1cb11}}$ and $G^{(2)}_{\text{cb7}}$ of pc1, respectively. They are depicted in Figure 5.30.

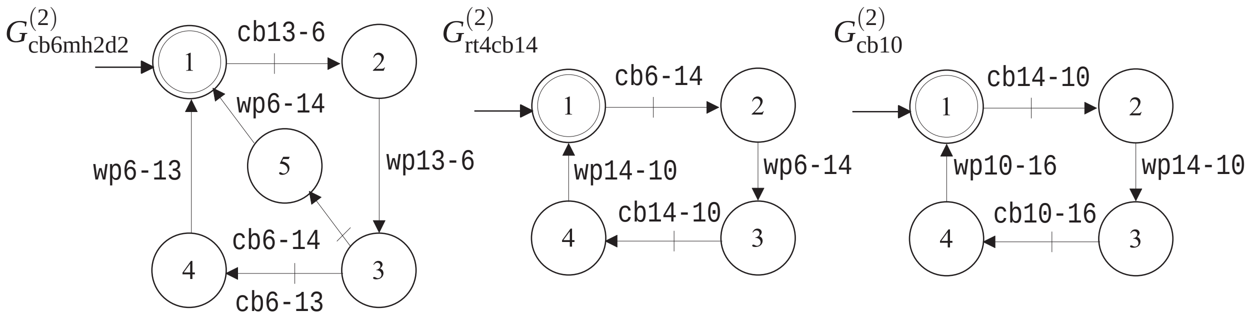


Figure 5.30: Level 2 models of cb6mh2d2, rt4cb14 and cb10

exit2: For the exit system, the level 2 model $G^{(2)}_{\text{exit2}}$ in Figure 5.29 has already been elaborated.

pc2: The level 2 model of the production cell pc2 is $G^{(2)}_{\text{pc2}} = G^{(2)}_{\text{cb6mh2d2}}||G^{(2)}_{\text{rt4cb14}}||G^{(2)}_{\text{cb10}}||G^{(2)}_{\text{exit2}}$. This automaton has 31 states. Applying a specification, demanding that there are at most 2 workpieces in the production cell yields the supervisor automaton $R^{(2)}_{\text{pc2}}$ with 25 states. The projection on the shared events $\Sigma^{(3)}_{\text{pc2}} = \{$`cb13-6`, `wp13-6`, `cb6-13`, `wp6-13`$\}$ results in the level 3 automaton $G^{(3)}_{\text{pc2}}$ with 4 states as presented in Figure 5.31.

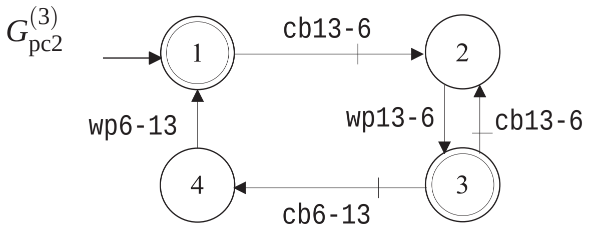


Figure 5.31: Level 3 model of the production cell pc2

Just as the production cell pc1, also pc2 is represented as a level 3 model. Note that all projected control systems involved in the hierarchy of pc2 are locally nonblocking and marked string accepting. In addition to that, all low-level supervisors are consistent implementations of live high-level supervisors. Thus, it is guaranteed that the overall production cell is nonblocking according to Theorem 3.1. The complete hierarchical architecture of the production cell pc2 is shown in Figure 5.32, and the sizes of the different automata are listed in Table 5.5.

Level 0

$G^{(0)}_{\text{cb6}}$	$D^{(0)}_{\text{cb6}}$	$R^{(0)}_{\text{cb6}}$	$G^{(0)}_{\text{mh2d2}}$	$D^{(0)}_{\text{mh2d2}}$	$R^{(0)}_{\text{mh2d2}}$	$G^{(0)}_{\text{cb14}}$	$D^{(0)}_{\text{cb14}}$	$R^{(0)}_{\text{cb14}}$
18	18	18	20	12	12	18	18	18
$G^{(0)}_{\text{rt2}}$	$D^{(0)}_{\text{rt2}}$	$R^{(0)}_{\text{rt2}}$	$G^{(0)}_{\text{cb10}}$	$D^{(0)}_{\text{cb10}}$	$R^{(0)}_{\text{cb10}}$			
10	12	12	18	10	10			

Level 1

$G^{(1)}_{\text{cb6}}$	$G^{(1)}_{\text{mh2d2}}$	$G^{(1)}_{\text{cb14}}$	$G^{(1)}_{\text{rt2}}$	$G^{(1)}_{\text{cb10}}$	$G^{(1)}_{\text{cb6mh2d2}}$	$D^{(1)}_{\text{cb6mh2d2}}$	$R^{(1)}_{\text{cb6mh2d2}}$	$G^{(1)}_{\text{rt2cb14}}$
6	2	6	4	4	12	4	7	16
$D^{(1)}_{\text{rt2cb14}}$	$R^{(1)}_{\text{rt2cb14}}$							
4	8							

Level 2 **Level 3**

$G^{(2)}_{\text{cb6mh2d2}}$	$G^{(2)}_{\text{rt2cb14}}$	$G^{(2)}_{\text{cb10}}$	$G^{(2)}_{\text{pc2}}$	$D^{(2)}_{\text{pc2}}$	$R^{(2)}_{\text{pc2}}$	$G^{(3)}_{\text{pc2}}$		
5	4	4	31	3	25	9		

Table 5.5: State numbers of the automata forming the production cell pc2

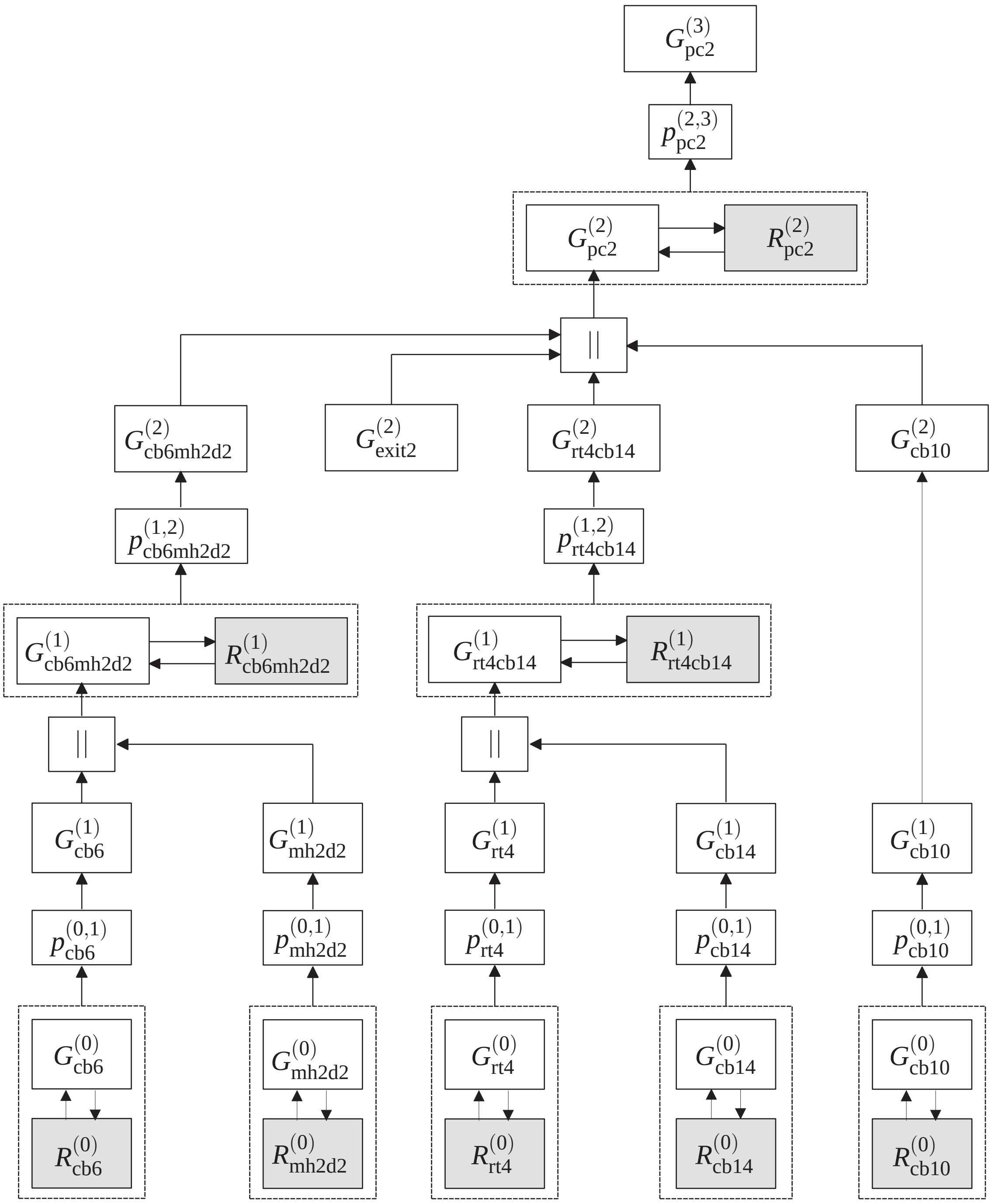


Figure 5.32: Hierarchical architecture for the production cell pc2

5.4.5 Interchange System ics

The interchange system consists of the rotary tables rt2 and rt3 with the conveyor belts cb12 and cb13, respectively, the conveyor belt cb5 and the conveyor belt cb9. It is the most complex component of the manufacturing system, as it allows transportation of workpieces from the distribution system to all other components of the manufacturing system as well as exchange of workpieces between the different components.[7]

cb12: The conveyor belt cb12 is allowed to receive workpieces from cb3 or cb5, and it can deliver workpieces only to cb4. The level 1 automaton $G^{(1)}_{\mathrm{cb12}}$ has 5 states.

rt2: The rotary table rt2 has the same functionality as rt1. It can rotate clockwise (from x to y) and counterclockwise (from y to x). On level 1, the automaton $G^{(1)}_{\mathrm{rt2}}$ has 4 states.

rt2cb12: For the composition $G^{(1)}_{\mathrm{rt2cb12}} = G^{(1)}_{\mathrm{rt2}}||G^{(1)}_{\mathrm{cb12}}$ of the rotary table rt2 with the conveyor belt cb12, the following specifications are required.

- rt2 is only allowed to move if cb2 does not move and vice versa.
- `cb3-12` must only happen if the rotary table is oriented in the y direction.
- `cb5-12` and `cb12-4` must only occur if the rotary table points in the x direction.
- workpieces are accepted from cb3 and cb5 and are delivered to cb4.

After synthesizing a supervisor for this specification and projecting to level 2, the automaton $G^{(2)}_{\mathrm{rt2cb12}}$ has 6 states. It is shown in Figure 5.33.

cb5: The conveyor belt cb5 gets workpieces from cb13 and delivers them to cb12. (see $G^{(2)}_{\mathrm{cb5}}$ in Figure 5.33).

[7]Note that the conveyor belt cb8 is not used in this example as is not needed for the specified operation.

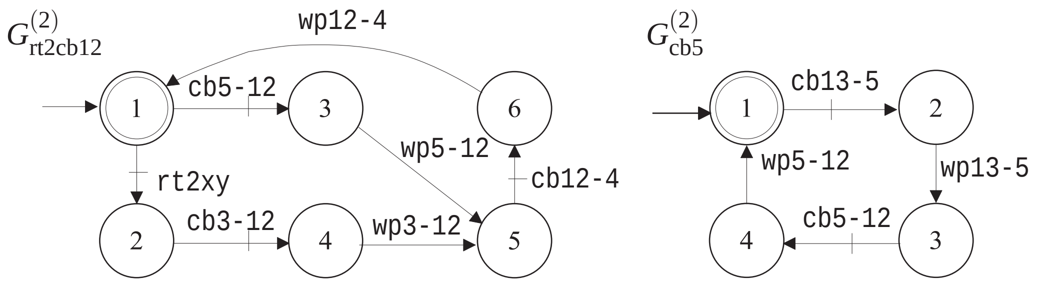


Figure 5.33: Level 2 automata $G^{(2)}_{\text{rt2cb12}}$ and $G^{(2)}_{\text{cb5}}$ of rt2cb12 and cb5

cb13: The conveyor belt cb13 can receive workpieces from cb2, cb6 or cb9 and deliver workpieces to cb6 or cb5. The level 1 model $G^{(1)}_{\text{cb13}}$ has 7 states.

rt3: The rotary table rt3 behaves analogously to rt2.

rt3cb13: The automaton $G^{(1)}_{\text{rt3cb13}} = G^{(1)}_{\text{rt3cb13}}||G^{(1)}_{\text{rt3cb13}}$ yields 32 states. The specifications for this component are

- rt3 can only move if cb13 does not move.
- workpieces can be received from cb9 and or cb2 if the rotary table is in the y position.
- workpieces can be delivered to cb5 or cb6 or received from cb6 only if rt3 is in the x position.
- workpieces which arrive from cb6 have to be transported to cb5
- workpieces coming from cb2 or cb9 must be delivered to cb6.

The level 2 automaton after implementing this specification is $G^{(2)}_{\text{rt3cb13}}$ with 9 states.

cb9: The conveyor belt cb9 receives workpieces from cb15 and delivers them to cb13.

Figure 5.34 shows the automata $G^{(2)}_{\text{rt3cb13}}$ and $G^{(2)}_{\text{cb9}}$.

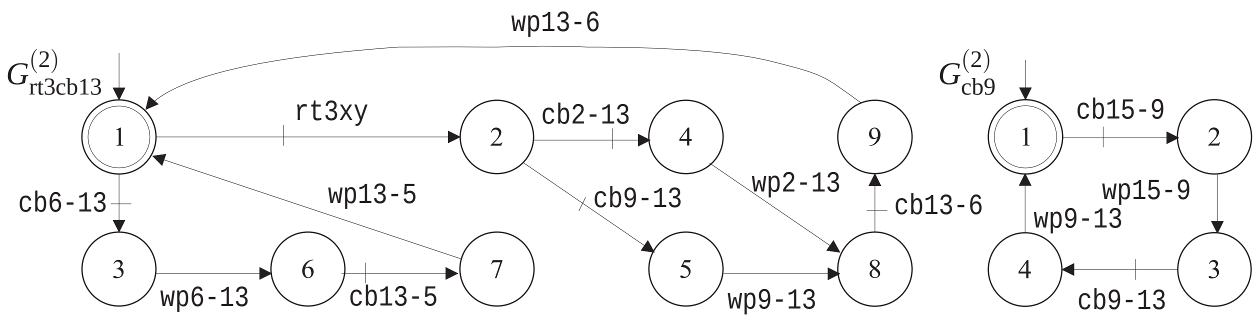


Figure 5.34: Level 2 automata of rt3cb13 and cb9

is: The overall interchange system $G^{(2)}_{\text{ics}} = G^{(2)}_{\text{rt2cb12}}||G^{(2)}_{\text{rt3cb13}}||G^{(2)}_{\text{cb5}}||G^{(2)}_{\text{cb9}}$ has 93 states. It is desired that

- workpieces which come from cb2 go to cb6, come back to cb13 and are delivered to cb5.
- workpieces which come from cb9 go to cb6 and do not come back.

Applying the specification and projecting to level 3, the automaton $G^{(3)}_{\text{ics}}$ has 50 states.

As all decentralized projected control systems in the hierarchy of the interchange system are locally nonblocking, marked string accepting and marked string controllable, it is possible to use $G^{(3)}_{\text{ics}}$ as a decentralized component of the overall manufacturing system.

Level 0

$G^{(0)}_{\text{cb12}}$	$D^{(0)}_{\text{cb12}}$	$R^{(0)}_{\text{cb12}}$	$G^{(0)}_{\text{rt2}}$	$D^{(0)}_{\text{rt2}}$	$R^{(0)}_{\text{rt2}}$	$G^{(0)}_{\text{cb13}}$	$D^{(0)}_{\text{cb13}}$	$R^{(0)}_{\text{cb13}}$
34	14	14	10	12	12	34	23	23
$G^{(0)}_{\text{rt3}}$	$D^{(0)}_{\text{rt3}}$	$R^{(0)}_{\text{rt3}}$	$G^{(0)}_{\text{cb5}}$	$D^{(0)}_{\text{cb5}}$	$R^{(0)}_{\text{cb5}}$	$G^{(0)}_{\text{cb9}}$	$D^{(0)}_{\text{cb9}}$	$R^{(0)}_{\text{cb9}}$
10	12	12	18	10	10	26	10	10

Level 1

$G^{(1)}_{\text{cb12}}$	$G^{(1)}_{\text{rt2}}$	$G^{(1)}_{\text{cb13}}$	$G^{(1)}_{\text{rt3}}$	$G^{(1)}_{\text{cb5}}$	$G^{(1)}_{\text{cb9}}$	$G^{(1)}_{\text{rt2cb12}}$	$D^{(1)}_{\text{rt2cb12}}$	$R^{(1)}_{\text{rt2cb12}}$
5	4	7	4	4	4	20	5	9
$G^{(1)}_{\text{rt3cb13}}$	$D^{(1)}_{\text{rt3cb13}}$	$R^{(1)}_{\text{rt3cb13}}$						
32	5	12						

Level 2 **Level 3**

$G^{(2)}_{\text{rt2cb12}}$	$G^{(2)}_{\text{rt3cb13}}$	$G^{(2)}_{\text{ics}}$	$D^{(2)}_{\text{ics}}$	$R^{(2)}_{\text{ics}}$	$G^{(3)}_{\text{ics}}$			
6	9	93	5	81	50			

Table 5.6: State numbers of the automata of the interchange system ics

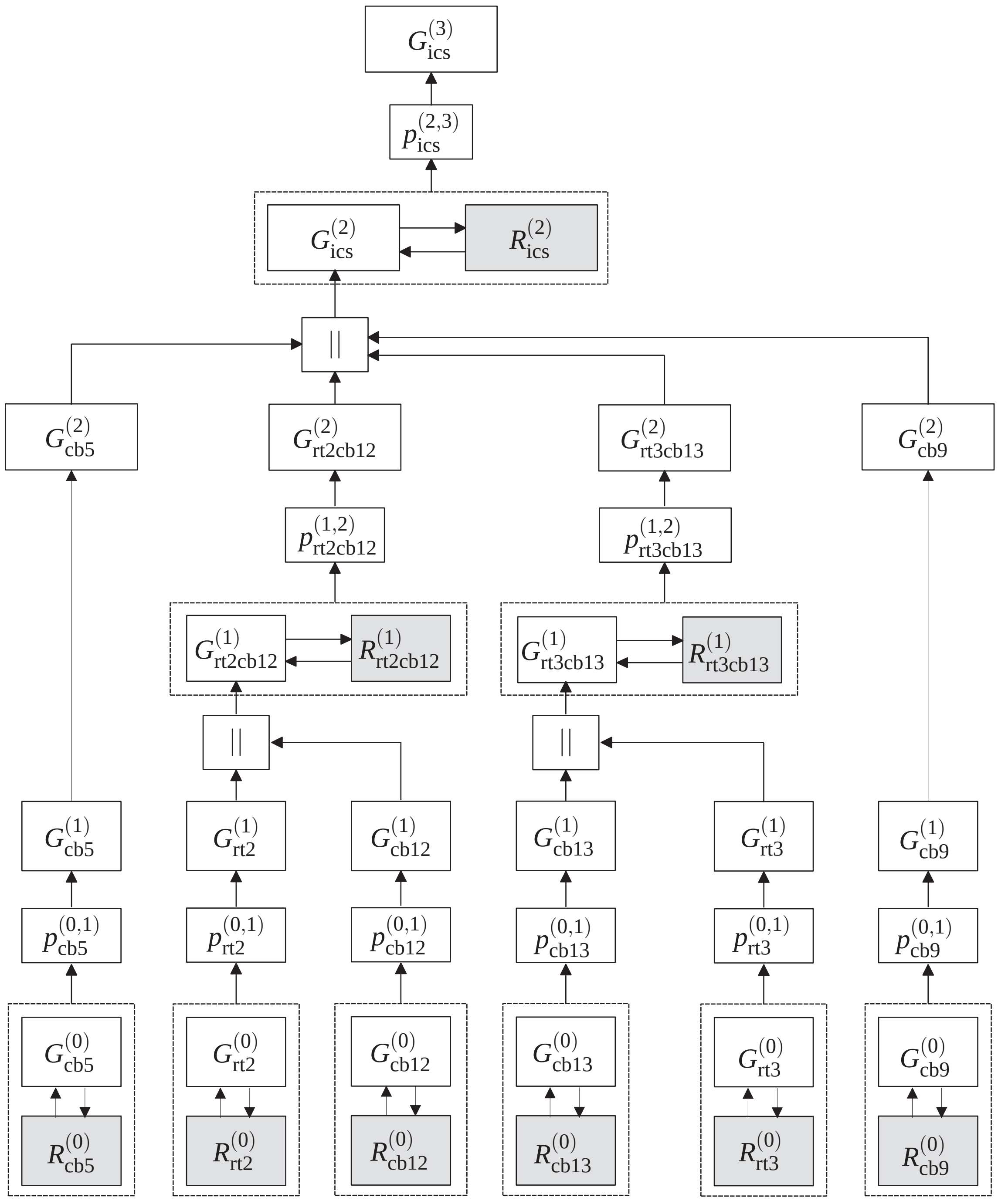


Figure 5.35: Hierarchical architecture for the interchange system is

5.4.6 Overall Manufacturing System

After synthesizing supervisors for all decentralized components of the manufacturing system ms, supervisory control is applied to the overall system on level 3. The resulting automaton $G_{\text{ms}}^{(3)} = G_{\text{dist}}^{(3)}||G_{\text{ics}}^{(3)}||G_{\text{pc1}}^{(3)}||G_{\text{pc2}}^{(3)}||G_{\text{exit1}}^{(3)}$ has 863 states. The following functionality is specified for the global plant:

- workpieces coming from cb3 have to go to cb9 via the rail transport system rts1.
- workpieces coming from cb5 have to go to the roll conveyor rc1.

The specification determines the order in which workpieces have to be distributed by exit1 after their arrival from different directions at rt2cb12. The automaton realization of the specification $D_{\text{ms}}^{(3)}$ has 7 states and is depicted in Figure 5.36.

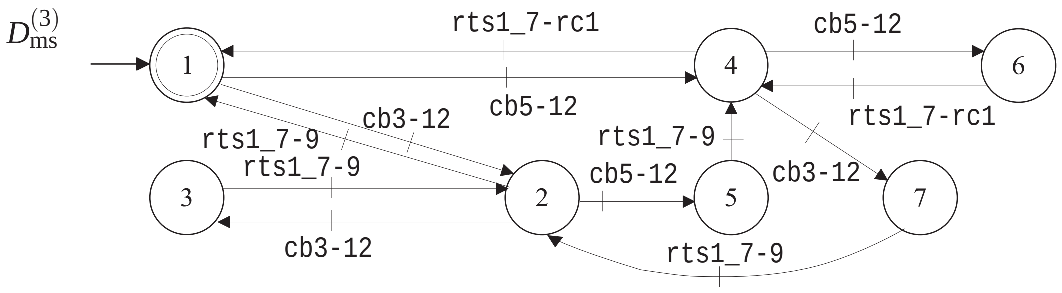


Figure 5.36: Level 3 specification automaton $D_{\text{ms}}^{(3)}$ for the manufacturing system

The supremal controllable sublanguage $\kappa_{L_m(D_{\text{ms}}^{(3)})}\left(L_m(G_{\text{ms}}^{(3)})\right)$ is recognized by the canonical recognizer $R_{\text{ms}}^{(3)}$ with 4700 states.

$G_{\text{ms}}^{(3)}$	$D_{\text{ms}}^{(3)}$	$R_{\text{ms}}^{(3)}$
863	7	4700

Table 5.7: State numbers for the manufacturing system on level 3

The supervisor $R_{\text{ms}}^{(3)}$ handles the coordination of tasks for the overall system, such as the cooperation of the decentralized system components. The overall plant is a composition of nonblocking components, which were derived by successive application of the hierarchical and decentralized control method presented in Chapter 4.

Globally, the closed-loop functionality of the manufacturing system is as follows.

- the distribution system provides workpieces via the conveyor belts cb2 and cb3. There are always 2 workpieces coming from cb3 before one workpiece is delivered by cb2.
- workpieces arriving from cb2 are processed by the machine mh2d2, sent to machine mh1d1 via cb5 and then unloaded on the roll conveyor rc1.
- workpieces coming from cb3 are first processed by machine mh1d1, then transported to mh2d2 via rts1 and cb9 and finally unloaded on the roll conveyor rc2 via rt2s.

The path of the two different types of workpieces is illustrated in Figure 5.37.

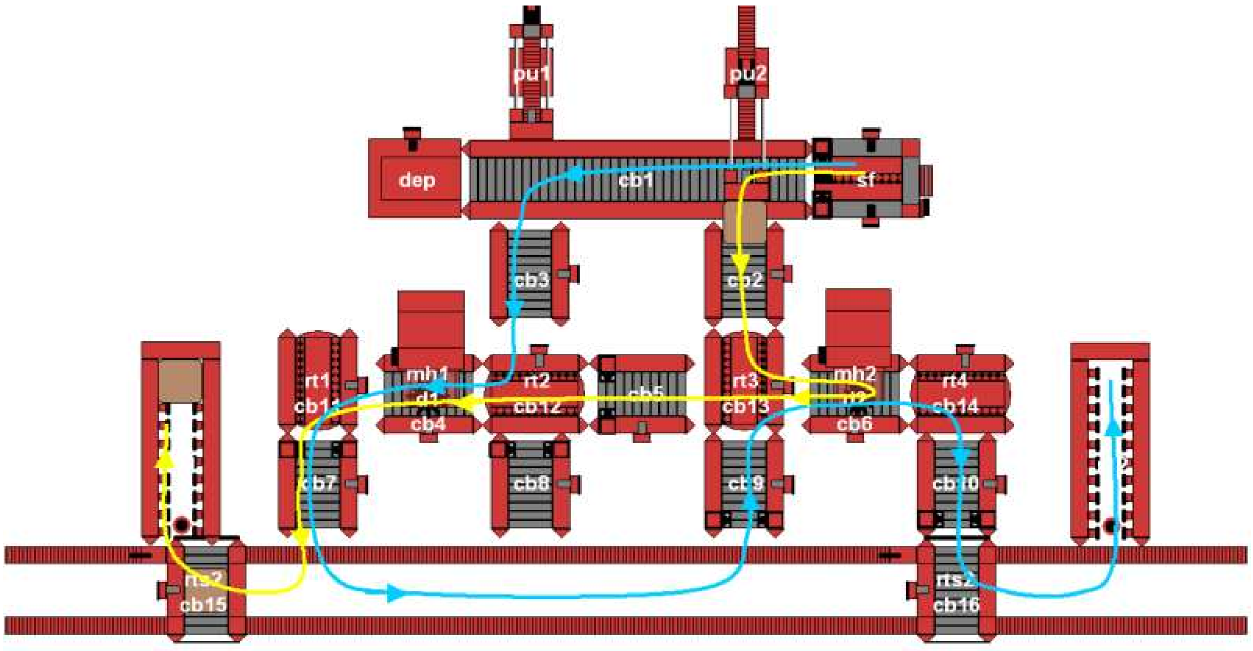


Figure 5.37: Flow of workpieces specified for the manufacturing system

Note that the paths cross at the rotary tables rt2 and rt3. This potential interaction of workpieces is the reason why blocking is possible in the manufacturing system. Yet, because of Theorem 4.2, the hierarchical and decentralized supervisor synthesis guarantees that situations where blocking can occur are avoided.

5.5 Summary

For the manufacturing system, supervisor synthesis on 3 levels has been carried out. The system is composed of 28 components on the low level, and an estimated monolithic low-level model reaches 10^{24} states, while the state count of the decentralized models on level 0 adds up to 517. A monolithic supervisor for the manufacturing system would have an estimated number of 10^{30} states. Both computation and implementation of the supervisors are infeasible for this scale.

Synthesis in the hierarchical and decentralized framework results in 39 decentralized supervisors on 3 levels. Altogether, they can be implemented in parallel with a total number of 5388 states. Correct operation of the manufacturing system could be determined after generating PLC-code from the hierarchical and decentralized supervisors ([Fig05]).

Chapter 6

Conclusions

The supervisory control of large scale composed discrete event sytems (DES) involves computations on states spaces which grow exponentially with the number of system components. In this dissertation, a method for exploiting the decentralized structure of composed systems in combination with a hierarchical abstraction is presented.

The mathematical concept of *formal languages*, which is used for describing the dynamical behavior of discrete event systems, is outlined in Chapter 2. For the sake of clarity, the basic ideas of the Ramadge/Wonham (RW) supervisoy control theory are elaborated in a pure language framework. The link to the equivalent automata formulation is also established.

In Chapter 3, our centralized hierarchical approach is developed. It involves abstracting a discrete event system to a smaller high-level model, performing supervisory control for the high-level system, and computing a low-level implementation of the high-level supervisor. The *hierarchical closed loop system* captures this hierarchical architecture. The *natural projection* is used for the system abstraction, and high-level supervisors are translated to the low level via the *consistent implementation*. On all levels of the hierarchy, the RW framework is employed. Our hierarchical approach can be applied if the system is *locally nonblocking* and *marked string accepting*. If these structural conditions are fulfilled, then either *liveness* of the high-level closed loop behavior or *marked string controllability* of the low-level system guarantee *hierarchically consistent* and *nonblocking* control.

The chapter also provides algorithms for the verification of the structural conditions. They are based on an automata representation of the low-level discrete event system. It is important to note, that the overall low-level model of the system has to be considered for computing the high-level model.

Addressing this issue, Chapter 4 extends our monolithic hierarchical approach to *decentralized discrete event systems*. Each of the subsystems is considered as a hierarchical closed loop system, and the hierarchical abstraction captures the shared behavior of the subsystems. Analogously to the consistent implementation for the centralized case, a *decentralized consistent implementation* realizes the control action on the low level. If the same conditions as in Chapter 3 are fulfilled for all subsystems, then hierarchically consistent and nonblocking control is guaranteed. The crucial advantage of the decentralized approach is that the overall low-level system need not be computed. It is possible to first abstract the low-level subsystems and then compose the high-level model, which reduces the computational effort tremendously. In addition to that, decentralized low-level supervisors for the different subsystems are implemented instead of one low level supervisor for the overall system, and they are coordinated by the high-level supervisor.

The computational benefit of our method is illustrated by a large scale example in Chapter 5. The manufacturing system used in the example comprises 28 components, and it has an estimated number of 10^{24} states. We synthesize 39 decentralized supervisors by using our hierarchical and decentralized method with 4 levels of abstraction and control. They can be implemented individually with an average number of 140 states. In comparison, for a monolithic implementation, we would expect a supervisor of order 10^{30} states.

There are two alternative supervisory control approaches which can handle large scale discrete event systems. The method presented in [Led02] is based on a *client server architecture*. It is applied to the "Atelier Inter-établissement de Productique" (AIP) example of order 10^{21} states. In [Ma04], a top-down view on large discrete event systems is employed. A supervisor for a version of the AIP example with 10^{24} states could be computed ([MW03]) by using *state tree structures* for structured system modeling. Our laboratory case study (Chapter 5) is of a similar scale, and our results compare well with [Led02, Ma04].

Appendix A

Proofs

The appendix provides proofs of several lemmas and theorems that were omitted in the previous chapters.

A.1 Projection of a Regular Language

Lemma 2.3 (Projection of a Regular Language)
Let $L \subseteq \Sigma^*$ be a regular language and let $p_0 : \Sigma^* \longrightarrow \Sigma_0^*$ with $\Sigma_0 \subseteq \Sigma$ be the natural projection. Then $p_0(L)$ is regular.

Proof: **(Outline)** Section 3.4.1 provides an algorithm which computes a deterministic finite automaton recognizing the projection of a regular language. Thus, the projection of a regular language is also regular. $\square$

A.2 Computation of the Projection

A.2.1 Space Complexity

Theorem 3.3 (Space complexity of the natural projection)
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a marked string accepting and locally nonblocking projected system with the automata representation (G, G^{hi}). Then, G^{hi} has an equal or smaller number of states as G, i.e. $|X^{\mathrm{hi}}| \leq |X|$.[1]

The result is derived from a theorem in [Won97]. The following terms are used in this theorem.

[1]Note that both G and G^{hi} are canonical recognizers.

Definition A.1 (Causal Reporter Map [ZW90])
Let $L \subseteq \Sigma^*$ and let Σ^{hi} be another event alphabet. A map $\theta : \overline{L} \rightarrow (\Sigma^{\mathrm{hi}})^*$ is a causal reporter map, if

$$\begin{aligned} \theta(\varepsilon) &= \varepsilon, \\ \theta(s\sigma) &= \begin{cases} \text{either } \theta(s) \\ \text{or } \theta(s)\sigma^{\mathrm{hi}}, \text{ for some } \sigma^{\mathrm{hi}} \in \Sigma^{\mathrm{hi}}. \end{cases} \end{aligned}$$

□

Definition A.2 (K-Observer)
Let $\theta : \Sigma^* \rightarrow (\Sigma^{\mathrm{hi}})^*$ be a causal reporter map, and let L, K be languages with $K \subseteq L \subseteq \Sigma^*$. θ is a K-observer for L, iff it holds that for arbitrary $s \in \overline{L}$ and $t \in (\Sigma^{\mathrm{hi}})^*$

$$\theta(s)t \in \theta(K) \Rightarrow \exists u \in \Sigma^* \text{ s.t. } su \in K \text{ and } \theta(su) = \theta(s)t. \tag{A.1}$$

□

In the next lemma, we show that the projection to the high-level event set constitutes an observer for marked string accepting and locally nonblocking projected control systems.

Lemma A.1
Let $P = (H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a marked string accepting and locally nonblocking projected control system with a nonblocking control system H. Then p^{hi} is a L_2-observer for L_1. □

Proof: First note that p^{hi} is a causal reporter map and that $L_2 \subseteq L_1$. Equation A.1 has to be verified. Assume that $s \in L_1$, $s^{\mathrm{hi}} := p^{\mathrm{hi}}(s)$ and $t \in (\Sigma^{\mathrm{hi}})^*$ s.t. $s^{\mathrm{hi}}t \in L_2^{\mathrm{hi}} = p^{\mathrm{hi}}(L_2)$. t is represented as $t = \sigma_0\sigma_1 \cdots \sigma_m$ with $\sigma_0 = \varepsilon$ and $\sigma_i \in \Sigma^{\mathrm{hi}}$ for $i = 1, \ldots, m$. It can be shown that there is a $u' = u_0\sigma_0u_1 \cdots u_m\sigma_m \in \Sigma^*$ with $u_i \in (\Sigma - \Sigma^{\mathrm{hi}}$, $i = 1, \ldots, m$ s.t. $su' \in L_1$, by induction. The base case is true as $su_0\sigma_0 = s \in L_1$. Now assume that $su_0\sigma_0u_1 \cdots u_i\sigma_i \in L_1$ for $i < m$. As P is locally nonblocking, there exists a $u_{i+1} \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ s.t. $su_0\sigma_0 \cdots \sigma_iu_{i+1}\sigma_{i+1} \in L_1$ (Definition 3.9). As this is true for all $i < m$, it holds that there is a $u' = u_0\sigma_0u_1 \cdots u_m\sigma_m \in \Sigma^*$ s.t. $su' \in L_1$, $p^{\mathrm{hi}}(su') = p^{\mathrm{hi}}(s)t$, and because of the construction of u', $su' \in L_{\mathrm{en},s^{\mathrm{hi}}t}$.

Now it has to be shown that there is a local string $u'' \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ s.t. $su'u'' \in L_2$. There are two cases. First consider the case, where $\Sigma^{\mathrm{hi}}(s^{\mathrm{hi}}) = \emptyset$. As H is nonblocking, there must be such u''. Secondly, let $\Sigma^{\mathrm{hi}}(s^{\mathrm{hi}}) \neq \emptyset$ and choose $\sigma \in \Sigma^{\mathrm{hi}}(s^{\mathrm{hi}})$. Then, there is a $\tilde{u} \in (\Sigma - \Sigma^{\mathrm{hi}})^*$ s.t. $su'\tilde{u}\sigma \in L_1$, because P is locally nonblocking. Considering that P is also marked string accepting, there exists a $u'' \leq \tilde{u}$ s.t. $su'u'' \in L_2$. Hence, in all cases there is a string $u = u'u''$ s.t. $su \in L_2$ and $p^{\mathrm{hi}}(su) = p^{\mathrm{hi}}(s)p^{\mathrm{hi}}(u) = p^{\mathrm{hi}}(s)t$. □

The result from [Won97] is recalled.

Lemma A.2 ([Won97])
Let G be a minimal, trim[2], finite generator with Σ as its event alphabet. Let $\Sigma^{\mathrm{hi}} \subseteq \Sigma$ and p^{hi} be the corresponding natural projection. Suppose that p^{hi} is an $L_{\mathrm{m}}(G)$-observer for $L(G)$. Then the number of states of the canonical recognizer of $p^{\mathrm{hi}}(L_{\mathrm{m}}(G))$ is less or equal to the number of states of G. $\square$

We now prove Theorem 3.3 by relating Lemmas A.1 and A.2.

Proof: An automata representation G of the control system H in Lemma A.1 is finite and trim because of Lemma 2.9. As $L_{\mathrm{m}}(G) = L_2$ and $L(G) = L_1$, it holds that p^{hi} is an $L_{\mathrm{m}}(G)$-observer of $L(G)$, and thus Lemma A.2 can be applied. $\square$

A.2.2 Time Complexity

Theorem 3.4 (Time complexity of the natural projection)
Let $(H, p^{\mathrm{hi}}, H^{\mathrm{hi}})$ be a marked string accepting and locally nonblocking projected system with the automata representation (G, G^{hi}). The time complexity of computing G^{hi} is at worst polynomial in the state size of G and the number of high-level events Σ^{hi}.

The corresponding result from [Won97] is given in the following lemma.

Lemma A.3 ([Won97])
Let G be a trim, finite generator with Σ as its event alphabet. Let $\Sigma^{\mathrm{hi}} \subseteq \Sigma$ and p^{hi} be the corresponding natural projection. Suppose that p^{hi} is an $L_{\mathrm{m}}(G)$-observer for $L(G)$. Then the time complexity of computing a generator for $p^{\mathrm{hi}}(L_{\mathrm{m}}(G))$ is at worst polynomial (in terms of the size of G and the size of Σ^{hi}). $\square$

We relate Lemmas A.1 and A.3 for showing Theorem 3.4.

Proof: An automata representation G of the control system H in Lemma A.1 is finite and trim because of Lemma 2.9. As $L_{\mathrm{m}}(G) = L_2$ and $L(G) = L_1$, it holds that p^{hi} is an $L_{\mathrm{m}}(G)$-observer of $L(G)$, and thus Lemma A.3 can be applied. $\square$

[2]A generator G is trim if $\overline{L_m(G)} = L(G)$.

A.3 Computation of the High-Level Plant

Proposition 4.1 (High Level Plant [SRM04, SMP05])
Let $(||_{i=1}^{n} H_i, p^{\mathrm{hi}}, ||_{i=1}^{n} H_i^{\mathrm{hi}})$ be a projected decentralized control system. Then the high level control system is $H^{\mathrm{hi}} = p^{\mathrm{hi}}(||_{i=1}^{n} H_i) = ||_{i=1}^{n} p_i^{\mathrm{dec}}(H_i)$.

Lemma A.4 is used for proving Proposition 4.1.

Lemma A.4
Let $L_1 \subseteq \Sigma_1^*, \ldots, L_n \subseteq \Sigma_n^*$ be languages over the alphabets $\Sigma_1, \ldots, \Sigma_n$. Assume that $\Sigma_0 \subseteq (\Sigma_1 \cup \cdots \cup \Sigma_n)$ and $\bigcup_{i,j,i \neq j}^{n} (\Sigma_i \cap \Sigma_j) \subseteq \Sigma_0$ with the natural projections $p_0 : (\Sigma_1 \cup \cdots \cup \Sigma_n)^* \to \Sigma_0^*$ and $p_i' : \Sigma_i^* \to (\Sigma_i \cap \Sigma_0)^*$, $i = 1, \ldots, n$. Then

$$p_0(L_1||\cdots||L_n) = p_1'(L_1)||\cdots||p_n'(L_n).$$

□

We use a result from [Won04, dQ00] for proving Lemma A.4.

Lemma A.5
Let Σ_{a} and Σ_{b} be alphabets and let $L_{\mathrm{a}} \subseteq \Sigma_{\mathrm{a}}^*$ and $L_{\mathrm{b}} \subseteq \Sigma_{\mathrm{b}}^*$. Assume $\Sigma_0 \subseteq \Sigma_{\mathrm{a}} \cup \Sigma_{\mathrm{b}}$ and $\Sigma_{\mathrm{a}} \cap \Sigma_{\mathrm{b}} \subseteq \Sigma_0$ with the natural projections $p_0 : (\Sigma_{\mathrm{a}} \cup \Sigma_{\mathrm{b}})^* \to \Sigma_0^*$, $p_{\mathrm{a}}' : \Sigma_{\mathrm{a}}^* \to (\Sigma_0 \cap \Sigma_{\mathrm{a}})^*$ and $p_{\mathrm{b}}' : \Sigma_{\mathrm{b}}^* \to (\Sigma_0 \cap \Sigma_{\mathrm{b}})^*$. Then $p_0(L_{\mathrm{a}}||L_{\mathrm{b}}) = p_{\mathrm{a}}'(L_{\mathrm{a}})||p_{\mathrm{b}}'(L_{\mathrm{b}})$. □

Proof: Several natural projections on different alphabets are needed in this proof. For convenience they are listed below.

$$\begin{array}{ll}
p_0 : (\Sigma_{\mathrm{a}} \cup \Sigma_{\mathrm{b}})^* \to \Sigma_0^* & \\
p_{\mathrm{a}}' : \Sigma_{\mathrm{a}}^* \to (\Sigma_{\mathrm{a}} \cap \Sigma_0)^* & p_{\mathrm{b}}' : \Sigma_{\mathrm{b}}^* \to (\Sigma_{\mathrm{b}} \cap \Sigma_0)^* \\
p_{\mathrm{a}} : (\Sigma_{\mathrm{a}} \cup \Sigma_{\mathrm{b}})^* \to \Sigma_{\mathrm{a}}^* & p_{\mathrm{b}} : (\Sigma_{\mathrm{a}} \cup \Sigma_{\mathrm{b}})^* \to \Sigma_{\mathrm{b}}^* \\
p_{0,\mathrm{a}} : \Sigma_0^* \to (\Sigma_{\mathrm{a}} \cap \Sigma_0)^* & p_{0,\mathrm{b}} : \Sigma_0^* \to (\Sigma_{\mathrm{b}} \cap \Sigma_0)^*
\end{array}$$

First, $p_0(L_{\mathrm{a}}||L_{\mathrm{b}}) \subseteq p_{\mathrm{a}}'(L_{\mathrm{a}})||p_{\mathrm{b}}'(L_{\mathrm{b}})$ is shown. Assume $t \in p_0(L_{\mathrm{a}}||L_{\mathrm{b}})$. Then there exists a $s \in L_{\mathrm{a}}||L_{\mathrm{b}}$, s.t. $p_0(s) = t$ and also $p_{\mathrm{a}}(s) \in L_{\mathrm{a}}$ and $p_{\mathrm{b}}(s) \in L_{\mathrm{b}}$. Consequently, $p_{\mathrm{a}}'(p_{\mathrm{a}}(s)) \in p_{\mathrm{a}}(L_{\mathrm{a}})$ and $p_{\mathrm{b}}'(p_{\mathrm{b}}(s)) \in p_{\mathrm{b}}(L_{\mathrm{b}})$. Observing that $p_{0,\mathrm{a}}(t) = p_{0,\mathrm{a}}(p_0(s)) = p_{\mathrm{a}}'(p_{\mathrm{a}}(s)) \in p_{\mathrm{a}}'(L_{\mathrm{a}})$ and $p_{0,\mathrm{b}}(t) = p_{0,\mathrm{b}}(p_0(s)) = p_{\mathrm{b}}'(p_{\mathrm{b}}(s)) \in p_{\mathrm{b}}'(L_{\mathrm{b}})$, it holds that $t \in (p_{0,\mathrm{a}})^{-1}\big(p_{\mathrm{a}}'(p_{\mathrm{a}}(s))\big) \cap (p_{0,\mathrm{b}})^{-1}\big(p_{\mathrm{b}}'(p_{\mathrm{b}}(s))\big) \subseteq (p_{0,\mathrm{a}})^{-1}\big(p_{\mathrm{a}}'(L_{\mathrm{a}})\big) \cap (p_{0,\mathrm{b}})^{-1}\big(p_{\mathrm{b}}'(L_{\mathrm{b}})\big) = p_{\mathrm{a}}'(L_{\mathrm{a}})||p_{\mathrm{b}}'(L_{\mathrm{b}})$.

Now, $p_{\mathrm{a}}'(L_{\mathrm{a}})||p_{\mathrm{b}}'(L_{\mathrm{b}}) \subseteq p_0(L_{\mathrm{a}}||L_{\mathrm{b}})$ is proven. Let $t \in p_{\mathrm{a}}'(L_{\mathrm{a}})||p_{\mathrm{b}}'(L_{\mathrm{b}})$. Then $p_{0,\mathrm{a}}(t) \in p_{\mathrm{a}}'(L_{\mathrm{a}})$ and $p_{0,\mathrm{b}}(t) \in p_{\mathrm{b}}'(L_{\mathrm{b}})$. Thus, there exists a $s_{\mathrm{a}} \in L_{\mathrm{a}}$ and a $s_{\mathrm{b}} \in L_{\mathrm{b}}$ s.t. $p_{\mathrm{a}}'(s_{\mathrm{a}}) = p_{0,\mathrm{a}}(t)$ and $p_{\mathrm{b}}'(s_{\mathrm{b}}) = p_{0,\mathrm{b}}(t)$. Using the fact that $\Sigma_{\mathrm{a}} \cap \Sigma_{\mathrm{b}} \subseteq \Sigma_0$, there is also a string $s \in L_{\mathrm{a}}||L_{\mathrm{b}}$ s.t. $p_{\mathrm{a}}(s) = s_{\mathrm{a}}$ and $p_{\mathrm{b}}(s) = s_{\mathrm{b}}$. It holds that $s \in (p_{\mathrm{a}})^{-1}\big((p_{\mathrm{a}}')^{-1}\big(p_{0,\mathrm{a}}(t)\big)\big)||(p_{\mathrm{b}})^{-1}\big((p_{\mathrm{b}}')^{-1}\big(p_{0,\mathrm{b}}(t)\big)\big)$. Hence, $p_0(s) = t$ and with $s \in L_{\mathrm{a}}||L_{\mathrm{b}}$ it follows that $t \in p_0(L_{\mathrm{a}}||L_{\mathrm{b}})$. □

Now Lemma A.4 is shown by induction.

Proof: Note that the same notation as in the proof of Lemma A.5 is used for the natural projections. In addition to that, the following notation is introduced.

$$\begin{aligned}&\Sigma'_k := \textstyle\bigcup_{i=1}^{k} \Sigma_i,\ k = 1,\dots,n\\ &\Sigma_{0,k} := \Sigma_0 \cap \Sigma'_k,\ k = 1,\dots,n \qquad p_{0,k} : (\Sigma'_k)^* \to \Sigma_{0,k}^*\end{aligned}$$

It has to be shown that $p_{0,2}(L_1||L_2) = p'_1(L_1)||p'_2(L_2)$ for the base case. Observing that $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_{0,2}$, Lemma A.5 provides the desired result for $\Sigma_\mathrm{a} = \Sigma_1$ and $\Sigma_\mathrm{b} = \Sigma_2$. For the induction step, assume that $p_{0,l-1}(L_1||\cdots||L_{l-1}) = p'_1(L_1)||\cdots p'_{l-1}(L_{l-1})$. We show that also $p_{0,l}(L_1||\cdots||L_{l-1}||L_l) = p'_1(L_1)||\cdots p'_{l-1}(L_{l-1})||p'_l(L_l)$. Grouping $L_1,\dots,L_l$, we define $\Sigma_\mathrm{a} := \Sigma'_{l-1}$, $\Sigma_\mathrm{b} := \Sigma_l$, $L_\mathrm{a} := L_1||\cdots L_{l-1}$ and $L_\mathrm{b} := L_l$. Also note that $\Sigma'_{l-1} \cap \Sigma_l \subseteq \Sigma_{0,l}$. Using this terminology, Lemma A.5 can directly be applied. It holds that $p_{0,l}\big((L_1||\cdots||L_{l-1})||L_l\big) = p_{0,l-1}(L_1||\cdots||L_{l-1})||p_l(L_l)$ and with the induction assumption $p_{0,l}(L_1||\cdots||L_{l-1}||L_l) = p'_1(L_1)||\cdots p'_{l-1}(L_{l-1})||p'_l(L_l)$. As $2 \le l \le n$ was arbitrary, Lemma A.4 follows. □

The projected decentralized control system is defined such that the high-level alphabet is a superset of the shared events of the decentralized subsystems. Observing this, it is clear that for the languages $L_{i,1}$ and $L_{i,2}$ of the decentralized subsystems, Lemma A.4 can be applied. This concludes the proof of Proposition 4.1.

Proof: Proposition 4.1 follows directly by applying Lemma A.4 to $L_{1,1}$, ..., $L_{n,1}$ and $L_{1,2}$, ..., $L_{n,2}$. □

A.4 Feasible Projected Decentralized Control Systems

Lemma 4.1
Let H_i, H_i^f, H_i^hi and $H_i^\mathrm{hi,f}$, $i = 1,\dots,n$ be defined as in Definition 4.3. Then

$$||_{i=1}^n H_i^\mathrm{hi} = ||_{i=1}^n H_i^\mathrm{hi,f} \ \text{ and } \ ||_{i=1}^n H_i = ||_{i=1}^n H_i^\mathrm{f}.$$

We will use the following lemma in the proof of Lemma 4.1. It states that the synchronous composition of projections of a language includes the original language.

Lemma A.6
Let $L \in \Sigma^*$ be a language and define the natural projections $p_i : \Sigma^* \to \Sigma_i^*$, $i = 1,\dots,n$, where $\Sigma_i \subseteq \Sigma$ and $\bigcup_{i=1}^n \Sigma_i = \Sigma$. Then it holds that

$$L \subseteq ||_{i=1}^n p_i(L).$$

□

Proof: Let $s \in L$. Then $s \in p_1(s)||\cdots||p_n(s) = p_1(s)||(\Sigma - \Sigma_1)^* \cap \cdots \cap p_n(s)||(\Sigma - \Sigma_n)^* \subseteq ||_{i=1}^n p_i(L)$. □

With this result, Lemma 4.1 can be shown.

Proof: At first, $||_{i=1}^n H_i^{\text{hi}} \subseteq ||_{i=1}^n H_i^{\text{hi,f}}$, i.e. $||_{i=1}^n L_{i,1}^{\text{hi}} \subseteq ||_{i=1}^n L_{i,1}^{\text{hi,f}}$ and $||_{i=1}^n L_{i,2}^{\text{hi}} \subseteq ||_{i=1}^n L_{i,2}^{\text{hi,f}}$ is proven. The result follows, oberving that $||_{i=1}^n L_{i,1}^{\text{hi}} = L_1^{\text{hi}} \subseteq ||_{i=1}^n p_i^{\text{hi}}(L_1^{\text{hi}}) = ||_{i=1}^n L_{i,1}^{\text{hi,f}}$ because of Lemma A.6. The same argument holds for the languages $L_{i,2}^{\text{hi}}$.

For the reverse direction, it can be written that

$$\begin{aligned}
||_{i=1}^n L_{i,1}^{\text{hi,f}} &= ||_{i=1}^n p_i^{\text{hi}}(L^{\text{hi}}) \\
&= ||_{i=1}^n p_i^{\text{hi}}\big(||_{k=1}^n L_{k,1}^{\text{hi}}\big) \\
&\subseteq ||_{i=1}^n \big(p_i^{\text{hi}}(L_{1,1}^{\text{hi}})||\cdots||p_i^{\text{hi}}(L_{n,1}^{\text{hi}})\big) \\
&= \big(||_{i=1}^n (p_i^{\text{hi}}(L_{1,1}^{\text{hi}}))\big)||\cdots||\big(||_{i=1}^n (p_i^{\text{hi}}(L_{n,1}^{\text{hi}}))\big) \\
&= L_{1,1}^{\text{hi}}||\cdots||L_{n,1}^{\text{hi}} \\
&= ||_{i=1}^n L_{i,1}^{\text{hi}}
\end{aligned}$$

Thus $||_{i=1}^n H_i^{\text{hi,f}} \subseteq ||_{i=1}^n H_i^{\text{hi}}$ and $||_{i=1}^n H_i^{\text{hi}} \subseteq ||_{i=1}^n H_i^{\text{hi,f}}$ and hence $||_{i=1}^n H_i^{\text{hi}} = ||_{i=1}^n H_i^{\text{hi,f}}$.

Now $||_{i=1}^n L_{i,1} = ||_{i=1}^n L_{i,1}^{\text{f}}$ shall be proven. Considering Definition 4.3, $L_{i,1}^{\text{f}} \subseteq L_{i,1}$ for all $i = 1,\ldots,n$. Thus $||_{i=1}^n L_{i,1}^{\text{f}} \subseteq ||_{i=1}^n L_{i,1}$. For showing the reverse direction, assume $s \in ||_{i=1}^n L_{i,1}$. Then $p^{\text{hi}}(s) \in L_1^{\text{hi}}$ and thus $p_i^{\text{hi}}(p^{\text{hi}}(s)) \in L_{i,1}^{\text{hi,f}}$ for all $i = 1,\ldots,n$. As $p_i(s) \in L_{i,1}$ and $p_i^{\text{dec}}(p_i(s)) = p_i^{\text{hi}}(p^{\text{hi}}(s)) \in L_{i,1}^{\text{hi,f}}$, it holds that $p_i(s) \in L_{i,1}^{\text{f}}$. It is readily observed that the same argument holds for the languages $||_{i=1}^n L_{i,2}$ and $||_{i=1}^n L_{i,2}^{\text{f}}$. Hence, $||_{i=1}^n H_i = ||_{i=1}^n H_i^{\text{f}}$. □

A.5 Mutual Controllability

Lemma 4.2
Let $(||_{i=1}^n H_i^{\text{f}}, p^{\text{hi}}, ||_{i=1}^n H_i^{\text{hi,f}})$ be a feasible projected decentralized control system and let S^{hi} be a high-level supervisor for the overall high-level system H^{hi}. If the high-level subsystems $H_i^{\text{hi,f}}$, $i = 1,\ldots,n$ are mutually controllable, i.e. $\forall i, j = 1,\ldots,n$, $i \neq j$

$$L_{j,1}^{\text{hi,f}}(\Sigma_{i,\text{uc}} \cap \Sigma_{j,\text{uc}}) \cap (p_{j,i})^{-1}\big(p_{i,j}(L_i^{\text{hi,f}})\big) \subseteq L_j^{\text{hi,f}},$$

then $p_i^{\text{hi}}(L_1^{\text{hi,c}})$ is controllable w.r.t. $L_{i,1}^{\text{hi,f}}$ for all $i = 1,\ldots,n$.

We establish the following useful properties of mutually controllable languages.

Lemma A.7
Let $L_1, L_2, \ldots, L_n$ be mutually controllable for $i, j = 1, \ldots, n$, $i \neq j$. Let $s_i \in L_i$ s.t. $s_i\sigma \in L_i$ with $\sigma \in \bigcup_{j=1, j\neq i}^{n}(\Sigma_{i,\mathrm{uc}} \cap \Sigma_{i,\mathrm{uc}})$. Then $\forall s_j \in L_j$ s.t. $p_{j,i}(s_j) = p_{i,j}(s_i)$, it holds that $s_j\sigma \in L_j$. □

Proof: Let j s.t. $\sigma \in \Sigma_j$. Because of mutual controllability, $L_j\sigma \cap (p_{j,i})^{-1}(p_{i,j}(L_i)) \subseteq L_j$. With $p_{j,i}(s_j) = p_{i,j}(s_i)$, also $p_{j,i}(s_j)\sigma = p_{i,j}(s_i\sigma) = p_{i,j}(s_i)\sigma$. Then, it holds that $s_j\sigma \in (p_{j,i})^{-1}(p_{i,j}(s_i\sigma))$ $\subseteq (p_{j,i})^{-1}(p_{i,j}(L_i))$ as $s_i\sigma \in L_i$. Thus $s_j\sigma \in L_j$. □

Lemma A.8
Let $L_1, L_2, \ldots, L_n$ be mutually controllable for $i, j = 1, \ldots, n$, $i \neq j$ and let $L := ||_{i=1}^{n} L_i$. Assume $s_i \in L_i$ and $\sigma \in \bigcup_{j=1, j\neq i}^{n}(\Sigma_{i,\mathrm{uc}} \cap \Sigma_{j,\mathrm{uc}})$ s.t. $s_i\sigma \in L_i$. Then $\forall s \in L$ s.t. $p_i(s) = s_i$, it holds that $s\sigma \in L$. □

Proof: Because of Lemma A.7, $\forall j$ s.t. $\sigma \in \Sigma_j$ it is true that $p_j(s)\sigma \in L_j$. For all j with $\sigma \notin \Sigma_j$, $p_j(s)\sigma \in L_j||(\Sigma - \Sigma_j)^*$. Thus $s \in L = ||_{k=1}^{n} L_k = \big(L_1||(\Sigma - \Sigma_1)^*\big) \cap \cdots \cap \big(L_n||(\Sigma - \Sigma_n)^*\big)$. □

We prove Lemma 4.2, using the properties stated in Lemma A.7 and Lemma A.8.

Proof: It has to be shown that the language $p_i^{\mathrm{hi}}(L_1^{\mathrm{hi,c}})$ is controllable w.r.t. $L_{i,1}^{\mathrm{hi,f}}$.

Assuming the contrary, it has to be the case that $\exists \sigma \in \Sigma_{i,\mathrm{uc}}^{\mathrm{hi}}$ and $s_i^{\mathrm{hi}} \in L_{i,1}^{\mathrm{hi,f}} \cap p_i^{\mathrm{hi}}(L_1^{\mathrm{hi,c}})$, s.t. $s_i^{\mathrm{hi}}\sigma \in L_{i,1}^{\mathrm{hi,f}}$ and $s_i^{\mathrm{hi}}\sigma \notin p_i^{\mathrm{hi}}(L_1^{\mathrm{hi,c}})$. Because of Lemma A.8, $\forall s^{\mathrm{hi}} \in L_1^{\mathrm{hi}}$ with $p_i^{\mathrm{hi}}(s^{\mathrm{hi}}) = s_i^{\mathrm{hi}}$, it holds that $s^{\mathrm{hi}}\sigma \in L_1^{\mathrm{hi}}$ and $\forall s^{\mathrm{hi}} \in L_1^{\mathrm{hi,c}}$ s.t. $p_i^{\mathrm{hi}}(s^{\mathrm{hi}}) = s_i^{\mathrm{hi}}$, it holds that $s^{\mathrm{hi}}\sigma \notin L_1^{\mathrm{hi,c}}$. But then $L_1^{\mathrm{hi,c}}\sigma \cap L_1^{\mathrm{hi}} \not\subseteq L_1^{\mathrm{hi,c}}$, which contradicts the assumption that $L_1^{\mathrm{hi,c}}$ is controllable w.r.t. L_1^{hi}. Thus $p_i^{\mathrm{hi}}(L_1^{\mathrm{hi,c}})$ is controllable w.r.t. $L_{i,1}^{\mathrm{hi,f}}$. □

Appendix B

Table of Events

The following table shows the list of events which are needed for modeling the manufacturing system in Chapter 5.

`sfmv`	belt of stack feeder moves	`sfstp`	belt of stack feeder stops
`sfwpar`	workpiece arrives at stack feeder	`sfwplv`	workpiece leaves the stack feeder
`sfr`	stack feeder at rest position	`sfnr`	stack feeder not at rest position
`sf-cb1`	workpiece from stack feeder to cb1	`t`	elapse of time
`cb1-x`	cb1 moves in -x-direction	`pu2wpar`	workpiece arrives at pu2
`pu2wplv`	workpiece leaves pu2	`sf-2`	transport workpiece to pu2
`pu2ar-y`	pu2 arrives at -y-direction	`pu2lv-y`	pu2 leaves -y-direction
`pu2ar+y`	pu2 arrives at +y-direction	`pu2lv+y`	pu2 leaves +y-direction
`pu2mv+y`	pu2 moves in +y-direction	`pu2mv-y`	pu2 moves in -y-direction
`pu2stp`	pu2 stops	`cb1awpar`	workpiece on cb1 arrives at pu2
`cb1awplv`	workpiece on cb1 leaves pu2	`sf-3`	transport workpiece to pu1
`sf-dep`	workpiece from cb1 to depot	`cb1stp`	cb1 stops
`pu2rdy`	operation of pu2 is ready	`cb1bwpar`	workpiece on cb1 arrives at pu1
`cb1-3`	workpiece from cb1 to pu1	`pu1rdy`	operation of pu1 is ready
`cb2-y`	cb2 moves in -y-direction	`cb2stp`	cb2 stops
`cb2wpar`	workpiece arrives at cb2	`cb2wplv`	workpiece leaves cb2
`cb2-13`	workpiece from cb2 to cb13	`wp2-13`	workpiece arrives at cb13 from cb2
`cb3-12`	workpiece from cb3 to cb12	`wp3-12`	workpiece arrives at cb12 from cb3
`cb12-4`	workpiece from cb12 to cb4	`wp12-4`	workpiece arrives at cb4 from cb12
`cb11-4`	workpiece from cb11 to cb4	`wp11-4`	workpiece arrives at cb4 from cb11
`cb4-12`	workpiece from cb4 to cb12	`wp4-12`	workpiece arrives at cb12 from cb4
`cb4-11`	workpiece from cb4 to cb11	`wp4-11`	workpiece arrives at cb11 from cb4
`mh1start`	machine mh1 starts operation	`mh1end`	machine mh1 terminates operation

`rt1xy`	rt1 from x- to y-position	`rt1y`	rt1 at y-position
`rt1yx`	rt1 from y- to x-position	`rt1x`	rt1 at x-position
`cb7-11`	workpiece from cb7 to cb11	`wp7-11`	workpiece arrives at cb11 from cb7
`cb11-7`	workpiece from cb11 to cb7	`wp11-7`	workpiece arrives at cb7 from cb11
`cb7-15`	workpiece from cb7 to cb15	`wp7-15`	workpiece arrives at cb15 from cb7
`cb15-rc1`	workpiece from cb15 to rc1	`wp15-rc1`	workpiece arrives at rc1 from cb15
`cb15-9`	workpiece from cb15 to cb9	`wp15-9`	workpiece arrives at cb9 from cb15
`rc1wplv`	workpiece leaves rc1	`rts1_7-8`	workpiece from cb7 to cb9 via rts1
`rts1_7-rc1`	workpiece from cb7 to rc1 via rts1	`cb10-16`	workpiece from cb10 to cb16
`cb13-6`	workpiece from cb13 to cb6	`wp13-6`	workpiece arrives at cb6 from cb13
`cb6-13`	workpiece from cb6 to cb13	`wp6-13`	workpiece arrives at cb13 from cb6
`cb6-14`	workpiece from cb6 to cb14	`wp6-14`	workpiece arrives at cb14 from cb6
`cb14-10`	workpiece from cb14 to cb10	`wp14-10`	workpiece arrives at cb10 from cb14
`rt2xy`	rt2 from x- to y-position	`wp10-16`	workpiece arrives at cb16 from cb10
`cb13-5`	workpiece from cb13 to cb5	`wp13-5`	workpiece arrives at cb5 from cb13
`cb5-12`	workpiece from cb5 to cb12	`wp5-12`	workpiece arrives at cb12 from cb5
`cb9-13`	workpiece from cb9 to cb13	`wp9-13`	workpiece arrives at cb13 from cb9
`rt3xy`	rt3 from x- to y-position		

References

[Bar99] G. Barrett. Modeling, Analysis and Control of Centralized and Decentralized Logical Discrete Event Systems. *PhD thesis, The University of Michigan*, 1999.

[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*, 15:111–117, 1990.

[BH93] Y. Brave and M. Heymann. Control of Discrete Event Systems Modeled as Hierarchical State Machines. *IEEE Transactions on Automatic Control*, 38(12):1803–1819, 1993.

[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.

[CL99] C.G Cassandras and S. Lafortune. Introduction to Discrete Event Systems. *Kluwer*, 1999.

[CTdC01a] J.E.R. Cury, C.R.C. Torrico, and A.E.C. da Cunha. A New Approach for Supervisory Control of Discrete Event Systems. *European Control Conference*, 2001.

[CTdC01b] J.E.R. Cury, C.R.C. Torrico, and A.E.C. da Cunha. Supervisory Control of Discrete Event Systems with Flexible Marking. *European Control Conference*, 2001.

[dCC02] A.E.C. da Cunha and J.E.R. Cury. Hierarchically Consistent Controlled Discrete Event Systems. *IFAC World Congress*, 2002.

[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. *Workshop on Discrete Event Systems*, 2002.

[dQ00] M.H. de Querioz. Controle Supervisório Modular de Sistemas de Grande Porte. *Master thesis, Universidade Federal de Santa Catarina*, 2000.

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

[Ers02] G. ERSOY. Anwendung und Erweiterung dezentraler Konzepte in der Supervisory Control Theory für ereignisdiskrete Systeme. *Diplomarbeit, Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg*, 2002.

[Fig05] S. FIGGEN. Design, Implementation and Validation of Supervisory Control for an Automated Manufacturing System. *Diplomarbeit, Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg*, 2005.

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

[GM05a] B. GAUDIN AND H. MARCHAND. Efficient Computation of Supervisors for loosely synchronous Discrete Event Systems: A State-Based Approach. *IFAC World Congress*, 2005.

[GM05b] B. GAUDIN AND H. MARCHAND. Safety Control of Hierarchical Synchronous Discrete Event Systems: A State-Based Approach. *Meditteranean Conference on Control and Automation*, 2005.

[Goh98] P. GOHARI. A Linguistic Framework for Controlled Hierarchical DES. *Master Thesis, Department of Electrical and Computer Engineering, University of Toronto*, 1998.

[Goh03] P. GOHARI. Fair Supervisory Control of Discrete Event Systems. *PhD thesis, Department of Electrical and Computer Engineering, University of Toronto*, 2003.

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

[Hop71] J. HOPCROFT. An $n \log n$-Algorithm for Minimizing the States in a Finite Automaton. *In Z. Kohavi, editor, The theory of machines and computations, Academic Press*, pages 189–196, 1971.

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

[JCK01] S. JIANG, V. CHANDRA, AND R. KUMAR. Decentralized Control of Discrete Event Systems with Multiple Local Specializations. *American Control Conference*, 2001.

[JK02] S. JIANG AND R. KUMAR. Decentralized Control of Discrete-Event Systems with Specializations to Local Control and Concurrent Systems. *IEEE Transactions on Systems, Man and Cybernetics*, 30(5):653–660, 2002.

[KS97] R. Kumar and M.A. Shayman. Centralized and Decentralized Supervisory Control of Nondeterministic Systems under Partial Observation. *SIAM Journal on Control and Optimization*, 35(2):363–383, 1997.

[KvS03] J. Komenda and J. H. van Schuppen. Decentralized Control with Coalgebra. *European Control Conference*, 2003.

[KvS04] J. Komenda and J. H. van Schuppen. Supremal Normal Sublanguages of Large Distributed Discrete-Event Systems. *Workshop on Discrete Event Systems*, 2004.

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

[LLW01] R.J. Leduc, M. Lawford, and W.M. Wonham. Hierarchical Interface-based Supervisory Control: AIP Example. *Allerton Conference on Communication, Control and Computation*, pages 396–305, 2001.

[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.

[LW97] S-H. Lee and K.C. Wong. Decentralised Control of Concurrent Discrete-Event Systems with Non-prefix Closed Local Specifications. *IEEE Conference on Decision and Control*, pages 2958–2963, December 1997.

[LW02] S-H. Lee and K.C. Wong. Structural Decentralised Control of Concurrent DES. *European Journal of Control*, 35:1125–1134, October 2002.

[LWL01] R.J. Leduc, W.M. Wonham, and M. Lawford. Hierarchical Interface-based Supervisory Control: Parallel Case. *Allerton Conference on Communication, Control and Computation*, pages 386–395, 2001.

[Ma99] C. Ma. A Computational Approach to Top-down Hierarchical Supervisory Control of Discrete Event Systems. *Master thesis, Department of Electrical and Computer Engineering, University of Toronto*, 1999.

[Ma04] C. Ma. Nonblocking Supervisory Control of State Tree Structures. *Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Toronto*, 2004.

[MG02] H. Marchand and B. Gaudin. Supervisory Control Problems of Hierarchical Finite State Machines. *IEEE Conference on Decision and Control*, pages 1199–1204, 2002.

[MRD03] T. MOOR, J. RAISCH, AND J.M. DAVOREN. Admissibility Criteria for a Hierarchical Design of Hybrid Control Systems. In *Conference on Analysis and Design of Hybrid Systems*, pages 389–394, 2003.

[MS05] T. MOOR AND K. SCHMIDT. Hierarchical Control from a Behavioral Perspective. In *International Conference on Methods and Models in Automation and Robotics*, 2005.

[MW03] C. MA AND W.M. WONHAM. Control of State Tree Structures. *Mediterranean Conference on Control and Automation*, 2003.

[Ner58] A. NERODE. Linear Automaton Transformations. *Proceedings AMS*, 9:541–544, 1958.

[Per04] S. PERK. Hierarchical Design of Discrete Event Controllers: An Automated Manufacturing System Case Study. *Diplomarbeit, Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg*, 2004.

[Pu00] K.Q. PU. Modeling and Control of Discrete Event Systems with Hierarchical Abstraction. *Master Thesis, Department of Electrical and Computer Engineering, University of Toronto*, 2000.

[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.

[RK91] S.I. MARCUS R. KUMAR, V. GARG. On Controllability and Normality of Discrete Event Dynamical Systems. *System and Control Letters*, 17:157–168, 1991.

[RL02] K. ROHLOFF AND S. LAFORTUNE. On the Computational Complexity of the Verification of Modular Discrete-Event Systems. *IEEE Conference on Decision and Control*, 2002.

[Rut99] J.J.M.M RUTTEN. Coalgebra, Concurrency, and Control. *Technical Report SEN-R9921, Centrum voor Wiskunde en Informatica*, 1999.

[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 Processes. *SIAM Journal of Control and Optimization*, 25:206–230, 1987.

[RW89] P.J. RAMADGE AND W.M. WONHAM. The Control of Discrete Event Systems. *Proceedings IEEE, Special Issue Discrete Event Dynamic Systems*, 77:81–98, 1989.

[RW95] K. Rudie and J.C. Willems. The Computational Complexity of Decentralized Discrete-Event Control Problems. *IEEE Transactions on Automatic Control*, 40(7):1313–1318, 1995.

[SMP05] K. Schmidt, T. Moor, and S. Perk. A Hierarchical Architecture for Nonblocking Control of Discrete Event Systems. *Mediterranean Conference on Control and Automation*, 2005.

[SPM05] K. Schmidt, S. Perk, and T. Moor. Nonblocking Hierarchical Control of Decentralized Systems. *IFAC World Congress*, 2005.

[SRM04] K. Schmidt, J. Reger, and T. Moor. Hierarchical Control of Structural Decentralized DES. *Workshop on Discrete Event Systems*, 2004.

[TC02] C.C. Torrico and J.E.R. Cury. Hierarchical Supervisory Control of Discrete Event Systems Based on State Aggregation. *IFAC World Congress*, 2002.

[Wan95] B. Wang. Top-down Design for RW Supervisory Control Theory. *Master thesis, Department of Electrical and Computer Engineering, University of Toronto*, 1995.

[WH91] Y. Willner and M. Heymann. Supervisory Control of Concurrent Discrete-Event Systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[Won97] K. Wong. On the Complexity of Projections of Discrete-Event Systems. *Technical Report 9705, Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto*, 1997.

[Won04] W.M Wonham. Notes on Control of Discrete Event Systems. *Department of Electrical Engineering, University of Toronto*, 2004.

[WR87] W.M. Wonham and P.J. Ramadge. On the Supremal Controllable Sublanguage of a Given Language. *SIAM Journal of Control and Optimization*, 25:637–659, 1987.

[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*, 1996.

[YL00] T. Yoo and S. Lafortune. A Generalized Framework for Decentralized Supervisory Control of Discrete Event Systems. *Workshop on Discrete Event Systems*, 2000.

[YL02] T. Yoo and S. Lafortune. A Generalized Architecture for Decentralized Supervisory Control of Discrete Event Systems. *Discrete Event Dynamic Systems: Theory and Applications*, 2002.

[Yoo02] TAE-SIC YOO. Monitoring and Control of Centralized and Decentralized Partially-Observed Discrete-Event Systems. *PhD thesis, The University of Michigan,* 2002.

[ZC94] R.M. ZILLER AND E.R. CURY. On the Supremal L_m-closed and the Supremal L_m-closed and $L-$controllable Sublanguages of a Given Language. *11th International Conference on Analysis and Optimization of Systems - Discrete Event Systems*, 1994.

[Zho92] H. ZHONG. Hierarchical Control of Discrete Event Systems. *PhD Thesis, Department of Electrical and Computer Engineering, University of Toronto*, 1992.

[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.

[ZW01] Z.H. ZHANG AND W.M. WONHAM. STCT: An Efficient Algorithm for Supervisory Control Design. *Symposium on Supervisory Control of Discrete Event Systems (SCODES2001), Paris*, 2001.

Lebenslauf

Zur Person:

Klaus Schmidt
geboren am 13. 08. 1976 in Fürth
verheiratet

Schulbildung:

1983–1987	Grundschule in Feilitzsch
1987–1996	Schillergymnasium in Hof
Juni 1996	Abschluss mit Abitur

Wehrdienst:

1996–1997	Grundwehrdienst beim Nachschubbataillon 4 in Weiden

Studium:

1997–2002	Studium der Elektrotechnik an der Friedrich-Alexander-Universität Erlangen-Nürnberg
1999	Tutor am Lehrstuhl für allgemeine und theoretische Elektrotechnik
Juli 2000	Aufnahme in die Studienstiftung des Deutschen Volkes
Nov. 2000	Hilfswissenschaftler am Lehrstuhl für Werkstoffe der Elektrotechnik
Aug. 2001	Praktikum bei Infineon Technologies in München
März 2002	Studienabschluss Dipl.-Ing.
Juli 2002	Preis des VDE für die beste Diplomarbeit in der Elektrotechnik

Hochschultätigkeit:

seit 2002	Wissenschaftlicher Assistent am Lehrstuhl für Regelungstechnik der Universität Erlangen-Nürnberg
2002-2003	Mitglied in der Studienkommission Elektrotechnik
2003-2004	einjähriger Gastaufenthalt an der Carnegie Mellon University in Pittsburgh