Modelling, control and fault detection of discretely-observed systems

Modelling, Control and Fault Detection of Discretely-Observed Systems

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Philips, Patrick P.H.H.

Modelling, Control and Fault Detection of Discretely-Observed Systems / by Patrick P.H.H. Philips.
Eindhoven : Technische Universiteit Eindhoven, 2001.
Proefschrift. - ISBN 90-386-1729-1
NUGI 831
Trefw.: regelsystemen; speciale methoden / hybride systemen / wiskundige modellen / discrete systemen / foutendetectie
Subject headings: nonlinear control systems / hybrid systems / modelling / discrete event systems / fault diagnosis

Druk: Universiteitsdrukkerij TU Eindhoven, The Netherlands

Modelling, Control and Fault Detection of Discretely-Observed Systems

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus,
prof.dr. M. Rem, voor een commissie
aangewezen door het College voor Promoties
in het openbaar te verdedigen op
maandag 23 april 2001 om 16.00 uur

door

Patrick Peter Hubert Helena Philips

geboren te Roermond

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.dipl-ing. H.A. Preisig
en
prof.dr.ir P.P.J. van den Bosch

Copromotor:
dr.ir. W.P.M.H. Heemels

Contents

Summary

In this thesis, controlled systems are studied for which the plant is described by continuous dynamics and the controller is given by a discrete-event model. Examples are plants that are observed by discrete sensors and manipulated by discrete inputs, and continuous systems for which the interaction with other processes is supervised by a computer program. For such systems models, control strategies and a fault detection scheme are developed.

The approach followed in the thesis to study this kind of hybrid systems is to translate the continuous dynamics of the plant (described by continuous differential equations) into a discrete-event system. In this way the interaction of two discrete-event systems is studied, which facilitates the analysis of the original hybrid system.

To 'discretize' the continuous dynamics, the state space of the system is partitioned into hypercubes associated to discrete states of an automaton. An discrete-event model can be described by specifying the possible transitions between discrete states. A transition from one discrete state to another discrete state is possible whenever there is a trajectory governed by the continuous dynamics from the corresponding hypercube to the other. To verify if there exists such a trajectory, an automated algorithm is developed that checks the boundary surface between two hypercubes for derivatives of trajectories (given by the continuous differential equation) that are directed towards the adjacent hypercube and therefore allowing a transition.

For systems with discrete measurements, *i.e.* measurements for which only a signal is emitted when a certain value is reached instead of when a certain time period has elapsed, it is shown how to reconstruct the continuous trajectory for both linear and nonlinear systems. When the input is known, the differential equations exactly describe the continuous plant, and measurement noise and disturbances are absent, a multipoint boundary value problem has to be solved to reconstruct the continuous state. Using the reconstructed information, a conventional continuous controller can be applied, or the information can be used for additional control actions. If still a discrete-event system is to be used as a controller, then for systems with outputs it is shown how to reconstruct the discrete state from a sequence of discrete measurements.

The discrete-event models that result from the modelling algorithm can be used for controller synthesis. Three controller design strategies are proposed. For these strategies, besides the obtained discrete-event model, also the knowledge that the underlying system is continuous is used.

The first controller design method is based on the construction of transitions that can be guaranteed to occur by choosing suitable inputs. Since it is difficult to determine whether this is possible, a sufficient condition is derived. The idea is to prevent (or correct) undesired transitions with inputs that are moving towards a desired transition. From all transitions that satisfy this sufficient condition, a graph is constructed. Suitable paths from a given initial state to a desired target state can be found from this graph by a constructive algorithm.

The principle of the second and third controller synthesis methods is based on the construction of controlled invariant sets for which it is certain that always inputs can be chosen that prevent the continuous trajectory from leaving this set. The idea is now to compute a sequence of nested invariant sets such that the first set contains the desired discrete states and the last one contains the initial state. This sequence of nested sets is constructed with the property that always inputs can be chosen that will move the continuous trajectory in the direction of a predecessor in the sequence, until eventually the target set is reached. The difference between the two synthesis methods is the construction of the controlled invariant sets. One method deletes discrete states that obstruct the invariance, whereas the other method includes discrete states to which the transition cannot be prevented.

The discrete-event models of the continuous systems can also be used for fault detection and isolation purposes. For this, for each possible fault that can occur a different discrete-event model of the plant is constructed. Next, by observing transitions of the real plant it is deduced by comparing the discrete-event models, if a measured transition can be caused by the nominal plant (*i.e.* without faults) or only can result from a fault that has occurred. Besides detecting a fault, it is also important to isolate the fault, which means that it has to be determined which fault actually happened. By merging two or more discrete states into one new state, the total number of discrete states can be reduced resulting in faster computations for on-line implementation of the fault detection scheme. However, it is only reasonable to merge discrete states if no information is lost. Explicit expressions are given that indicate that (*e.g.* transition or fault detection) information is preserved after merging states.

All the controller design and fault detection schemes are explicitly stated in terms of Boolean vectors and matrices and can, in principle, be implemented directly. Furthermore, all concepts are illustrated by means of examples, showing the capabilities of the methods.

Chapter 1

Introduction

This chapter serves as a first introduction to the subject of this thesis. Hybrid systems are introduced as being dynamical systems of a mixed continuous and discrete-event nature. The goals of this study, as well as the achievements that have been made, are presented.

1.1 Hybrid world

Many objects surrounding us in daily life have a hybrid nature in the sense that they possess continuous dynamics (described by *e.g.* differential equations) as well as discrete characteristics (*e.g.* logic switching). That is, their behavior is influenced by both continuous[1] and discrete[2] variables. For example, a car has a speed which can take any (continuous) value between zero and its top speed. Also the gas pedal takes any (continuous) position in between the admissable range. But there is only a finite (discrete) number of gears from which we can choose. For each gear the car has different dynamical properties. Switching between the gears is done externally by a driver or by an automatic gearbox. Other examples of hybrid systems are washing machines, VCR's, cd-players, microwaves, and the temperature control in a house via a thermostat, to mention just a few.

In industrial environments examples of hybrid systems are given by conveyer belts, robots, batch processes, and many (complex) machines. The introduction of computers in control systems has caused a considerable growth of hybrid systems. Digital computers can only deal with discrete variables by nature. They have to be supplied with discrete information and only discrete information will be returned. Consequently, for any continuous plant that is to be controlled by a computer, the closed-loop system will be a hybrid system.

[1] A continuous variable is a real-valued variable.

[2] A discrete variable is a variable that is an element of a countable set.

This may be disregarded for the case where the computer is only used for implementing conventional control laws and the influence of the 'discretization' of the plant's output and input can be neglected. However, in case where the computer serves, for example, as a supervisor that has to control the interaction of several smaller processes, it might be necessary to take the hybrid nature into account. Moreover, in process industry often such a supervisor not only has to (hierarchically) operate a process at a certain working point, but also it has to take care of the start-up or shut-down procedure of a plant. Also other tasks, such as safety handling and fault detection may need to be performed and incorporated in the control strategy. It is evident that in these situations the overall control strategy requires an (in some sense) intelligent character. That is, the controller has to recognize the 'discrete-state' the plant is in (*e.g.* the shut-down or an unsafe mode) and has to respond adequately. In most cases it is impossible to find a continuous controller that can deal with all these tasks.

Traditionally, system and control theory deals with systems for which the 'state' describing the system and the input acting on it have continuous values. Although early modern (*i.e.* based on the state space approach) books on control theory already treat discrete-event systems *(e.g.* (Kalman *et al.* 1969, Zadeh and Desoer 1963)), control theoretical results are mainly obtained for continuous systems. The first general results on controlling discrete-event systems only became available a few decades later (Ramadge and Wonham 1982, Ramadge and Wonham 1987*b*). It is rather difficult to establish general methods for constructing controllers for the large class of hybrid systems. Successful approaches to hybrid control system design are mainly established in the form of case studies like automated vehicles (Lygeros *et al.* 1998) or traffic management (Tomlin *et al.* 1998) for which specific strategies can be applied, or when a special structure of the underlying model class is imposed (*e.g.* (Tittus and Egardt 1998, Bett and Lemmon 1999, Bemporad and Morari 1999)). For an overview of many other interesting control strategies, see *e.g.* (Henzinger and Sastry 1998, Vaandrager and van Schuppen 1999, Lynch and Krogh 2000). However, it is clear that still a lot of research is needed to obtain generally applicable methods for designing practically useful controllers that can be implemented in an industrial environment.

1.2 Systems

To discuss the notion of a hybrid system in some detail, first it is explained what we mean by continuous systems and discrete-event systems. Let us recapitulate the definition of a system (Sontag 1990):

DEFINITION 1.2.1 *A system or machine* $\Sigma = (\mathcal{T}, \mathcal{X}, \mathcal{U}, \phi)$ *consists of:*

- *A time set* $\mathcal{T}$,
- *A nonempty set* $\mathcal{X}$ *called the state space of* Σ,
- *A nonempty set* $\mathcal{U}$ *control-value or input-value space of* Σ,
- *A map* $\phi : \mathcal{D}_\phi \to \mathcal{X}$ *called the transition map of* Σ, *which is defined on a subset* $\mathcal{D}_\phi$ *of*

  $$\{(\tau, \sigma, x, \omega) | \sigma, \tau \in \mathcal{T}, \sigma \leq \tau, x \in \mathcal{X}, \omega \in \mathcal{U}^{[\sigma,\tau)}\},$$

  such that the (1) nontriviality, (2) restriction, (3) semigroup, and (4) identity properties hold (Sontag 1990).

The transition map ϕ can be read as the state at time τ resulting from applying input ω, starting at time σ with state x. Often ϕ is written as $\phi(x, \omega)$ when τ and σ are clear from the context. To illustrate this definition of a system, a constant (time invariant), continuous time, linear system can be defined (see (Sontag 1990) or (Kalman *et al.* 1969)) by $\mathcal{T} = \mathbb{R}$, $\mathcal{X} = \mathbb{R}^n$, $\mathcal{U} = \mathbb{R}^m$, and

$$\phi(\tau, \sigma, x, \omega) = e^{A(\tau-\sigma)}x(\sigma) + \int_{\theta=\sigma}^{\tau} e^{A(\tau-\theta)}B\omega(\theta)d\theta,$$

where A, B are the system and input matrix, respectively.

1.2.1 Continuous systems

Using this definition, we can explain what is meant by a *continuous system* in this thesis. A continuous system is a system with the following properties:

1. $\mathcal{T} = \mathbb{R}$,
2. $\mathcal{X} \subseteq \mathbb{R}^n$ and is open and connected,
3. $\mathcal{U} \subseteq \mathbb{R}^m$,
4. The transition map ϕ is induced by a differential equation of the form

   $$\dot{x}(t) = f(x(t), u(t)), \qquad x(t_0) = x_0, \tag{1.1}$$

   where f is a continuous function in its arguments $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$.

So, the term 'continuous system' refers to the continuity of the state and input space of the system Σ and is not related to the time set $\mathcal{T}$ (and thus should not be confused with continuous-time as opposed to discrete-time systems[3]). The differential equations often follow from first principle modelling, resulting in a state space model.

1.2.2 Discrete-event systems

In general, *discrete-event systems* are systems for which both the dynamics is event-driven as opposed to time driven, and for which (at least some of) the variables required to describe the dynamics are discrete (Cassandras *et al.* 1995). So, the behavior of a discrete-event system is governed by the occurrence of events and not by the fact that time evolves.

Possible descriptions of discrete-event systems are (Cassandras *et al.* 1995):

- Automata (Hopcroft and Ullman 1979),
- Petri nets (Reisig 1985),
- Stochastic timed automata (Doberkat 1981),
- Linear systems in the $(\max, +)$-algebra (Olsder 1993),
- Markov chains (Freedman 1971).

Using Definition 1.2.1, the type of discrete-event systems we will use are given by systems with the following properties:

1. $\mathcal{X}$ is a finite set of discrete states,
2. $\mathcal{U}$ is a finite set of inputs, also called the input alphabet,
3. $\mathcal{T} = \mathbb{Z}$,
4. The transition map is defined as the *next-state* or *transition* map $\phi(t+1, t, x, u)$.

Note that for discrete-event systems there is no notion of time and the time set $\mathcal{T}$ is only used for the ordering of the events. In computer science, these systems are called automata. In the sequel, an automaton is described by $\Sigma = (\tilde{X}, \tilde{U}, \phi)$, with $\tilde{X}$ the set of discrete states, $\tilde{U}$ the set of discrete inputs, and ϕ the transition map (*cf.* Section 2.1 for a more detailed definition).

[3] In this thesis, we are only concerned with continuous-time systems. For discrete-time, the differential equation (1.1) will be replaced by a difference equation $x(t+1) = f(x(t), u(t))$ and $\mathcal{T} = \mathbb{Z}$.

1.2.3 Hybrid systems

There are various descriptions for modelling hybrid systems (see *e.g.* (Van der Schaft and Schumacher 2000)). Often, these modelling methods have emerged from a specific background. When a certain system is investigated, *e.g.* some mechanical system with hysteresis, then a (hybrid) model is needed that can describe the characteristics of hysteresis. If the interaction between a computer and a continuous plant is the subject of interest, then probably a completely different hybrid model is needed. In such cases, it is not necessary (or desirable) to use a more sophisticated hybrid modelling method that also can describe other hybrid phenomena, thereby rendering the analysis (in general) more difficult. Indeed, the more powerful the modelling method, the more difficult the analysis of the model class will be. So, a trade-off between modelling power (expressiveness) and decisive power has to be made.

Examples of methods for modelling hybrid phenomena are

- Hybrid Petri nets (David and Alla 1994),
- Mixed logical dynamical models (Bemporad and Morari 1999),
- Linear complementarity systems (Van der Schaft and Schumacher 1996, Heemels *et al.* 2000),
- Differential automata (Tavernini 1987),
- Hybrid automata (Branicky *et al.* 1998).

Historically, two different approaches for solving hybrid problems can be distinguished; the 'continuous' approach (mostly adopted by control engineers), which tries to capture the discrete variables into continuous models, and the 'discrete-event' approach (mostly developed by computer scientists), which attempts to include the continuous dynamics (approximately) into a discrete-event model. Nowadays, both views have developed in the same direction, resulting in the acceptance of a model first referred to as the *controlled general hybrid dynamical system* (Branicky *et al.* 1998), but most widely known as the hybrid automaton. The hybrid automaton model as presented below is a trimmed version of the one in (Branicky *et al.* 1998) as the discrete events are generated by the continuous dynamics only. Branicky's model is more general, because it incorporates for instance externally supplied discrete events. The model as described here suits our purpose and points out the characteristics of a hybrid system: continuous phases separated by events at which discrete actions such as re-initialization of the continuous state x and discrete state q take place. A hybrid automaton is defined as a quadruple

(using the notation as in (Branicky *et al.* 1998))

$$H = \{Q, \Sigma, A, G\},$$

where

- Q is the set of discrete states,
- $\Sigma = \{\Sigma_q\}_{q \in Q}$ is the collection of continuous systems $\Sigma_q = (\mathcal{X}_q, \mathcal{U}_q, \phi_q)$, see Section 1.2.1,
- $A = \{A_q\}_{q \in Q}$. $A_q \subseteq \mathcal{X}_q$ is the jump set for discrete state q,
- $G = \{G_q\}_{q \in Q}$. $G_q : A_q \to S$ is the jump transition map,

where, $S = \bigcup_{q \in Q} \mathcal{X}_q \times \{q\}$ is the hybrid state space of H. Loosely speaking, a hybrid trajectory evolves as follows. Starting in some initial state $s_0 = (x_0, q_0)$ for which $x_0 \in \mathcal{X}_{q_0} \setminus A_{q_0}$, the continuous part of the state evolves according to ϕ_{q_0} with input $u \in U_{q_0}$ until it reaches (if ever) A_{q_0} at time t_1 (the discrete part remains constant). In this case, the hybrid state will jump to $s_1 = (x_1, q_1) = G_{q_0}(x(t_1^-))$ with $x(t_1^-) := \lim_{\tau \uparrow t_1} x(\tau)$ from which the process continues.

This model captures a large number of hybrid phenomena such as switching and jumps of the state. However, due to its generality analysis is difficult and the obtained results are therefore rather limited. In (Branicky *et al.* 1998) an existence result for optimal controls is given which in general is hard to verify for concrete systems. In (Van Schuppen 1998) a sufficient condition for controllability of hybrid automata is presented. In (Lygeros *et al.* 1998, Tomlin *et al.* 1998) controllers are designed for specific case studies like automated vehicles or traffic management. So, although the modelling power of hybrid automata is large, for obtaining practically relevant results, additional restrictions have to be imposed to limit the class of systems under study on one hand, and enlarge the capabilities of deducing theoretical as well as practical results on the other. Fortunately, there are enough practically relevant problems that can be classified as being hybrid control problems, but for which it is not necessary that the overall hybrid control problem has to be solved, before obtaining interesting solutions.

1.3 Discrete-event models of continuous systems

In this thesis, we are concerned with continuous plants that are only observed by discrete sensors (*i.e.* only emitting a signal when a certain boundary in the state space is reached or crossed).

The continuous system is described, as before, by a set of differential equations:

$$\dot{x}(t) = f(x(t), u(t)), \qquad x(t_0) = x_0,$$

with $x(t) \in \mathbb{R}^n$. For the moment, we assume that $u(t)$ takes values from some discrete set, *i.e.* $u(t) \in \{\tilde{u}_1, ..., \tilde{u}_k\}$. For the case that $x(t) \in \mathbb{R}^2$ a trajectory evolves in a 2-dimensional state space, see Figure 1.1 (a).

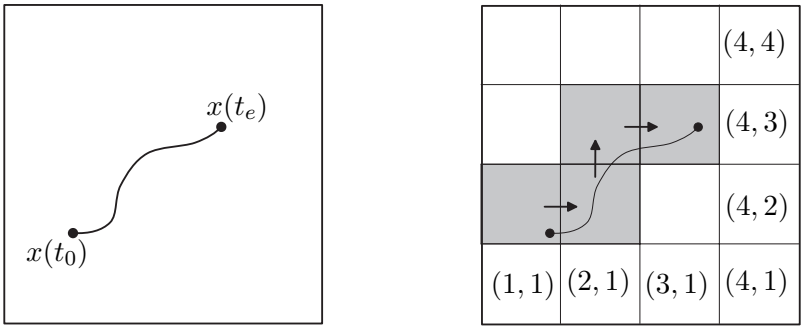

Figure 1.1: *(a) A trajctory in 2-dimensional state space; (b) discretization of the state space and the resulting discrete trajectory*

The discrete sensors are represented by a set of boundaries in the state space. To be specific, for each component x^i of the state vector x the following values give the positions of the 'sensors'.

$$\beta_0^i < \beta_1^i < ... < \beta_{n_i}^i \qquad (n_i \geq 1).$$

These discrete sensors induce a partitioning of the state space into hypercubes, being rectangles in Figure 1.1 (b). Since it is only measured when a boundary is reached, the only available information is the knowledge in which hypercube the system is currently in. This means that we only have coarse, discretized (quantized) information of the plant's state.

The motivation for studying systems observed by discrete sensors is twofold. The first reason is the frequent occurrence of these type of sensors in practical situations, such as level sensors and encoders. The second reason is that control on the basis of discretized information can be used for hierarchical control. The discretized information can serve as a basis for a coarse representation of the plant. This model then can be employed for high level hierarchical control or decision making that does not depend on the exact continuous state of the plant. Low level controllers can be used for fine-tuning. Note that it is

not strictly necessary that we have actually discrete sensors; we can also have continuous sensors in our plant, but we only act on the basis of the (artificial) discrete information.

Based on this discrete (or quantized) information, we want to influence the plant by control actions. One objective can be the steering of the process from one place in the state space to another (and possibly keep it there), the reachability or stabilization problem, respectively. The controller is modelled by a discrete-event system. Since a discrete-event controller cannot communicate with the system at a continuous level, an interface is required for the interconnection of the discrete-event controller and the continuous system, see Figure 1.2 (a).

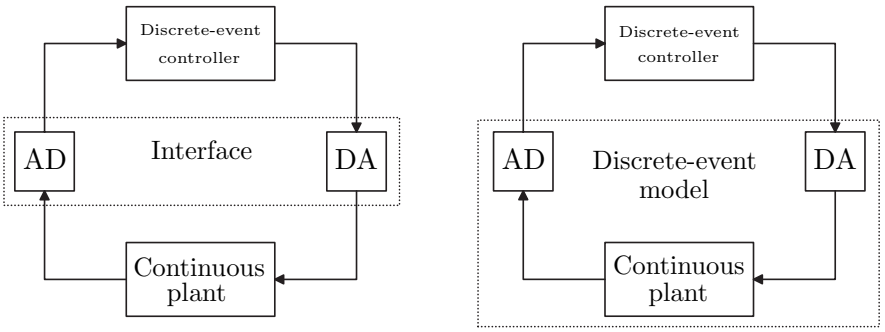

Figure 1.2: *(a) Hybrid system and (b) two discrete-event systems*

In this figure, the discrete to analog conversion is denoted by 'DA', which is sometimes also called the injector (Lunze *et al.* 1997). The block with 'AD' represents the analog to discrete conversion, also called quantizer (Lunze 1994). The interaction of a discrete-event controller with a continuous system (Figure 1.2 (a)) results in a hybrid system. The approach we will follow for dealing with this particular kind of hybrid systems is to model the continuous plant together with the interface as a discrete-event model, see Figure 1.2 (b). In this case, the interaction of two discrete-event models can be studied, which is less difficult than considering the overall hybrid system. In literature, these kind of systems are also referred to as 'quantized systems' (Lunze 2000). The resulting discrete-event models are known as 'discrete abstractions' (Lunze 1999), or 'qualitative models' (Lunze 1994).

Roughly, the construction of the discrete-event model of the continuous model is done as follows. The discrete sensors induce a partitioning of the state space into hypercubes, as is depicted in Figure 1.1 (b). Each hypercube can be identified with a tuple as shown in the figure. Such tuple can be interpreted as a discrete state. If there exists a trajectory from $x(t_0)$ to $x(t_e)$ in the

continuous state space (see Figure 1.1 (a)), then the discrete-event model that has to result must allow the corresponding path in terms of discrete states, as depicted in Figure 1.1 (b). Hence, transitions from $(1,2)$ to $(2,2)$, from $(2,2)$ to $(2,3)$ and from $(2,3)$ to $(3,3)$ must be possible with the automaton representing the continuous system. Since a continuous trajectory that evolves from one hypercube to another has to cross the boundary between the two hypercubes, it will be shown that it is sufficient to check the trajectories (in fact, only the derivatives of the trajectories) on the boundary between two hypercubes. The resulting discrete-event models then will be used for control and fault detection purposes.

1.4 Goals

The goals of this thesis are the following:

(i) Give a mathematically sound and practically relevant method for abstracting discrete-event systems from continuous systems described by differential equations.

(ii) Develop algorithms to design controllers on the basis of the obtained models.

(iii) Use these models for fault detection (*i.e.* determining if a fault has occurred) and fault isolation (*i.e.* deduce which fault has occurred) purposes.

(iv) Explore the (im)possibilities and (dis)advantages when using discrete-event models of continuous systems.

Concerning (i) it should be remarked that the idea of determining transitions between discrete states (induced by a partitioning of the continuous state space) by examining derivatives on the boundaries separating states has already been reported in (Lemmon and Antsaklis 1993, Preisig 1996*b*) and further developed in (Preisig *et al.* 1997) for linear systems (some preliminary work can be found in (Preisig 1992, Preisig 1993)). The method has been adapted for the nonlinear case in Chapter 3, see also (Bruinsma 1997, Philips *et al.* 1997).

The idea of abstracting discrete-event models from continuous systems is quite general. There are a number of authors that approximate continuous systems by discrete-event models as well. The differences between the various approaches are mainly caused by the description of the resulting discrete-event models (automata, timed automata, rectangular automata or stochastic automata), the model of the continuous plant (continuous time or discrete

time, linear or nonlinear) and the choice of the discrete states as related to subsets of the continuous state space (rectangles, half spaces, or more general shapes). For more details on other discrete-event modelling methods, we refer the reader to the end of Chapter 3.

Of course, it would not make sense to derive a model without being able to use it for analysis or controller synthesis purposes. With respect to (ii), the most straightforward approach for control once a discrete-event model is obtained, is to use the existing theory (*e.g.* (Ramadge and Wonham 1987*b*)) on controller design for discrete-event systems. However, much more effective synthesis techniques are obtained, if the continuous nature of the underlying system is exploited. Particular actions that are possible for this kind of discretely observed systems are not included in design methods for general discrete-event systems. In Chapter 5 we introduce three types of such actions and show their additional value for the control problems considered here.

Using the obtained discrete-event model for diagnosis (iii) seems a natural step since conventional methods have already been based on making logic decisions from observed plant behavior. The logic (rules) behind this are often deduced from operator's insight and experience or expert systems. When using discrete-event models abstracted from physical modelling, extracting the logic can be done systematically.

In the procedure of trying to achieve (i), (ii) and (iii) one is bound to reach the limitations of the discrete-event modelling approach that is used. These limitations are as important as the results that are established, since they indicate possible bottlenecks and ways to improve and enlarge the capabilities of the methods for future research in this direction.

1.5 Achievements

Modelling

The concept of a discrete-event model of a continuous system as used in this thesis, is given on the basis of an automaton, a continuous system, and mappings between the continuous state space and the discrete states. Much effort is put in exactly defining the mapping from the continuous state to the discrete state, which is defined for (pieces of) trajectories instead of points in the state space. In this way, neither the situation can occur where it cannot be decided to which discrete state a point in the state space should be assigned, nor will it lead to an incorrect discrete-event model. Moreover, the proposed mapping from the continuous to the discrete domain allows other (more general) kinds of discrete states, such as ordered combinations of the discrete states as we will use them. These kind of states can be exploited for the construction of more 'precise' discrete-event models, but most likely at the cost of elaborate

computations (such as integrations) and a large number of discrete states. In this thesis, we restrict ourselves to discrete states that allow a computationally less involved modelling method. To reduce the number of computations, the structural properties of the continuous system such as linearity or sparsity are exploited. Also, it is shown how to reduce the number of discrete states without losing information in some sense. This is done by merging two or more states into one new state, thus reducing the total number of states, and then checking if explicitly given conditions are satisfied, guaranteeing no loss of information.

A substantial piece of the modelling part is dedicated to continuous systems with outputs. A procedure is given that adapts the modelling method for these systems. It is shown that outputs (in general) complicate the modelling method. For outputs that are linear combinations of the state vector, alternative and less computationally demanding methods are presented.

State reconstruction

A method is proposed to reconstruct the continuous state for nonlinear systems from the information provided by discrete measurements of parts of the continuous state (not necessarily the same part for each measurement).

Control

For systems with discrete measurements (and possibly discrete inputs) three controller design methods are proposed. These control strategies use the discrete-event model of the continuous system as well as additional information provided by the continuous plant, such as continuity and information on derivatives that holds for parts of the state space. All controller design methods are illustrated by means of a two tank system.

Fault detection and isolation

It is shown how to use discrete-event models of continuous systems for fault detection and isolation purposes. Conditions are given for which a possible fault can be detected and isolated by the proposed detection scheme. Also, reduction of the number of discrete states is discussed together with a condition that has to be satisfied in order not to loose information concerning the detection and isolation of faults.

Boolean matrices

In this thesis, we make extensive use of Boolean matrices and vectors. Since a Boolean vector can be used for representing a set, it is possible to give explicit

expressions for particular sets in terms of operations on Boolean matrices. Therefore all controller design methods, the fault detection scheme, conditions and definitions are explicitly described by Boolean matrices and vectors.

1.6 Outline

After some preliminaries in Chapter 2, we will present the definition of a discrete-event model of a continuous system in Chapter 3. Next, the discrete-event modelling algorithm is presented that is used to extract discrete-event models from continuous systems. Important issues here are the mapping from the continuous state space to the discrete state space, and the determination of the transition function. For systems with outputs, a procedure is given for incorporating outputs in the modelling structure. If these outputs (in the continuous domain) are linear combinations of the continuous state, then alternative techniques can be given. Furthermore, it is shown how to use linearity and/or sparsity of the continuous system to reduce the computational effort necessary to obtain the discrete-event model. A three tank system serves as an example for illustrating the modelling method and the benefits that can be made by exploiting the structure of the continuous system.

In Chapter 4 it is first shown how to reconstruct the continuous state from partial discrete measurements. This is done for linear as well as for nonlinear systems. Next, a procedure is proposed for reconstructing the discrete state from partial discrete observations.

In Chapter 5 three controller design strategies are proposed. First, the control objectives and concepts that are used are explained. Then, the notion of 'preventing', 'correcting', and 'moving' inputs is formalized, followed by the definition of controlled invariant sets in the context as they are needed here. Next, the three synthesis methods are explained. The first method is based on preventing transitions to undesired states with inputs that ensure the movement towards a desired state. The second controller strategy constructs a nested set of controlled invariant sets that will be used for letting the state evolve to the desired objective. The third design method has a similar strategy, but differs in the construction of the controlled invariant sets. All control strategies are illustrated by a two tank example. Each controller design method is explained for the case that it is measured when the continuous trajectory reaches a boundary (defining our discrete states). Next, it is discussed how to adapt the strategies for the case that it is only measured that such boundary is crossed (instead of reached). The chapter is ended with the discussion of some additional computations that are required when an assumption that is made cannot be satisfied.

Chapter 6 deals with the detection and isolation of faults. After explaining

how the discrete-event models can be used for these purposes, conditions are given that have to be satisfied for a fault to be detectable or uniquely identifiable. Next, the reduction of the number of discrete states by merging two or more discrete states into one new discrete state is examined. Specifically, a condition is given that has to be satisfied in order not to loose information, for instance, causing a fault no longer to be detectable. A heat exchanger plant is used for showing the possibilities of the discussed material.

Finally, conclusions and recommendations will complete the thesis.

Chapter 2

Preliminaries

The following concepts are frequently used in this thesis.

2.1 Automata

DEFINITION 2.1.1 *A* nondeterministic *automaton is defined as the triple* $\Sigma = (\tilde{X}, \tilde{U}, \phi)$ *with*

- $\tilde{X}$*, the set of discrete states,*
- $\tilde{U}$*, the set of discrete inputs,*
- $\phi : \tilde{X} \times \tilde{U} \to 2^{\tilde{X}}$*, the partial transition function.*

Given a discrete state $\tilde{x} \in \tilde{X}$ and a discrete input $\tilde{u} \in \tilde{U}$, the transition function ϕ determines the next possible state: $\tilde{x}_{new} \in \phi(\tilde{x}, \tilde{u})$. Since ϕ needs not to be defined for all discrete states, it is a partial function. In case of finite $\tilde{X}$ the automaton is called a finite automaton. For a nondeterministic automaton it is not certain what the result of applying an input $\tilde{u}$ will be for a given discrete state $\tilde{x}$, since the next state is only known to be an element of some (known) set. Clearly, this makes the control of such systems difficult. An automaton is said to be *deterministic* if only one new state is possible, *i.e.* $\phi : \tilde{X} \times \tilde{U} \to \tilde{X}$ and $x_{new} = \phi(\tilde{x}, \tilde{u})$.

An automaton with *outputs* is a 5-tuple $\Sigma = (\tilde{X}, \tilde{U}, \phi, \tilde{Y}, h)$ consisting of an automaton together with a set of discrete outputs $\tilde{Y}$ and an output map $h : \tilde{X} \times \tilde{U} \to \tilde{Y}$. In computer science, automata with outputs often are referred to as sequential machines (Booth 1967, Hopcroft and Ullman 1979). If the output is only a function of the state (*i.e.* $\tilde{y} = h(\tilde{x})$), then it is called a Moore machine. If it is a function of the state and the input (*i.e.* $\tilde{y} = h(\tilde{x}, \tilde{u})$) then the term Mealy machine is used.

2.2 Representation of discrete states

The elements of the discrete sets $\tilde{X}$ and $\tilde{U}$, the discrete states $\tilde{x}$ and inputs $\tilde{u}$ respectively, can be represented in various ways. Each representation has its own applications and advantages. In this thesis, three different kinds of representation are used for the discrete states; the tuple representation $\tilde{x}$, the integer representation $\bar{x}$, and the Boolean vector representation $\hat{x}$. Accordingly the corresponding discrete sets are denoted by $\tilde{X}$, $\bar{X}$, and $\hat{X}$, respectively. For the remainder of this thesis, when no particular representation is specified, a discrete state is denoted by $\tilde{x}$.

Tuple representation In this representation, a discrete state is given by an n-tuple: $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_n) \in \tilde{X} = \{1, ..., m_1\} \times \{1, ..., m_2\} \times ... \times \{1, ..., m_n\}$, with $\tilde{x}_i \in \{1, ..., m_i\}$. This is probably the most natural representation for our purpose.

To facilitate the translation into other forms in a general manner, the following ordering of elements in $\tilde{X}$ is assumed:

$$\tilde{X} = \{(1,1,...,1),(2,1,...,1),...,(m_1,1,...,1),\\ (1,2,1,...,1),...,(m_1-1,m_2,...,m_n),(m_1,m_2,...,m_n)\}.$$

Integer representation A discrete state is given by an integer $\bar{x} \in \bar{X} = \{1, ..., p\}$, where $p = \prod_{j=1}^{n} m_j$. This is the most commonly used representation of a discrete state in literature, but is only used in this thesis as an intermediate form between the tuple representation and the Boolean vector representation to be explained next. The transformation $T_{\text{ti}} : \tilde{X} \to \bar{X}$ is given by

$$T_{\text{ti}}(\tilde{x}) = (\sum_{i=1}^{n} a_i(\tilde{x}_i - 1)) + 1,$$

with

$$a_i = \begin{cases} 1 & i = 1 \\ \prod_{j=1}^{i-1} m_j & i = 2, ..., n. \end{cases} \tag{2.1}$$

The ordering of the elements in $\tilde{X}$ is not necessary, but as can be seen from Example 2.2.1, results in the fact that the i-th element in the ordered set $\tilde{X}$ actually corresponds to the integer i in the integer domain. The inverse transformation $T_{\text{it}} : \bar{X} \to \tilde{X}$ is given by the following recur-

sive scheme

$$\tilde{x}_n = \left\lceil \frac{\bar{x}}{a_n} \right\rceil,$$
$$\tilde{x}_{n-k} = \left\lceil \frac{\bar{x} - \Pi_{j=n-k+1}^{n}(\tilde{x}_j - 1)a_j}{a_{n-k}} \right\rceil, \; k = 1, ..., n-1,$$

from which we get the coordinates of $\bar{x}$. Here $\lceil a \rceil$ denotes the smallest integer greater than or equal to a.

Boolean vector representation A discrete state is given by means of a vector $\hat{x} \in \hat{X} \subset \{0,1\}^p$, where $\hat{X}$ are all vectors that have only one nonzero element. The transformation from the integer to the Boolean vector representation $T_{\text{ib}} : \bar{X} \to \hat{X}$ is defined as:

$$T_{\text{ib}}(\bar{x}) = [0, 0, ..., 0, 1, 0, ..., 0, 0]^T,$$

with the 1 at the $\bar{x}$-*th* position. In this way, $\hat{x} = T_{\text{ib}}(\bar{x})$ is equal to the unit-vector denoted by $e_{\bar{x}}$. Consequently, the transformation $T_{\text{bi}} : \hat{X} \to \bar{X}$ is defined by

$$T_{\text{bi}}(\hat{x}) = j \text{ if } \hat{x}^j = 1.$$

The transformations $T_{\text{tb}} : \tilde{X} \to \hat{X}$ and $T_{\text{bt}} : \hat{X} \to \bar{X}$ follow immediately from the composite mapping

$$T_{\text{tb}}(\tilde{x}) = T_{\text{ib}} \circ T_{\text{ti}}(\tilde{x}), \text{ and}$$
$$T_{\text{bt}}(\hat{x}) = T_{\text{it}} \circ T_{\text{bi}}(\hat{x}),$$

respectively.

The different forms of representation are explained for the discrete states but of course can also be used of the discrete inputs.

EXAMPLE 2.2.1 *As a comparison, the three representation forms are put together in one table for the case where* $m_1 = 3$, $m_2 = 2$, *and* $m_3 = 2$, *implying that* $p = 12$.

3-Tuple	Integer	Boolean vector
$(1,1,1)$	1	$[1,0,0,0,0,0,0,0,0,0,0,0]^T$
$(2,1,1)$	2	$[0,1,0,0,0,0,0,0,0,0,0,0]^T$
$(3,1,1)$	3	$[0,0,1,0,0,0,0,0,0,0,0,0]^T$
$(1,2,1)$	4	$[0,0,0,1,0,0,0,0,0,0,0,0]^T$
$(2,2,1)$	5	$[0,0,0,0,1,0,0,0,0,0,0,0]^T$
$(3,2,1)$	6	$[0,0,0,0,0,1,0,0,0,0,0,0]^T$
$(1,1,2)$	7	$[0,0,0,0,0,0,1,0,0,0,0,0]^T$
$(2,1,2)$	8	$[0,0,0,0,0,0,0,1,0,0,0,0]^T$
$(3,1,2)$	9	$[0,0,0,0,0,0,0,0,1,0,0,0]^T$
$(1,2,2)$	10	$[0,0,0,0,0,0,0,0,0,1,0,0]^T$
$(2,2,2)$	11	$[0,0,0,0,0,0,0,0,0,0,1,0]^T$
$(3,2,2)$	12	$[0,0,0,0,0,0,0,0,0,0,0,1]^T$

It is possible to have the transformations mentioned in the foregoing operate on sets. Suppose $\tilde{X}_1$ is a subset of $\tilde{X}$. Then the corresponding set in integer representation will be denoted as $\bar{X}_1$ and has the same number of elements as in the tuple representation. Moreover, we define $T_{\mathrm{TI}} : 2^{\tilde{X}} \to 2^{\bar{X}}$ by

$$\bar{X}_1 = T_{\mathrm{TI}}(\tilde{X}_1) = \{\bar{x} \mid \bar{x} = T_{\mathrm{ti}}(\tilde{x}) \text{ for some } \tilde{x} \in \tilde{X}_1\}.$$

In the Boolean vector domain it is possible to represent a set of discrete states $\tilde{X}_1 \subseteq \tilde{X}$ by the single vector $\hat{x}_1 \in 2^{\hat{X}} = \{0,1\}^p$ defined by $\hat{x}_1^i = 1$ if $i \in \bar{X}_1$, and $\hat{x}_1^i = 0$ if $i \notin \bar{X}_1$ for $i = 1, ..., p$. The advantage of representing sets in the Boolean vector domain is, that various operations on sets can be easily expressed. These computations are based on the binary operations $\oplus$ ('or'), $\otimes$ ('and'), and $\ominus$ on $\{0,1\}$ defined by $0 \oplus 0 = 0 \otimes 0 = 0 \otimes 1 = 1 \otimes 0 = 0 \ominus 0 = 0 \ominus 1 = 1 \ominus 1 = 0$, and $0 \oplus 1 = 1 \oplus 0 = 1 \oplus 1 = 1 \otimes 1 = 1 \ominus 0 = 1$.

Union The union of two sets $\tilde{Z} := \tilde{X}_1 \cup \tilde{X}_2$ is computed by $\hat{z} = \hat{x}_1 \oplus \hat{x}_2$ with $\hat{z}^i = \hat{x}_1^i \oplus \hat{x}_2^i$.

Intersection The intersection $\tilde{Z} := \tilde{X}_1 \cap \tilde{X}_2$ is computed by $\hat{z} = \hat{x}_1 \otimes \hat{x}_2$ with $\hat{z}^i = \hat{x}_1^i \otimes \hat{x}_2^i$. Furthermore, note that $\tilde{X}_1 \cap \tilde{X}_2 \neq \emptyset \iff \hat{x}_1^T \hat{x}_2 \neq 0$.

Difference The set difference $\tilde{Z} := \tilde{X}_1 \setminus \tilde{X}_2$ is computed by $\hat{z} = \hat{x}_1 \ominus \hat{x}_2$ with $\hat{z}^i = \hat{x}_1^i \ominus \hat{x}_2^i$.

Subsets One can check if $\hat{x}_1 \subseteq \hat{x}_2$ in the Boolean vector notation by using

$$\hat{x}_1 \subseteq \hat{x}_2 \iff \hat{x}_1 \oplus \hat{x}_2 = \hat{x}_2 \iff \hat{x}_1 \ominus \hat{x}_2 = 0.$$

2.3 Directed graphs

An automaton $\Sigma = (\tilde{X}, \tilde{U}, \phi)$ describes the relations between elements of the set $\tilde{X}$ (the discrete states) which depend on the elements of the set $\tilde{U}$ (the inputs) by means of the transition function ϕ. This information can be visualized by using labeled directed graphs. A directed graph (digraph) is a tuple $\Gamma = (V, E)$, where V is the set of vertices (or nodes), and E is the set of ordered pairs of vertices called edges (or arcs). As an example in Figure 2.1 the digraph is depicted for the set of vertices $V = \{1, 2, 3, 4\}$ and the set of edges $E = \{(1, 3), (3, 4), (4, 3), (2, 4), (4, 2), (2, 1)\}$. To each edge we can also

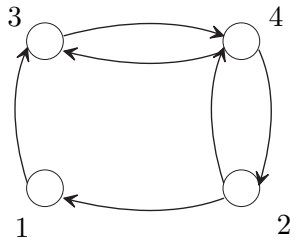

Figure 2.1: *A directed graph*

attach a label or a set of labels in which case we obtain a labeled digraph. Now, a labeled digraph Γ is associated with an automaton Σ by assigning to each discrete state $\tilde{x} \in \tilde{X}$ a vertex $v \in V$ and to each discrete input $\tilde{u} \in \tilde{U}$ a collection of edges $E_{\tilde{u}} \subseteq E$ with the label '$\tilde{u}$' in such a way that there is an edge from v_1 to v_2 with the label $\tilde{u}$ iff the automaton allows a transition from the corresponding discrete states $\tilde{x}_1$ to $\tilde{x}_2$ with the input $\tilde{u}$. So, if v_1 and v_2 are associated with $\tilde{x}_1$ and $\tilde{x}_2$, respectively and (v_1, v_2) is labeled with $\tilde{u}$ then

$$(v_1, v_2) \in E_{\tilde{u}} \Longleftrightarrow \tilde{x}_2 \in \phi(\tilde{x}_1, \tilde{u}).$$

For conveniently representing an automaton by a digraph, we will use the same symbols for both the discrete states and the corresponding vertices. An example of an automaton represented by a directed graph is given in Figure 2.2.

EXAMPLE 2.3.1 *Given the nondeterministic automaton* $\Sigma = (\tilde{X}, \tilde{U}, \phi)$*, with* $\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4\}$*,* $\tilde{U} = \{\tilde{u}_1, \tilde{u}_2\}$ *and* ϕ *defined by*

$$\begin{aligned} \phi(\tilde{x}_1, \tilde{u}_1) &= \{\tilde{x}_3\}, \\ \phi(\tilde{x}_2, \tilde{u}_1) &= \{\tilde{x}_4\}, \\ \phi(\tilde{x}_3, \tilde{u}_1) &= \{\tilde{x}_4\}, \\ \phi(\tilde{x}_4, \tilde{u}_2) &= \{\tilde{x}_2, \tilde{x}_3\}, \end{aligned}$$

then the corresponding labelled digraph is depicted in Figure 2.2.

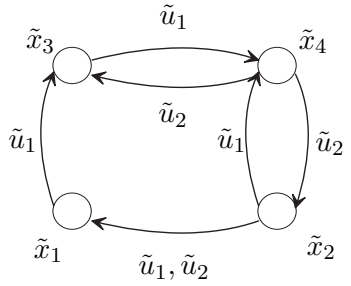

Figure 2.2: *Digraph of an automaton*

2.4 Adjacency matrices

Digraphs can be represented by so called adjacency matrices. For a digraph $\Gamma = (V, E)$ the adjacency matrix $A \in \{0,1\}^{p \times p}$ is a Boolean matrix defined by

$$a_{ij} = \begin{cases} 1 & \text{if } (v_j, v_i) \in E \\ 0 & \text{if } (v_j, v_i) \notin E. \end{cases}$$

As an example, the adjacency matrix of the digraph depicted in Figure 2.1 is given by

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

Since an adjacency matrix represents a digraph and a labelled digraph can represent an automaton, it can be seen that an automaton can be represented by a set of adjacency matrices $\{E_{\tilde{u}}\}_{\tilde{u} \in \tilde{U}}$; for each input $\tilde{u} \in \tilde{U}$ a different adjacency matrix $A_{\tilde{u}}$ is needed.

If the discrete states of the automaton are represented by Boolean vectors, then adjacency matrices facilitate many computational and analysis issues. In this respect, the following operations are useful.

Boolean matrix multiplication Given $A \in \{0,1\}^{n \times l}$, $B \in \{0,1\}^{l \times m}$, then $C = AB$ is defined as $c_{ij} = \bigoplus_{k=1}^{l} a_{ik} \otimes b_{kj} = (a_{i1} \otimes b_{1j}) \oplus (a_{i2} \otimes b_{2j}) \oplus \ldots \oplus (a_{il} \otimes b_{lj})$.

Boolean matrix "or" Given $A, B \in \{0,1\}^{n \times m}$, then $C = A \oplus B$ is defined as $c_{ij} = a_{ij} \oplus b_{ij}$.

Boolean matrix "and" Given $A, B \in \{0,1\}^{n \times m}$, then $C = A \otimes B$ is defined as $c_{ij} = a_{ij} \otimes b_{ij}$.

For instance, the possible next new states given a set of initial states $\tilde{X}_1$ and an input $\tilde{u}$ for automata, can easily be computed by representing $\tilde{X}_1$ by $\hat{x}_1$ in the Boolean vector domain (see page 18). The vector $\hat{x}_2 = A_{\tilde{u}}\hat{x}_1$ represents now the set of states that can be reached within one step from the set of states $\tilde{X}_1$ with input $\tilde{u}$. The set of discrete states that can be reached *after* k transitions from initial set $\tilde{X}_1$ with the input $\tilde{u}$ is equal to $\hat{x}_2 = A^k_{\tilde{u}}\hat{x}_1$. If we want to know which discrete states can be reached *within* k steps, then this is computed by $\hat{x}_2 = (A^k_{\tilde{u}} \oplus A^{k-1}_{\tilde{u}} \oplus ... \oplus A_{\tilde{u}} \oplus I)\hat{x}_1 = (A_{\tilde{u}} \oplus I)^k\hat{x}$, where I is the identity matrix of appropriate dimensions. To compute the set of states that can be reached after (within) k steps using arbitrary discrete input sequences (possibly a different one after each transition) it is sufficient to replace the adjacency matrix $A_{\tilde{u}}$ in the previous, by the 'overall' adjacency matrix defined by

$$A_{\tilde{U}} := A_{\tilde{u}_1} \oplus A_{\tilde{u}_2} \oplus ... \oplus A_{\tilde{u}_q},$$

where q is the number of discrete inputs, *i.e.* $q = \#(\tilde{U})$.

The set of states that can be reached from some initial state (or set of initial states) using any input-sequence is characterized by the so-called reachability matrix of a graph, which is defined by $(A_{\tilde{U}} \oplus I)^{p-1}$. Note, that it is sufficient to take the $(p-1)$-th power (Kim 1982).

Also the transposed adjacency matrix A^T is useful for computations. By the definition of an adjacency matrix it holds that $a_{ij} = 1$ iff there is an edge from vertex v_j to vertex v_i. If $B = A^T$, we obtain $b_{ij} = a_{ji}$. Hence, $b_{ij} = a_{ji} = 1$ if there is an edge from vertex v_i to vertex v_j. As a consequence, given an initial discrete state (or set of states) $\hat{x}_1$ then $\hat{x}_2 = A^T_{\tilde{u}}\hat{x}_1$ is the set of states from which $\hat{x}_1$ can be reached in one step with input $\tilde{u}$.

2.5 Labeled digraphs

To facilitate computations, it is sometimes convenient to represent the labeled digraphs by matrices. For enabling actual computations, each label is represented by a unique number such that from this number the original label can be restored and moreover, from the sum of such numbers also the corresponding set of labels can be restored. One possibility is via the function g defined for discrete inputs in the integer domain: $g : \bar{U} \to \mathbb{N}$

$$g(\bar{u}) = 2^{\bar{u}-1}.$$

Now, the labelled adjacency matrix $\mathbf{A}_{\tilde{U}}$ is constructed by

$$\begin{aligned}\mathbf{A}_{\tilde{U}} &= g(\bar{u}_1)A_{\tilde{u}_1} + g(\bar{u}_2)A_{\tilde{u}_2} + ... + g(\bar{u}_q)A_{\tilde{u}_q} \\ &= 2^0 A_{\tilde{u}_1} + 2^1 A_{\tilde{u}_2} + 2^2 A_{\tilde{u}_3} + ... + 2^{q-1} A_{\tilde{u}_q},\end{aligned}$$

where '+' is the regular (not the logical) addition. It is easily computed what the next possible discrete states are from a given initial discrete state $\hat{x}$ in the Boolean vector domain, and which input will lead to which state by computing $w = \mathbf{A}_{\tilde{U}}\hat{x}$. Although the initial discrete state is represented in the Boolean vector domain, the (matrix) multiplication is not the Boolean matrix multiplication but the ordinary one. The result w is a p-dimensional vector of integers. Suppose that $w^i \neq 0$. This implies that the discrete state $\bar{x}' = i$ (in the integer domain) can be reached from the initial state $\hat{x}$. Moreover, from the value of w^i it can be seen which discrete inputs actually are able to cause this transition. This can be deduced from the following recursive scheme:

$$v_q = \left\lfloor \frac{w^i}{2^{q-1}} \right\rfloor,$$
$$v_k = \left\lfloor \frac{w^i - \sum_{k+1}^{q} v_i 2^{i-1}}{2^{k-1}} \right\rfloor, \; k = q-1, ..., 1$$

where $\lfloor a \rfloor$ denotes the largest integer smaller than or equal to a. Hence, the input $\bar{u} = j$ (in integer representation) can cause the transition $\bar{x} \rightarrow \bar{x}' = i$ if and only if $w^i \neq 0$ and $v_j = 1$. The set of inputs that allow the transition from discrete state $\bar{x}$ to discrete state i is thus given by $\bar{U}_1 = \{l \in \bar{U} \mid v_l = 1\}$.

EXAMPLE 2.5.1 *Given the automaton Σ defined in Example 2.2.1, the labeled adjacency matrix $\mathbf{A}_{\tilde{U}}$ is given by*

$$\mathbf{A}_{\tilde{U}} = 2^0 \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} + 2^1 \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 2 & 2 & 0 \end{bmatrix}.$$

Given the initial state $\hat{x} = [0, 1, 0, 0]^T$ we obtain $w = \mathbf{A}_{\tilde{U}}\hat{x} = [3, 0, 0, 2]^T$ from which we can see that the states $\tilde{x}' = 1$, and $\tilde{x}'' = 4$ can be reached. The transition to $\tilde{x}' = 1$ can be caused by both $\bar{u}_1$ and $\bar{u}_2$ (since $3 = 1 \cdot 2^0 + 1 \cdot 2^1$ indicates that $v_2 = 1$ and $v_1 = 1$). The transition to $\tilde{x}'' = 4$ is caused by $\bar{u}_2$ only (as $2 = 0 \cdot 2^0 + 1 \cdot 2^1$ indicates that $v_1 = 0$ and $v_2 = 1$).

2.6 The Neighbor matrix

Finally, the so-called Neighbor matrix $N \in \{0, 1\}^{p \times p}$ is defined as:

$$n_{ij} = \begin{cases} 1 & \text{if } j \in \{i \pm a_k \mid k = 1, ..., n\} \\ 0 & \text{else.} \end{cases}$$

with a_k defined as in (2.1). The Neighbor matrix N can be interpreted as follows: for the discrete state $\hat{x}_1$ the set $\hat{x}_2 = N\hat{x}_1$ contains all discrete states for which the tuple representation differs only one unit in each coordinate from the tuple representation of $\hat{x}_1$ (these are the 'neighbors' of $\tilde{x}_1$). For example, in Figure 1.1 in Section 1.3 the neighbors of $(2,2)$ are given by $(1,2)$, $(2,1)$, $(2,3)$ and $(3,2)$.

2.7 The (relative) interior of a set

Given the set $D' \subseteq \mathbb{R}^n$. Define the set D as the smallest linear variety (*i.e.* affine subspace) of $\mathbb{R}^n$ containing D'. Throughout this thesis, when mentioning the interior of D', denoted by $int(D')$ the interior relative to D is meant. To be precise,

$$int(D') = \{x \in D' \mid \exists \delta > 0,\, B_D(x,\delta) \subseteq D'\},$$

with $B_D(x_0,\delta) := \{x \in D \mid \|x - x_0\| < \delta\}$.

In case $D = \mathbb{R}^n$ this definition is equivalent to the conventional interior of D'.

2.8 Notes and references

Automata

In this section we used the 'classical' definition of an automaton, because it corresponds to our definition of a system. For references, see *e.g.* (Hopcroft and Ullman 1979, Sontag 1990). For a nondeterministic automaton with outputs, a more general description is given by using the behavioral relation $L(\tilde{x}', \tilde{y}, \tilde{x}, \tilde{u})$ which includes all 4-tuples $(\tilde{x}', \tilde{y}, \tilde{x}, \tilde{u})$ for which it holds that the automaton switches from state $\tilde{x}$ to $\tilde{x}'$ for the input $\tilde{u}$ and simultaneously produces output $\tilde{y}$ (Lichtenberg and Lunze 1997).

Representation of discrete states

The material in this section is used for facilitating computation and notation in the Chapters 4, 5 and 6. The tuple representation of a discrete state is especially suited for our purposes, but is not used in general. The integer and Boolean vector notation are more commonly used each at their own field of application: in automata theory, see *e.g.* (Hopcroft and Ullman 1979, Booth 1967) and the theory of digraphs, see *e.g.* (Robinson and Foulds 1980) respectively. The theory of adjacency matrices is closely connected with Boolean matrix theory, see (Kim 1982).

Chapter 3

Discrete-event models of continuous systems

In this chapter a notion of a discrete-event model of a continuous system is formalized. We will discuss a method that creates a discrete-event model from a given set of differential equations and a partitioning of the continuous state space. Moreover, it is shown how to construct discrete-event models from systems for which the (measured) output is not equal to the state. Using a straightforward approach, the computational effort to obtain discrete-event models from continuous systems can become very large. Taking advantage of structural properties, the effort can be significantly reduced. One method proposed in this thesis exploits the linearity of the system, while the other approach is based on a hierarchical decomposition.

3.1 The models

Consider a continuous-time system described by the differential equations:

$$\dot{x}(t) = f(x(t), u(t)), \qquad x(t_0) = x_0, \tag{3.1}$$

involving the variables $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$, which represent the state and input, respectively. It is assumed that f is continuous and that the system (3.1) has a unique solution ξ for every initial value in the region of interest and for any input of interest (the exact input function space will be defined next). As indicated in Chapter 1, the first goal of the thesis is to provide a mathematically rigorous concept of discrete-event models of continuous systems, which is not present in the literature for continuous-time systems. We start by a formal and abstract notion, that allows general choices of discrete states as will be demonstrated in Example 3.1.4. Next the hypercube set-up as described in Section 1.3 will be formalized based on this general concept.

To introduce the concept of a discrete-event system (DES) of a continuous system, we define the set of continuous functions from closed intervals to $\mathbb{R}^n$ as $\mathcal{C}^n := \bigcup_{a \leq b} C^0([a,b], \mathbb{R}^n)$, where a may be equal to $-\infty$ but b may not be equal to ∞. Also, we define the set of piecewise continuous functions from closed intervals to $\mathbb{R}^m$, which are right continuous[1]: $\mathcal{PC}^m := \bigcup_{a \leq b} PC^0([a,b], \mathbb{R}^m)$. For $T \subseteq \mathbb{R}$ the space $PC^0(T, \mathbb{R}^m)$ denotes the collection of all piecewise continuous functions from T to $\mathbb{R}^m$ with a finite number of discontinuity points in a time interval of finite length. We will first formalize the concept of a discrete-event system of a continuous system, after which the ingredients of the definition are discussed in more detail.

Definition 3.1.1 (DES of a continuous system) *A discrete-event model of the system (3.1) is given by an automaton*

$$\Sigma = (\tilde{X}, \tilde{U}, \phi), \tag{3.2}$$

together with the mappings

$$Q : \mathcal{C}^n \to \tilde{X},$$
$$S : \mathcal{PC}^m \to \tilde{U},$$

such that the following holds:

Condition 3.1.2 (completeness) *Let $\upsilon \in PC^0([t_0, t_e], \mathbb{R}^m)$ be an input-signal applied to the system (3.1) from time t_0 to t_e. Let $\xi \in C^0([t_0, t_e], \mathbb{R}^n)$ denote a trajectory satisfying (3.1) for input υ. For all $t_i \in [t_0, t_e]$ satisfying $Q(\xi_{[t_0,t_i]}) \neq Q(\xi_{[t_0,t_i+\varepsilon]})$ for all sufficiently small $\varepsilon > 0$, it holds that $\tilde{x}_2 \in \phi(\tilde{x}_1, \tilde{u})$ where $\tilde{x}_1 := Q(\xi_{[t_0,t_i]})$, $\tilde{x}_2 := \lim_{t \downarrow t_i} Q(\xi_{[t_0,t]})$, and $\tilde{u} = S(\upsilon_{[t_0,t_i]})$.*

Condition 3.1.2 states that if for a given input the continuous trajectory results in a change of the discretized states, then also the automaton should allow the transition between the corresponding discrete states and with the corresponding discrete input. In other words, it is not possible that for the continuous system a transition occurs that is not modelled by the discrete-event model.

To each piece of trajectory (including the history) we associate a discrete state with the mapping Q. To illustrate this, in the following examples possible choices for Σ are presented.

Example 3.1.3 *Consider the linear autonomous differential equation*

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} x$$

[1] A function ξ is right continuous if for all τ it holds that $\lim_{t \downarrow \tau} \xi(t) = \xi(\tau)$.

resulting in the vector field as depicted in Figure 3.1 for the region of interest $\{x| -0.5 \leq x_1 \leq 1$ *and* $-1 \leq x_2 \leq 0.5\}$. *For this system with no inputs* $\tilde{U}$

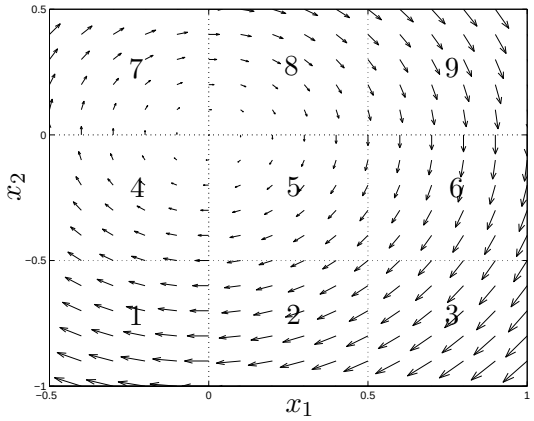

Figure 3.1: *Vector field*

and S *are irrelevant. The mapping* Q *is based on the regions numbered by* $1, ..., 9$ *in Figure 3.1. The interior of each cell belongs to the corresponding region. A region includes the parts of the boundary where the flow leaves the region, while the parts where the flow enters the region are excluded from the region (the point* $(0, 0)$ *is assigned to only one region, say* 7*). As an example, region* 5 *is defined by* $\{x \in \mathbb{R}^2 \mid 0 \leq x_1 < 0.5, -0.5 \leq x_2 < 0\}$. Q *is defined as follows: trajectories currently in region* 1 $(2, ..., 9)$ *are assigned the discrete state* 1 $(2, ..., 9)$. *From Figure 3.1 it can be seen that the transition function* ϕ *must allow the following transitions between discrete states to satisfy Condition 3.1.2:*

$\phi(1) = \{4\}$	$\phi(4) = \{7\}$	$\phi(7) = \{8\}$
$\phi(2) = \{1\}$	$\phi(5) = \{2, 4\}$	$\phi(8) = \{5, 9\}$
$\phi(3) = \{2\}$	$\phi(6) = \{3, 5\}$	$\phi(9) = \{6\}$

By concatenating discrete states resulting from sequential transitions a complete trajectory can be transformed into a sequence of discrete states. If Condition 3.1.2 is satisfied, then also the automaton will generate this state sequence. So, all discretized (quantized) trajectories of the continuous system can also be generated by the discrete-event model. The property that the set of discretized trajectories of the continuous system is a subset of the set of discrete trajectories of the discrete-event model (called *completeness* in literature) is crucial (Lunze 1994). Completeness allows us to transfer the results obtained for the discrete-event model to the underlying continuous system, *i.e.* if a result holds for the discrete-event model, then it also holds for

the continuous system. For results on fault diagnosis this is shown in (Förstner and Lunze n.d.). It is clear that a discrete-event representation of a continuous system also may generate discrete-state sequences that are not related to a trajectory of the system (3.1), that is, the automaton (3.2) generates spurious solutions. If we have two discrete-event models (Σ_1, Q_1, S_1) and (Σ_2, Q_2, S_2) of the continuous system (3.1), then we can say that (Σ_1, Q_1, S_1) is better than (Σ_2, Q_2, S_2) (or has a better quality) if it has less spurious solutions. The choice of the discrete states is very important for the quality of a model, as can be seen from the following Example.

EXAMPLE 3.1.4 *Again, consider the system described in Example 3.1.3. Note that with the set of discrete states defined as in Example 3.1.3, the automaton allows the sequence* $8, 5, 2$*. However from Figure 3.2, it can be seen that in the continuous domain there is no trajectory from region* 8 *to region* 2 *going through region* 5*, so this is a spurious solution. However, suppose we*

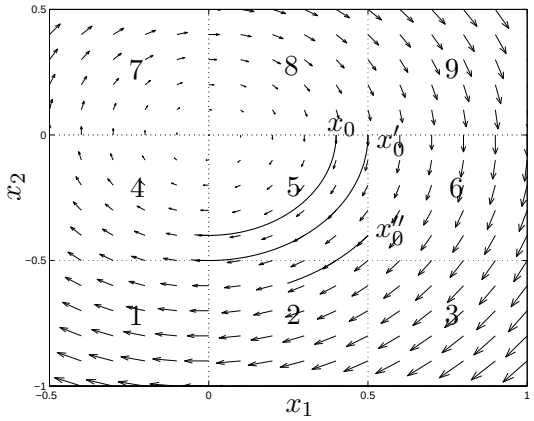

Figure 3.2: *Vector field and trajectories*

define a different set of discrete states and a mapping Q *as follows: trajectories completely in region* 1 $(2, ..., 9)$ *are assigned the discrete state* 1 $(2, ..., 9)$*. Any trajectory currently in region* 1 *and coming from region* 2 *is denoted as discrete state* $\langle 2, 1\rangle$*, etc. The set of discrete states is then defined as* $\tilde{X} = \{1, 2, ..., 9, \langle 1, 2\rangle, \langle 2, 1\rangle, ..., \langle 8, 9\rangle, \langle 9, 8\rangle\}$*. From the trajectories in Figure 3.2 it is clear that there is no path from region* 8 *to region* 2 *via region* 5*. The new transition function therefore should not allow the transition* $\langle 8, 5\rangle \rightarrow \langle 5, 2\rangle$*. In fact, if the new transition function allows a transition* $\langle a, b\rangle \rightarrow \langle b, c\rangle$ *then we are certain that there exists a trajectory in the continuous domain from region* a *to region* c *via region* b*. In terms of the old discrete states, we are certain that the sequence* a, b, c *is no spurious solution of our automaton, so our new discrete-event model of the continuous system is of a better quality*

than our previous model. However, this improved quality is at the cost of the larger number of discrete states that is grown from 9 *to* 33.

3.1.1 Isomorphism

When looking at Example 3.1.3, it is obvious that another discrete-event system can be obtained by identifying the nine regions in Figure 3.1 by $9, 8, 7, ..., 1$ instead of $1, 2, 3, ..., 9$ and adapting the transition function ϕ correspondingly. It is clear that by doing so, the dynamics of both discrete-event models are intrinsically the same, that is, no additional transitions arise and no transitions disappear. This means that different discrete event-models of a continuous system can result in the same behavior of the discrete-event models. This is formalized in the concept of isomorphism.

Definition 3.1.5 (Isomorphism) *Consider two discrete-event models of a continuous system (3.1)* $(\Sigma = (\tilde{X}, \tilde{U}, \phi), Q, S)$, *and* $(\Sigma' = (\tilde{X}', \tilde{U}', \phi'), Q', S')$. *Furthermore two invertible mappings* $T : \tilde{X} \to \tilde{X}'$, $V : \tilde{U} \to \tilde{U}'$ *are given such that* $Q' = T \circ Q$ *and* $S' = V \circ S$. *Then the two discrete-event models of the continuous system are said to be isomorphic, if for all* $\xi \in \mathcal{C}^n$ *and* $\upsilon \in \mathcal{PC}^m$ *satisfying (3.1) we have for* $\tilde{x}' = Q'(\xi) = T(Q(\xi)) = T(\tilde{x})$ *and* $\tilde{u}' = S'(\upsilon) = V(S(\upsilon)) = V(\tilde{u})$ *that*

$$\phi(\tilde{x}, \tilde{u}) = T^{-1}\phi'(T(\tilde{x}), V(\tilde{u})).$$

Note that this implies that $\phi'(\tilde{x}', \tilde{u}') = T\phi(T^{-1}(\tilde{x}'), V^{-1}(\tilde{u}'))$.

Definition 3.1.5 states that if two discrete-event models are isomorphic, then we can always switch between them without changing possible transitions. This is depicted in Figure 3.3.

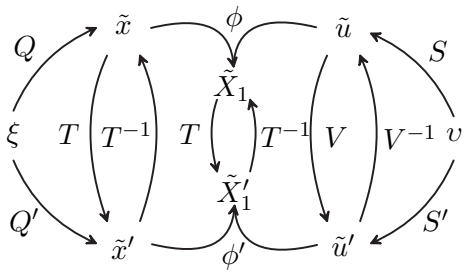

Figure 3.3: *Isomorphic discrete-event models of a continuous system*

A consequence of this isomorphism is that, for theoretical purposes, we are not concerned with the exact representation of the elements of the set of discrete states $\tilde{X}$. In the remainder of this thesis a discrete state will be represented by an n-tuple $\tilde{x}$, an integer $\bar{x}$, or a Boolean vector $\hat{x}$, see Chapter 2 for more details.

3.2 Discrete-event modelling algorithm

A continuous system described by (3.1) is given. First, we have to define a set of discrete states $\tilde{X}$ and a collection of discrete inputs $\tilde{U}$ for the discrete-event system to be constructed. Next, these discrete states and inputs are related to the continuous state space and input space by the mappings Q and S, respectively. Finally, the transition function ϕ is constructed.

3.2.1 State discretization

For each component x^i, $(i = 1, ..., n)$, a set of boundaries is given:

$$\beta_0^i < \beta_1^i < \ldots < \beta_{n_i}^i \quad (n_i \geq 1). \tag{3.3}$$

The extreme boundaries β_0^i and $\beta_{n_i}^i$ determine the region of interest in the state space:

$$\mathcal{V} = \{x \in \mathbb{R}^n | \beta_0^i \leq x^i \leq \beta_{n_i}^i, i = 1, ..., n\}.$$

We can think of the state space being partitioned into hypercubes (cells) naturally induced by the boundaries. Each hypercube can be labelled by an n-tuple $a = (a^1, \ldots, a^n)$ of integers a^i with $1 \leq a^i \leq n_i$ for each i. To be precise, the hypercube $H_x(a)$ is defined as the bounded region in $\mathbb{R}^n$ given by

$$H_x(a) := \{x \in \mathbb{R}^n | \beta_{a^i-1}^i \leq x^i \leq \beta_{a^i}^i, i = 1, ..., n\}. \tag{3.4}$$

Each such n-tuple a corresponds to a discrete state.

As explained in Definition 3.1.5 it does not matter whether we identify a discrete state $\tilde{x}$ with its corresponding n-tuple, an integer or a Boolean vector (see Figure 3.4). Therefore we choose the notation $\tilde{x}$ for a discrete state mostly, that is, $H_x(\tilde{x})$ should be read as $H_x(a)$, where $\tilde{x}$ and a are essentially the same.

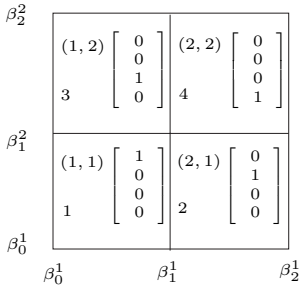

Figure 3.4: *Different discrete-state representations*

It can be seen that the set of boundaries (3.3) define $\prod_i n_i$ hypercubes, so the cardinality of the set of discrete states is equal to $\#(\tilde{X}) = \prod_i n_i$.

Two discrete states $\tilde{x}_1$ and $\tilde{x}_2$ are said to be *adjacent* if their corresponding hypercubes $H_x(\tilde{x}_1)$ and $H_x(\tilde{x}_2)$ share an $(n-1)$-dimensional boundary plane. The transition from one discrete state to another is called a *discrete event* and is denoted by $\tilde{x}_1 \rightarrow \tilde{x}_2$. We will use the following standing assumption:

ASSUMPTION 3.2.1 *Only transitions between adjacent hypercubes are allowed.*

Stated differently, the possibility is excluded that two components x^i and x^j $(i \neq j)$ of a trajectory cross a boundary at the same time instant (which only can happen if the continuous trajectory crosses edges of hypercubes). These trajectories cross a set of states with measure zero and, hence, have no influence on the generic system behavior. In practice such situations are unlikely to happen; for almost all initial states and almost all input functions Assumption 3.2.1 holds for the corresponding trajectories.

Mapping Q

The trajectory ξ evolves through several hypercubes in the state space. The mapping Q maps this signal to the set $\tilde{X}$. At each time-instant t a discrete state can be assigned to the (continuous) state $\xi(t)$. It is obvious that it is not possible to define unambiguously the relation between a discrete state and the corresponding hypercube. Since a point on a boundary plane belongs to two hypercubes, a decision procedure has to be proposed. A practically relevant and useful procedure incorporates how a trajectory has reached that particular point. This implies that Q becomes a dynamic mapping as part of the past has to be included. Since Q is a dynamic mapping, it acts on a trajectory ξ from a certain time-instant t_0 until the time-instant of interest t. Since the specification of an initial state does not involve a history, the situation can occur that it cannot be decided to which discrete state a trajectory starting and remaining on boundary planes between hypercubes belongs. One approach to overcome this situation is to assume that the discrete state at the initial time is known. However, we will opt for restricting the domain of Q to a new class of functions $\mathcal{C}^n_{x,int}$ that is relevant for the purposes in this thesis. To be specific,

$$\mathcal{C}^n_{x,int} := \{\xi \in \mathcal{C}^n \mid \exists \tilde{x} \in \tilde{X} \text{ such that } \xi(t_0) \in int(H_x(\tilde{x})), \mathcal{D}(\xi) = [t_0, t_e]\},$$

where $\mathcal{D}(\xi)$ denotes the domain[2] of a function ξ. Hence, we only consider trajectories in $\mathcal{C}^n$ that start in the interior of one of the hypercubes $H_x(\tilde{x})$. Since this class is characterized by initial conditions in the state space, the label x appears in the subscript of $\mathcal{C}^n_{x,int}$.

To define the mapping $Q : \mathcal{C}^n_{x,int} \rightarrow \tilde{X}$, it is convenient to use the n-tuple representation of a discrete state $\tilde{x} = a = (a^1, ..., a^n)$. In this way, Q

[2]In case $t_0 = -\infty$ we assume that $\lim_{t \rightarrow -\infty} \xi(t) \in int(H_x(\tilde{x}))$

can be composed from the component-wise mappings Q^i, $i = 1, ..., n$ such that $Q = (Q^1, ..., Q^n)$. Let ξ^i denote the i-th coordinate of ξ and define $H_x^i(a^i) := \{z \in \mathbb{R} \mid \beta_{a^i-1}^i \leq z \leq \beta_{a^i}^i\}$. The mapping Q^i is defined as follows:

$$Q^i(\xi) = a^i \iff \exists t^* \in \mathcal{D}(\xi) \text{ s.t. } \begin{cases} \xi^i(t^*) \in int(H_x^i(a^i)), \\ \xi^i(\tau) \in H_x^i(a^i), \forall \tau \in \mathcal{D}(\xi),\, \tau \geq t^*, \end{cases}$$

with $\xi \in \mathcal{C}_{x,int}^n$.

In words, this means that for points in the interior of a hypercube the corresponding uniquely defined discrete state applies, and that for points on the boundary plane between hypercubes the discrete state corresponds to the last hypercube that contained part of the trajectory in its interior. Assumption 3.2.1 implies that if at a time-instant t a single transition occurs, then it holds that the n-tuples $Q(\xi_{[t_0,t]})$ and $Q(\xi_{[t_0,t+\varepsilon]})$ differ in exactly one coordinate with one unit, for all sufficiently small $\varepsilon > 0$.

3.2.2 Discrete inputs

In many situations, the input u is chosen from a discrete set, for example because we are dealing with switches turning a device on or off. Other examples are valves (closed/open), a gear box of a car, and quantized outputs of a computer. In such cases, the output v is a piecewise constant function, where at time-instance t we have that $v(t) \in U = \{u_1, u_2, ..., u_k\}$. In this case, it suffices to take our mapping S to be a static mapping, *i.e.* each element in U is identified uniquely by an element of the set of discrete inputs $\tilde{U}$ by a one-to-one mapping S:

$$S(v(t)) \in \tilde{U}.$$

3.2.3 Input discretization

If the input evolves in the continuous domain then the same procedure can be followed as for the state discretization. However, special attention has to be paid to the input signals that are allowed. Since this discussion is rather technical, it can be skipped for readers who are primarily interested in the case of discrete inputs or only want to grasp the general ideas instead of details.

The difficulty with input signals in the continuous domain in case piecewise continuous functions are allowed (and this is our intention), is that it is not clear to which discrete input the signal belongs at the discontinuity. This problem is similar as we have seen for the initial condition lying on the boundary between two hypercubes in the case of the state discretization. Now, at each discontinuity of the input signal a new 'initial input' arises. To overcome this problem, we restrict our inputs to signals for which at the discontinuity, the input signal starts in the interior of some hypercube.

To formalize this, for each input component u^i, $(i = 1, ..., m)$, a set of boundaries is given:

$$\gamma_0^i < \gamma_1^i < \ldots < \gamma_{m_i}^i \quad (m_i \geq 1). \tag{3.5}$$

Similar to the state space, each hypercube induced by the boundaries in the input space is labelled by an m-tuple $b = (b^1, \ldots, b^m)$ of integers b^i with $1 \leq b^i \leq m_i$ for each i, such that the corresponding hypercube is defined by

$$H_u(b) := \{u \in \mathbb{R}^m | \gamma_{b^i-1}^i \leq u^i \leq \gamma_{b^i}^i, i = 1, ..., m\}.$$

In this case the set of boundaries (3.5) define $\prod_i m_i$ hypercubes. Each m-tuple corresponds to a discrete input and similar to the discrete states, discrete inputs will just be denoted by $\tilde{u}$ independently of the representation.

Let $\{t_i\} \subset \mathcal{D}(\upsilon) = [t_0, t_e]$ be the set of time-instants where discontinuities occur in the signal υ and let t_0 be included. The domain of the mapping S is restricted to the space $\mathcal{PC}_{u,int}^m$ defined as

$$\mathcal{PC}_{u,int}^m := \{\upsilon \in \mathcal{PC}^m \mid \forall t_j \in \{t_i\}, \exists \tilde{u} \in \tilde{U} \text{ s.t. } \lim_{t \downarrow t_j} \upsilon(t) \in int(H_u(\tilde{u}))\}.$$

Now, $S : \mathcal{PC}_{u,int}^m \to \tilde{U}$ is defined as $S := (S^1, ..., S^m)$, with

$$S^i(\upsilon) = b^i \iff \exists t^* \in \mathcal{D}(\upsilon) \text{ s.t. } \begin{cases} \upsilon^i(t^*) \in int(H_u^i(b^i)), \\ \upsilon^i(\tau) \in H_u^i(b^i), \forall \tau \in \mathcal{D}(\upsilon),\ \tau \geq t^*, \end{cases}$$

with $\upsilon \in \mathcal{PC}_{u,int}^m$ and $H_u^i(b^i)$ defined in a similar way as $H_x^i(a^i)$.

3.2.4 Transition function

Next, the possibly nondeterministic (partial) transition function $\phi : \tilde{X} \times \tilde{U} \to 2^{\tilde{X}}$ has to be determined such that Condition 3.1.2 is satisfied.

LEMMA 3.2.2 *A DES of the continuous system (3.1) with S, Q as above is obtained iff for all $\tilde{x}_1, \tilde{x}_2 \in \tilde{X}$, $\tilde{x}_1 \neq \tilde{x}_2$ and $\tilde{u} \in \tilde{U}$, ϕ satisfies the following condition.*

CONDITION 3.2.3 $\tilde{x}_2 \in \phi(\tilde{x}_1, \tilde{u}) \iff \exists \xi, \upsilon, t^*$ *such that* $Q(\xi_{[t_0,t^*]}) = \tilde{x}_1$, $\lim_{t \downarrow t^*} Q(\xi_{[t_0,t]}) = \tilde{x}_2$, *and* $\lim_{t \downarrow t^*} S(\upsilon_{[t_0,t]}) = \tilde{u}$. *Here, $\xi \in \mathcal{C}_{x,int}^n$ is the solution of (3.1) for the input $\upsilon \in \mathcal{PC}_{u,int}^m$.*

Condition 3.2.3 is a reformulation of Condition 3.1.2 in the sense that only trajectories of the continuous system with one discrete transition are considered. Note that this is without loss of generality as trajectories of the

continuous system that contain more than one transition can be split in appropriate pieces. Since this gives a condition that is necessary and sufficient for the existence of a single transition $\tilde{x}_1 \to \tilde{x}_2$ for input $\tilde{u}$, it can be used to construct a discrete-event model of a continuous system.

REMARK 3.2.4 *Note that Condition 3.2.3 is not imposed on transitions between identical discrete states $\tilde{x}_1 = \tilde{x}_2$ (called 'self-loops'). Hence, in principle self-loops might be either included or omitted. We decided to omit them as no additional information is added to the model. Indeed, for any input a transition between identical states (i.e. staying in the same hypercube) is possible.*

3.2.5 Computing the transition function

The idea for using Lemma 3.2.2 is to examine the flow generated by the dynamical system (3.1) along the boundary separating the two regions that correspond to two adjacent discrete states.

CONDITION 3.2.5 *Consider two adjacent states $\tilde{x}_1 = (\tilde{x}^1, ..., \tilde{x}^r, ..., \tilde{x}^n)$ and $\tilde{x}_2 = (\tilde{x}^1, ..., \tilde{x}^r + 1, ..., \tilde{x}^n)$, such that $\{x \in \mathbb{R}^n \mid x^r = \beta_j^r\}$ is the separating hyperplane and $x \in H_x(\tilde{x}_1) \Longrightarrow x^r \leq \beta_j^r$ and $x \in H_x(\tilde{x}_2) \Longrightarrow x^r \geq \beta_j^r$.*

Note that j is equal to the r-th coordinate of the n-tuple $\tilde{x}_1$, *i.e.* $j = \tilde{x}^r$. Denote in (3.1) the r-th coordinate of f by f^r. Then we have the following result.

THEOREM 3.2.6 *Given $\tilde{x}_1$ and $\tilde{x}_2$ as in Condition 3.2.5. The transition $\tilde{x}_1 \to \tilde{x}_2$ is possible with the input $\tilde{u}$ if and only if*

$$\exists x \in H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2) \text{ and } \exists u \in H_u(\tilde{u}) \text{ such that } f^r(x,u) > 0.$$

For the proof of this we will use a result originally presented in 1942 by Nagumo (Nagumo 1942) and reformulated in (Blanchini 1999) in the form it will be used here (see also (Aubin and Cellina 1984) for more details).

THEOREM 3.2.7 (NAGUMO) (Blanchini 1999) *Consider the system given by $\dot{x} = f(x(t), u(t))$, with $u(t) \in \mathcal{U}$, where $\mathcal{U} \subset \mathbb{R}^n$ is a compact set. Assume that, for each initial condition in a set $\mathcal{X} \subseteq \mathbb{R}^n$, it admits a globally unique solution. Let $K \subseteq \mathcal{X}$ be a closed and convex set. Then the set K is positively invariant (i.e. $x(0) \in K \Longrightarrow \forall t \geq 0,\ x(t) \in K$) if and only if*

$$\forall x \in \partial K, \forall u \in \mathcal{U}, \liminf_{h \downarrow 0} \frac{dist(x + hf(x,u), K)}{h} = 0.$$

Note that the result does not depend on the norm that induces the distance $dist(x, K) := \min_{z \in K} \|x - z\|$.

The result has the following geometrical interpretation (Blanchini 1999). If for all $x \in \partial K$ the derivative $\dot{x}$ 'points inside or is tangent to K', then the trajectory ξ remains in K.

Proof. (**Theorem 3.2.6)** The *if* part follows from continuity of $f(x, u)$. Suppose that for $x_0 \in H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$ and $u \in H_u(\tilde{u})$ it holds that $f^r(x_0, u) > 0$. Then there exists $\delta > 0$ such that $f^r(x, u) > 0$ for all $x \in B(x_0, \delta) := \{x \mid \|x - x_0\| < \delta\}$ and $B(x_0, \delta) \cap int(H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)) \neq \emptyset$. This implies that there always exists $x_0' \in int(H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2))$ for which $f^r(x_0', u) > 0$.

By continuity, there exists an neighborhood V of x_0' for which it holds that $V \subseteq int(H_x(\tilde{x}_1) \cup H_x(\tilde{x}_2))$ and $f^r(x_0, u) > 0$ for all $x \in V$. Let $x(t)$ be a solution of the system with input $u(t) = u$ such that $x(0) = x_0'$ and let $\varepsilon > 0$ be such that $x(t) \in V$ for all $t \in (-\varepsilon, \varepsilon)$. Then $x^r(t)$ is strictly increasing on the same interval and consequently $x(-\varepsilon) \in int(H_x(\tilde{x}_1))$ and $x(\varepsilon) \in int(H_x(\tilde{x}_2))$ from which it follows that the transition $\tilde{x}_1 \to \tilde{x}_2$ has occurred.

For the *only if* part we take $\mathcal{X} = H_x(\tilde{x}_1) \cup H_x(\tilde{x}_2)$, $K = H_x(\tilde{x}_1)$ and $\mathcal{U} = H_u(\tilde{u})$. Now assume that $f^r(x, u) \leq 0$ for all $x \in K$ and $u \in H_u(\tilde{u})$, then from Theorem 3.2.7 it follows that K is positively invariant implying that no transition is possible. Indeed, since $f^r(x, u) \leq 0$ for all $x \in H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2) \subset K$ and realizing that the possibility of a transition does not depend on $f^r(x, u)$ for $x \notin H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$ shows that no transition is possible. ■

The consequence of this result is that, to assess whether a transition between two adjacent states is possible or not, we need to look at the sign of a coordinate of f on the separating boundary. Concretely, if we want to decide whether a transition is possible from $\tilde{x}_1$ to $\tilde{x}_2$, we shall start by checking the so-called extremal points of $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$, *i.e.* the points x satisfying

$$x^i = \beta^i_{\tilde{x}_1^i - 1} \text{ or } \beta^i_{\tilde{x}_1^i}\ , \quad i \neq r,$$

$$x^r = \beta^r_{\tilde{x}_1^r}.$$

If the value of f^r in any of these points is positive, we conclude by continuity that there is also a point in the (relative) interior of $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$ at which f^r takes a positive value and the transition from $\tilde{x}_1$ to $\tilde{x}_2$ is possible. In the case that the values of f^r in all these points (there are 2^{n-1} of them) are non-positive, it is necessary to search for a possible positive value of f^r on $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$. This can be conveniently performed by using an optimization procedure which searches the maximum of f^r on $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$. If the maximum value is non-positive as well, then we can conclude that the transition from $\tilde{x}_1$ to $\tilde{x}_2$ is impossible. If the maximum value is positive, then the transition is possible.

3.3 Systems with outputs

Consider the continuous-time system described by the set of differential equations and the output equation

$$\begin{aligned} &\dot{x}(t) = f(x(t), u(t)), \quad x(t_0) = x_0, \\ &y(t) = g(x(t)), \end{aligned} \tag{3.6}$$

where $x(t) \in \mathbb{R}^n$ is the state variable, $u(t) \in \mathbb{R}^m$ is the input and $y(t) \in \mathbb{R}^l$ is the output.

DEFINITION 3.3.1 *A discrete-event model of the system (3.6) is given by an output automaton*

$$\Sigma = (\tilde{X}, \tilde{U}, \tilde{Y}, \phi, h),$$

together with the mappings

$$\begin{aligned} &Q : \mathcal{C}^n_{x,int} \to 2^{\tilde{X}}, \\ &R : \mathcal{C}^l_{y,int} \to \tilde{Y}, \\ &S : \mathcal{PC}^m_{u,int} \to \tilde{U}, \end{aligned}$$

such that the following holds:

CONDITION 3.3.2 *Let $\upsilon \in PC^0([t_0, t_e], \mathbb{R}^m)$ be an input signal applied to the system (3.6) from time t_0 until time t_e. Let $\theta \in C^0([t_0, t_e], \mathbb{R}^l)$ denote an output signal satisfying (3.6) for input υ. For all $t_i \in [t_0, t_e]$ satisfying $R(\theta_{[t_0,t_i]}) \neq R(\theta_{[t_0,t_i+\varepsilon]})$ for all sufficiently small $\varepsilon > 0$ it holds that $\tilde{y}_2 = h(\tilde{x}_2)$, $\tilde{x}_2 \in \phi(\tilde{x}_1, \tilde{u})$ and $\tilde{y}_1 = h(\tilde{x}_1)$, where $\tilde{y}_1 := R(\theta_{[t_0,t_i]})$, $\tilde{y}_2 := \lim_{t \downarrow t_i} R(\theta_{[t_0,t]})$, $\exists \tilde{x}_1 \in Q(\xi_{[t_0,t_i]})$, $\exists \tilde{x}_2 \in \lim_{t \downarrow t_i} Q(\xi_{[t_0,t]})$ and $\tilde{u} = S(\upsilon_{[t_0,t_i]})$.*

For the same reason as for the state and input, the mapping R is defined on output signals starting in the interior of some hypercube in the output space. Hence, the class $\mathcal{C}^l_{y,int}$ is defined similarly as $\mathcal{C}^n_{x,int}$. Note that Q maps (pieces of) trajectory to *sets* of discrete states instead of only one discrete state. This is because a hypercube in the state space can be associated with more than one output. Since each discrete state $\tilde{x}$ can only correspond to one output $\tilde{y}$, the hypercube has to be related with more than one discrete states.

3.3.1 Building the discrete-event model

To build a discrete-event model of a continuous system with outputs again the set of discrete states $\tilde{X}$, the set of discrete inputs $\tilde{U}$, and the transition function ϕ need to be specified. But now also the set of discrete outputs $\tilde{Y}$

and the output map h have to be defined. Finally we need the map R which maps the output space to the set of discrete outputs.

For the construction of an output automaton the set of discrete states $\tilde{X}$ must be chosen in a different way as for the automaton in Section 3.2. Also the definition of the mapping Q needs to be adapted. The reason for these differences is that with the old definitions the situation can occur that a discrete output changes while the discrete state still remains the same. This behavior cannot be captured in an output automaton.

EXAMPLE 3.3.3 *Suppose two equivalent continuous systems are given and described by (3.6) such that state x corresponds to the first system and state z to the second system. Both systems have the same output y. See Figure 3.5 for the two related discretized state- and measurement spaces. From Figure 3.5 it*

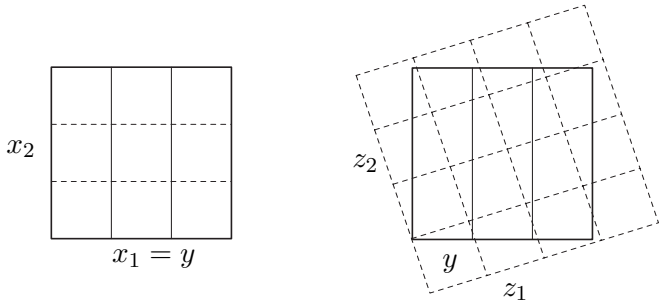

Figure 3.5: *Equal output and different state-coordinates*

can be seen that for the second system with state z, one discrete state is related with several discrete outputs, whereas for the first system each discrete state is only related with one discrete output. For the second system it can happen that the discrete output $\tilde{y}$ changes while the continuous state still remains inside one hypercube, such that no transition occurs between two discrete states.

Discrete outputs

The discrete outputs are chosen in a similar way as the discrete states and the discrete inputs in Section 3.2, *i.e.* for each component $y^i, i \in \{1, ..., l\}$ of the output space $\mathbb{R}^l$ a set of boundaries is defined:

$$\rho_0^i < \rho_1^i < ... < \rho_{k_i}^i \qquad (k_i \geq 1). \tag{3.7}$$

Again, each hypercube induced by these boundaries is labeled by an l-tuple $c = (c^1, ...c^l)$ with integers $c^i \in \{1, ...k_i\}$. The hypercube corresponding to c is now defined as the bounded region in $\mathbb{R}^l$ described by

$$H_y(c) := \{y \in \mathbb{R}^l | \rho_{c^i-1}^i \leq y^i \leq \rho_{c^i}^i, i = 1, ..., l\}.$$

With the set of boundaries (3.7) in the output space, we have $\prod_i k_i$ hypercubes. Each hypercube corresponds to a discrete output, implying that the cardinality $\#(\tilde{Y})$ of the set of discrete outputs is equal to $\prod_i k_i$.

Mapping R

To map the signal that lives in the continuous output space $\mathbb{R}^l$ to the set of discrete outputs a similar approach is followed as with the mappings Q and S in the previous sections. With a similar notation, $R := (R^1, ..., R^l)$ is composed from component-wise mappings R^i, $i = 1, ..., l$ defined as

$$R^i(\theta) = c^i \iff \exists t^* \in \mathcal{D}(\theta) \text{ s.t. } \begin{cases} \theta^i(t^*) \in int(H_y^i(c^i)), \\ \theta^i(\tau) \in H_y^i(c^i), \forall \tau \in \mathcal{D}(\theta),\ \tau \geq t^*, \end{cases}$$

with $\theta \in \mathcal{C}^l_{y,int}$ being a continuous function with closed domain and $\theta(t_0)$ in the interior of some $H_y(\tilde{y})$.

Output map and discrete states

Next, the set of discrete states $\tilde{X}$ is related with the set of discrete outputs $\tilde{Y}$. That is, given a discrete state $\tilde{x}$ the corresponding discrete output $\tilde{y}$ has to be determined. As mentioned before, each discrete state $\tilde{x}$ may only correspond to one particular discrete output $\tilde{y}$, since otherwise (measured) outputs can change while there is no change of the discrete state. Since this cannot be modelled by an output automaton, the set of discrete states $\tilde{X}$ is constructed such that each discrete state only is related to one discrete output. To achieve this, the construction of the set of discrete states and the construction of the output map are combined. First a temporary set of discrete states $\tilde{X}_{temp}$ is defined in a similar way as described in Section 3.2. For each discrete state $\tilde{x}$ in the set $\tilde{X}_{temp}$ it is decided what the corresponding (set of) output(s) $\tilde{y}$ is to the associated hypercube $H_x(\tilde{x})$ and the output function $g(x)$. From this, the final set of discrete states $\tilde{X}$ and the output map h can be determined. The procedure is as follows.

step 1 Define the temporary set of discrete states $\tilde{X}_{temp}$ from the set of boundaries (3.3) as described in Section 3.2.

step 2 Determine for each discrete state $\tilde{x} \in \tilde{X}_{temp}$ the corresponding region in $\mathbb{R}^n$, *i.e.* the hypercube $H_x(\tilde{x})$. Map this set to the output space by means of the output function $g(x)$, *i.e.* compute $g(H_x(\tilde{x}))$. We compute now the (set of) discrete output(s) that corresponds to this region in $\mathbb{R}^l$,

i.e. compute $R(g((H_x(\tilde{x}))))$[3]. The result is that each discrete state $\tilde{x}$ is related to a set of corresponding discrete outputs $\tilde{y}$. This composite relation is denoted $h_{temp} : \tilde{X}_{temp} \to 2^{\tilde{Y}}$. See Figure 3.6.

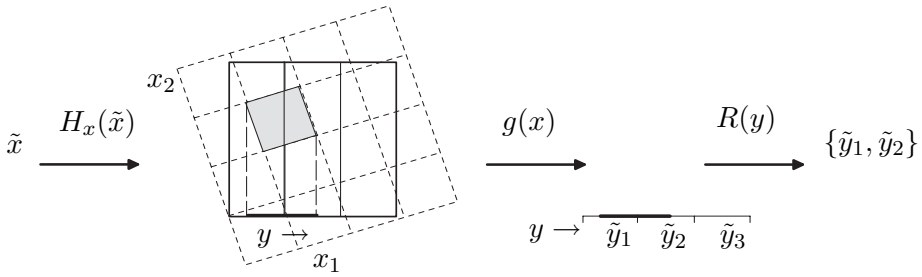

Figure 3.6: *From discrete state to discrete output*

step 3 Each state $\tilde{x} \in \tilde{X}_{temp}$ that corresponds to more than one discrete output, say q outputs, is replaced by q new discrete states. To introduce some notation, assume that if $\tilde{x} = 1$ corresponds to $\tilde{y} = \{3, 4, 5\}$, then $\tilde{x} = 1$ is replaced by the states $\tilde{x}_1 = 1_3$, $\tilde{x}_2 = 1_4$, and $\tilde{x}_3 = 1_5$. This results in the final set of discrete states $\tilde{X}$. In this way each discrete state $\tilde{x} \in \tilde{X}_{temp}$ is related to (a set of) discrete state(s) $\{\tilde{x}_i\} \subset \tilde{X}$ described by a relation $\mathcal{R} : \tilde{X}_{temp} \to 2^{\tilde{X}}$,

$$\mathcal{R}(\tilde{x}) = \{\tilde{x}_{\tilde{y}} \mid \tilde{y} \in h_{temp}(\tilde{x})\}.$$

step 4 To each of the discrete states $\tilde{x}_{\tilde{y}} \in \mathcal{R}(\tilde{x})$, $\tilde{x}_{\tilde{y}} \in \tilde{X}$, $\tilde{x} \in \tilde{X}_{temp}$, is uniquely assigned one of the outputs $\tilde{y} \in \tilde{Y}$, such that $\tilde{y} = h(\tilde{x}_{\tilde{y}}) \in h_{temp}(\tilde{x})$. This defines the output map h. For example, $h(1_3) = 3$, $h(1_4) = 4$, and $h(1_5) = 5$. For those discrete states $\tilde{x}$ that corresponded to just one discrete output $\tilde{y}$, naturally $h(\tilde{x}) = \tilde{y}$. As a result, only one discrete output $\tilde{y}$ is assigned to each discrete state $\tilde{x} \in \tilde{X}$ by the output map h.

Mapping Q

From the construction of the set of discrete states it follows that a hypercube as induced by the boundaries (3.3) may correspond to more than one discrete state. Therefore the mapping Q also has to be adapted. First, the temporary

[3]Each point $y \in g(H_x(\tilde{x}))$ should be associated with the function $\theta \in \mathcal{C}^l_{y,int}$ defined by $\theta(t) = y$ for all $t \in [a, b]$. The reason for this is that the discretization mapping R maps on functions only and not on points.

mapping $Q_{temp} : \mathcal{C}^n_{x,int} \to \tilde{X}_{temp}$ is constructed in the usual manner. Then, $Q : \mathcal{C}^n_{x,int} \to 2^{\tilde{X}}$ is defined as the composite mapping:

$$Q = \mathcal{R} \circ Q_{temp}.$$

In words: Q maps the (continuous) states in a hypercube to the set of all discrete states that are related to that hypercube by step 3, as discussed before.

Transition function

Finally the transition function is constructed such that Condition 3.3.2 is satisfied with the sets and mappings as defined before. For this, the following algorithm is executed.

1. Compute the temporary transition function ϕ_{temp} by determining the transitions between the discrete states of the set $\tilde{X}_{temp}$. This can be done by applying the algorithm as described in Section 3.2.

2. For all $\tilde{x}_1, \tilde{x}_2 \in \tilde{X}_{temp}$ and $\tilde{u} \in \tilde{U}$ a transition rule is defined for ϕ as follows

$$\tilde{x}_2 \in \phi_{temp}(\tilde{x}_1, \tilde{u}) \Longrightarrow \tilde{x}_{2_i} \in \phi(\tilde{x}_{1_j}, \tilde{u}),\ \forall \tilde{x}_{1_j} \in \mathcal{R}(\tilde{x}_1), \forall \tilde{x}_{2_i} \in \mathcal{R}(\tilde{x}_2).$$

3. Add transitions between states $\tilde{x}_i, \tilde{x}_j \in \mathcal{R}(\tilde{x})$ by the rule

$$\tilde{x}_j \in \phi(\tilde{x}_i, \cdot) \Longleftrightarrow h(\tilde{x}_j),\ h(\tilde{x}_i) \text{ are neighbors.}$$

By this procedure, the transition function ϕ is defined such that Condition 3.3.2 is satisfied. Clearly, this procedure will induce many spurious solutions.

3.3.2 Choice of the state coordinates

The difference between systems with and without outputs is that in the former case we (usually) have some freedom to choose our state-coordinates while in the latter case the state-coordinates are completely fixed. The freedom to choose the state-coordinates can be used to influence the quality and the complexity of the discrete-event model of the continuous system. The choice of the state-coordinates is important for the quality of the discrete-event model since the following scheme applies

$$\begin{array}{lccc} \text{continuous model} & \dot{x} = f(x,u) & \underrightarrow{T} & \dot{z} = f'(z,u) \\ & \downarrow & & \downarrow \\ \text{discrete-event model} & \Sigma = (\tilde{X}, \tilde{U}, \phi) & & \Sigma = (\tilde{Z}, \tilde{U}, \phi') \\ & Q, S & & Q', S' \end{array}$$

This means that two systems that are equivalent in the continuous domain (that is, they only differ by a similarity transformation of the form $z = Tx$) do not need to have the same discrete-event models. Also, the measured output of the continuous system (3.6) define the discrete outputs, which in turn are connected with the discrete states by the output map h.

The choice of discrete states influences the complexity for determining the output map h. By some clever choices, the construction of a discrete-event model of a continuous system with outputs can be simplified considerably, as will be shown in the next subsections.

3.3.3 Nonlinear systems with linear output maps

To obtain simpler procedures for computing the output map h, first the class of nonlinear systems with linear output maps is considered, which are described by

$$\begin{aligned} &\dot{x}(t) = f(x(t), u(t)), \quad x(t_0) = x_0, \\ &y(t) = Cx(t), \end{aligned} \tag{3.8}$$

where $x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m$, and $y(t) \in \mathbb{R}^l$ and C has full row-rank.

To compute the discrete-event model of this system, first a variable $z \in \mathbb{R}^{n-l}$ is chosen such that $z = Dx$, where $D \in \mathbb{R}^{(n-l)\times n}$ is a matrix chosen rendering $T = [C^T\ D^T]^T$ invertible. With this, the following equivalent system can be obtained

$$\begin{bmatrix} \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} C \\ D \end{bmatrix} f(\begin{bmatrix} C \\ D \end{bmatrix}^{-1} \begin{bmatrix} y(t) \\ z(t) \end{bmatrix}, u(t)).$$

By defining the new state $x^* := Tx = [y^T\ z^T]^T$, the system (3.8) is represented equivalently by

$$\begin{aligned} &\dot{x}^*(t) = f^*(x^*(t), u(t)), \quad x^*(t_0) = x_0^*,\ u(t) \in \mathbb{R}^m, \\ &y(t) = [I\ 0]x^*(t), \qquad\qquad y(t) \in \mathbb{R}^l. \end{aligned} \tag{3.9}$$

For the system in this state space form, a discrete-event model is constructed. The set of discrete inputs $\tilde{U}$ follows from the set of bounds (3.5). The set of discrete outputs $\tilde{Y}$ is determined by the set of boundaries (3.7) reflecting for instance the positions of the measurements (sensors). Recall that each discrete output $\tilde{y} \in \tilde{Y}$ is related to an l-tuple $c = (c_1, ..., c_l)$ defining the hypercube $H_y(c)$ in $\mathbb{R}^l$. According to the partitioning of the state space $x^* = [y^T\ z^T]^T$, for the first l coordinates $x^{*,i}$, $i = 1, ..., l$, the same set of boundaries (3.7) as for the output space can be used. For the last $(n - l)$ coordinates a set of additional boundaries $\{\beta_j^i\}$ can be chosen freely. So, $\beta_j^i = \gamma_j^i$ for $i = 1, ..., l$ and $j = 1, ..., n_i$, while β_j^i can be chosen freely for $i = l + 1, ...n$. Each n-tuple

$a = (a_1, ..., a_n)$ assigned to a hypercube in the state space can be partitioned as $a = (c, d)$, where c is the l-tuple as for the outputs and d is an $(n-l)$-tuple following from the additional boundaries β_j^i, $i = l+1, ..., n$ and $j = 0, ..., n_i$. This defines the set of discrete states $\tilde{X}$ for the transformed system. In this way every discrete state is already uniquely assigned to a discrete output. There is no need for a temporary set of discrete states $\tilde{X}_{temp}$ or a temporary output map h_{temp} since the output map h is defined readily by

$$h(a) = c \Longleftrightarrow a = (c, d) \text{ for some } d.$$

Note that each discrete state is related to only one discrete output. Therefore the transition function ϕ can be determined by applying the algorithm of Section 3.2 to the set of differential equations $\dot{x}^* = f^*(x^*, u)$ and the specified boundaries. This concludes the construction of the discrete-event model, which is more easy than following the complicated procedure described in Subsection 3.3.1.

The freedom in choosing D and the boundaries for discretizing z can be used for constructing the best possible automaton meeting the requirements one has. For linear, observable systems, a simple way to find D for the equivalent system (3.9) is to choose the rows of D from the (regular) observability matrix.

3.3.4 Linear systems

For linear systems, one can follow the same approach as for nonlinear systems with linear output maps. However, in some special cases, it is even possible to rewrite the continuous system such that the set of discrete outputs $\tilde{Y}$ and the set of discrete states $\tilde{X}$ can be taken equal. In fact, this implies that we can apply a state reduction of the continuous system where the new states are equal to the measurements y. For these kind of systems the proposed discrete-event modelling algorithm for systems without outputs can be applied readily, since the construction of the output function h is trivial.

Consider the linear time-invariant system given by

$$\begin{aligned} &\dot{x}(t) = Ax(t) + Bu(t), \quad x(t_0) = x_0, \\ &y(t) = Cx(t), \end{aligned} \tag{3.10}$$

where $x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m, y(t) \in \mathbb{R}^l$, and C has full row-rank.

PROPOSITION 3.3.4 *Let the linear system (3.10) be given. The variable y is governed by equations of the form*

$$\dot{y}(t) = Fy(t) + Gu(t)$$

iff $\ker(C)$ *is A-invariant*[4].

[4] A vector space $\mathcal{V}$ is A-invariant if $A\mathcal{V} \subseteq \mathcal{V}$.

Proof. The *if*-part: by using the pseudo-inverse of C (or any other left-inverse), we can express the state x as a function of y and a 'free' variable w as follows

$$x = C^T(CC^T)^{-1}y + Mw,$$

with

$$\mathrm{Im}(M) = \ker(C).$$

By differentiating the output equation and substituting x we get

$$\dot{y} = CAC^T(CC^T)^{-1}y + CAMw + CBu.$$

Since $A(\ker(C)) \subseteq \ker(C)$ we obtain that $CAM = 0$ and thus $\dot{y} = Fy + Gu$ for $F = CAC^T(CC^T)^{-1}$ and $G = CB$.

For the *only if*-part suppose that $\dot{y}(t) = Fy(t) + Gu(t)$ and choose $x(0) = x_0 \in \ker(C)$. Hence, $y(0) = Cx_0 = 0$ and take $u(t) = 0$ for all t. This results in $y(t) = 0$ for all $t \in \mathbb{R}$. For the original system,

$$\begin{aligned} \dot{x}(t) &= Ax(t), \\ 0 = y(t) &= Cx(t) \end{aligned}$$

for initial condition $x(0) \in \ker(C)$, which implies that $x(t) \in \ker(C)$ for all $t \in \mathbb{R}$. This shows that $\ker(C)$ is A-invariant because x_0 was chosen in $\ker(C)$. ■

EXAMPLE 3.3.5 *Given the system (3.10) where the output matrix C can be written as $C = C'T^{-1}$, where $C' = [I\ 0]$ and T is a similarity transformation such that $T^{-1}AT = \Lambda$, where Λ is a diagonal matrix. Then it is clear to see that*

$$\ker(C) = \ker(\begin{bmatrix} I \\ 0 \end{bmatrix} T^{-1}) = \mathrm{Im}(T \begin{bmatrix} 0 \\ I \end{bmatrix}).$$

Since $A = T\Lambda T^{-1}$ we have that $A\ker(C)$ results in

$$\begin{aligned} AT \begin{bmatrix} 0 \\ I \end{bmatrix} &= T\Lambda T^{-1} T \begin{bmatrix} 0 \\ I \end{bmatrix} \\ &= T \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} \begin{bmatrix} 0 \\ I \end{bmatrix} \\ &= T \begin{bmatrix} 0 \\ \Lambda_2 \end{bmatrix} \subseteq \mathrm{Im}(T \begin{bmatrix} 0 \\ I \end{bmatrix}) = \ker(C). \end{aligned}$$

From which we get that $\ker(C)$ is A-invariant. It can be seen that y is governed by $\dot{y} = \Lambda_1 y + CBu$, with $y(t_0) = Cx_0$.

3.4 Computational effort

A disadvantage of the state discretization of continuous plants is the computational effort which is necessary to obtain these models. The underlying combinatorial growth characteristic is known as the state-explosion problem. It is straightforward to compute the number of optimizations NO that is necessary to compute a discrete-event model of a continuous system given the discretization of the state space and input space, see Figure 3.7.

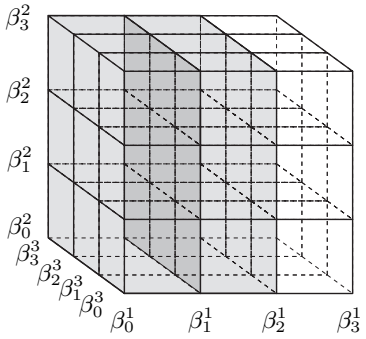

Figure 3.7: *The boundary plane $x^1 = \beta_1^1$ separates $3 \cdot 3$ pairs of states*

For each discrete input (*i.e.* $\prod_{k=1}^{m} m_k$ times) the following computations have to be performed for each coordinate (*i.e.* n times); for the i-th coordinate, $n_i - 1$ boundary planes have to be checked and for each such boundary plane there are $\prod_{j \neq i} n_j$ boundaries $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$ separating two hypercubes. Generally, for each such boundary an optimization has to be performed. With this, the total number of optimizations NO becomes:

$$NO = \sum_{i=1}^{n} \left((n_i - 1)(\prod_{j \neq i} n_j)(\prod_{k=1}^{m} m_k) \right). \tag{3.11}$$

Furthermore, in the most general case when also the input space is continuous, each optimization is performed over $m(n-1)$ variables.

Two methods will now be discussed that can be used to reduce the computational effort. The first one exploits the properties of a linear system to efficiently compute the possible transitions. The second method can be used for both linear and nonlinear systems and exploits (possible) sparsity of a system.

3.4.1 Linear systems

For linear systems the computational effort can be reduced by exploiting the fact that for each coordinate transitions in the positive and negative directions

are separated by a single hypersurface in the state-input space. Therefore, no (numerical) optimizations are involved. Furthermore, it is not necessary to determine the possibility for a transition between each (adjacent) pair of discrete states, since in one step the possibility of several transitions are immediately decided.

For linear systems we have

$$\dot{x}(t) = Ax(t) + Bu(t), \qquad x(t_0) = x_0, \tag{3.12}$$

with $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$. First, we define $z := [x^T, u^T]^T \in \mathbb{R}^{n+m}$, with $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$. Also the space $\mathbb{R}^{n+m}$ is partitioned into hypercubes defined by the boundaries β_i^j, and γ_i^j[5]. Furthermore, the $(n+m) \times (n+m)$-dimensional matrix $\bar{A}$ is defined as

$$\bar{A} = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix}.$$

If we use the $(n+m)$-dimensional unit-vector e_k, then the first derivative of the k-th element in a point $x \in \mathbb{R}^n$ and for $u \in \mathbb{R}^m$ can be written as

$$\dot{x}^k = e_k^T \bar{A} z.$$

In the $(n+m)$-dimensional space $\mathbb{R}^{n+m}$, the derivative of the k-th component of x $(1 \leq k \leq n)$, defines two half-spaces, one where the sign of the derivative for that component is positive and one where the sign of the derivative is negative. These half-spaces are separated by a hyperplane whose elements result in a derivative zero in the specific coordinate, *i.e.* $\dot{x}^k = 0$. This observation will be the basis for a computationally attractive method for obtaining the transition function ϕ.

The $(n+m-1)$-dimensional hyperplane in $\mathbb{R}^{n+m}$ on which the derivative for the k-th component is zero, is defined by

$$N_k := \{z \in \mathbb{R}^{n+m} \mid e_k^T \bar{A} z = 0\}.$$

For each discrete input it has to be decided separately which transitions are possible. Hence, fix $\tilde{u} \in \tilde{U}$. We will use the n-tuple representation of a discrete state, *i.e.* $\tilde{x} = (a^1, ..., a^n)$. Now, fix a^j for $j \neq k$. Consider all discrete states from the set $\tilde{S} := \{\tilde{x} |\ \tilde{x} = (a^1, ..., a^k, ..., a^n),\ 1 \leq a^k \leq n_k\}$ and the set of corresponding points in $\mathbb{R}^{n+m}$ (a column of hypercubes) which we will call $Z_{\tilde{S}}$ (see Figure 3.8 for $k = 2, \tilde{u} = 1$ and $a^1 = 2$).

For all the discrete states in $\tilde{S}$ it is possible to determine in one step the possible transitions in the e_k-direction for the given discrete input $\tilde{u}$. For a

[5]In case of discrete inputs, the procedure to be explained is repeated for each single discrete input $\tilde{u} \in \tilde{U}$ and z is defined as $z := [x^T, \tilde{u}^T]^T$.

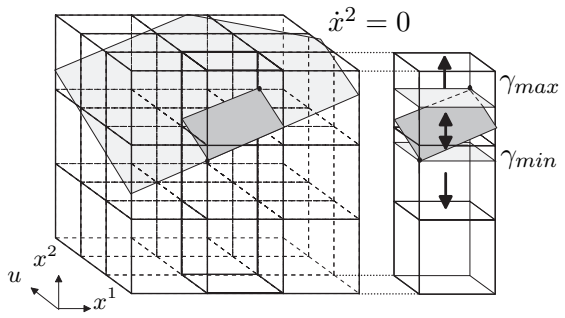

Figure 3.8: *State-input space*

particular discrete state labelled by, say $(a^1, ..., a^k, ..., a^n)$, transitions to states corresponding to $(a^1, ..., a^k - 1, ..., a^n)$ and $(a^1, ..., a^k + 1, ..., a^n)$ have to be examined. For this purpose, the k-th component $z^k = \gamma e_k$ of the elements z on the hyperplane N_k are, if possible, expressed as a function of the remaining coordinates $z^- \in \ker(e_k^T)$. Substituting $z = \gamma e_k + z^-$ in $e_k^T \bar{A} z = 0$ results for the case where $e_k^T \bar{A} e_k = a_{kk} \neq 0$ in the following expression for γ:

$$\gamma = \frac{-1}{e_k^T \bar{A} e_k} e_k^T \bar{A} z^-, \; z^- \in \ker(e_k^T).$$

Now, for a point $z = z^* + \delta e_k$ with $z^* \in N_k$, and $\delta \in \mathbb{R}$ we have that

$$\begin{aligned} sign(\dot{x}^k) &= sign(\underbrace{e_k^T \bar{A} z^*}_{=0} + \delta e_k^T \bar{A} e_k) \\ &= sign(\delta e_k^T \bar{A} e_k) = sign(\delta) sign(e_k^T \bar{A} e_k) \\ &= sign(\delta) sign(a_{kk}). \end{aligned} \tag{3.13}$$

This implies, that if for a point z^* on the hyperplane N_k, we move towards the (positive) e_k-direction, then the derivative in the new point $z = z^* + \delta e_k$ has a component in the e_k-direction and the sign of this component is equal to the sign of the a_{kk}-element of the system matrix A (*i.e.* $sign(a_{kk})$). If from all points in $Z_{\tilde{S}}$ that also belong to the hyperplane N_k we know the point with the maximal element $\gamma_{\max}$ in the e_k-direction, then we are certain that *all* points $z \in Z_{\tilde{S}}$ that have a component in the e_k-direction bigger than $\gamma_{\max}$ will have a derivative with a component in the e_k-direction with the same sign as $sign(a_{kk})$ (see Figure 3.8). To be more precise, first we define

$$\gamma_{\max} := \max_{z^-} \left\{ \gamma = \frac{-1}{e_k^T \bar{A} e_k} e_k^T \bar{A} z^- \mid z^- \in \ker(e_k^T), \; z^- + \gamma e_k \in Z_{\tilde{S}} \cap N_k \right\}.$$

PROPOSITION 3.4.1 *$\forall z := z^- + \gamma e_k \in Z_{\tilde{S}}$, $z^- \in \ker(e_k^T)$, with $\gamma > \gamma_{\max}$ it holds that $sign(\dot{x}^k) = sign(a_{kk})$.*

Proof. For any z satisfying the assumption, choose δ such that $z^* = z^- + (\gamma - \delta)e_k \in N_k$. Since $\gamma > \gamma_{\max}$ it follows that $\delta > 0$. Now with $z = \underbrace{z^- + (\gamma - \delta)e_k}_{z^*} + \delta e_k$ it follows from (3.13) that $sign(\dot{x}^k) = sign(a_{kk})$. ■

PROPOSITION 3.4.2 *$\exists z := z^- + \gamma e_k \in Z_{\tilde{S}}$, $z^- \in \ker(e_k^T)$, with $\gamma < \gamma_{\max}$, for which it holds that $sign(\dot{x}^k) = -sign(a_{kk})$.*

Proof. Choose $z = z^- + \gamma e_k \in Z_{\tilde{S}}$ such that there exists a δ for which $z^* = z^- + (\gamma - \delta)e_k \in N_k$ and $(\gamma - \delta) = \gamma_{\max}$. Then, since $\gamma < \gamma_{\max}$ it must hold that $\delta < 0$ and from (3.13) it follows that for $z = \underbrace{z^- + (\gamma - \delta)e_k}_{z^*} + \delta e_k$ we have that $sign(\dot{x}^k) = -sign(a_{kk})$. ■

The first proposition states that "above" the elements in the plane $\Gamma_{\max} = \{z \in Z_{\tilde{S}} \mid z^k = \gamma_{\max}\}$ all derivatives of the k-th component of x have the same sign and the second propositions states that "below" the plane $\Gamma_{\max}$ there exists at least one point for which the derivative of the k-th component of x has the opposite sign (see Figure 3.8). With this information and the knowledge to which hypercube the elements in $\Gamma_{\max}$ belong, it is possible to decide for all discrete states in $\tilde{S}$, labelled by $(a^1, ..., a^k, ..., a^n)$, if a transition to $(a^1, ..., a^k + q, ..., a^n)$ is possible, where $q = -sign(a_{kk})$. Following the same reasoning, it is possible to compute $\gamma_{\min}$ and the corresponding plane $\Gamma_{\min}$ from which the discrete states that allow a transition in the opposite direction follow. In conclusion, by computing $\gamma_{\max}$ and $\gamma_{\min}$ all transitions of the discrete states in $\tilde{S}$ from $(a^1, ..., a^k, ..., a^n)$ to $(a^1, ..., a^k \pm 1, ..., a^n)$ can be determined at once. Since we are dealing with linear systems, the computation of $\gamma_{\max}$ and $\gamma_{\min}$ is straightforward:

$$\gamma_{\max} = \begin{cases} \frac{-1}{a_{kk}} e_k^T(\bar{A}^+ z_{\min} + \bar{A}^- z_{\max}), & a_{kk} > 0 \\ \frac{-1}{a_{kk}} e_k^T(\bar{A}^+ z_{\max} + \bar{A}^- z_{\min}), & a_{kk} < 0 \end{cases}$$

$$\gamma_{\min} = \begin{cases} \frac{-1}{a_{kk}} e_k^T(\bar{A}^+ z_{\max} + \bar{A}^- z_{\min}), & a_{kk} > 0 \\ \frac{-1}{a_{kk}} e_k^T(\bar{A}^+ z_{\min} + \bar{A}^- z_{\max}), & a_{kk} < 0 \end{cases}$$

where the entries of $\bar{A}^+$ and $\bar{A}^-$ defined by

$$\bar{a}_{ij}^+ = \begin{cases} a_{ij} & a_{ij} > 0 \\ 0 & \text{else} \end{cases} \quad \text{and} \quad \bar{a}_{ij}^- = \begin{cases} a_{ij} & a_{ij} < 0 \\ 0 & \text{else.} \end{cases}$$

and

$$z^i_{\max} = \begin{cases} \max\{z_i \mid z \in Z_{\tilde{S}}\} & i \neq k \\ 0 & i = k \end{cases}$$

$$z^i_{\min} = \begin{cases} \min\{z_i \mid z \in Z_{\tilde{S}}\} & i \neq k \\ 0 & i = k \end{cases}$$

In words: for computing $\gamma_{\max}$ in case $a_{kk} < 0$ (and consequently $\frac{-1}{a_{kk}} > 0$), all elements of $\bar{A}$ that have a positive influence on the maximum (*i.e.* $\bar{A}^+$) should contribute maximally (hence $z_{\max}$) whereas all elements that have a negative influence ($\bar{A}^-$) should contribute minimally (by $z_{\min}$).

For the case that $a_{kk} = 0$, it is not possible to compute $\gamma_{\max}$, and $\gamma_{\min}$. In this case the hyperplane N_k is parallel to the e_k-direction. Therefore it is computed for each set $Z_{\tilde{S}}$ if it is intersected by N_k. If so, then the discrete states in $\tilde{S}$ allow transitions from discrete states labelled by $(a^1, ..., a^k, ..., a^n)$ to both $(a^1, ..., a^k{+}1, ..., a^n)$ and $(a^1, ..., a^k{-}1, ..., a^n)$. If not, then only transitions to $(a^1, ..., a^k + q, ..., a^n)$ are possible, where $q \in \{-1, 1\}$ can be determined by computing $sign(\dot{x}^k)$ in one point $z \in Z_{\tilde{S}}$; for all other points in $Z_{\tilde{S}}$ the $sign(\dot{x}^k)$ will be the same. It is easily verified whether $Z_{\tilde{S}}$ is intersected by N_k by computing the intersection of N_k with the line $l := z_{\min} + \lambda(z_{\max} - z_{\min})$, $\lambda \in \mathbb{R}$, as will be explained. The parameter λ^* that corresponds to the intersection point can be computed readily. Indeed from $e_k^T \bar{A} z_{\min} + \lambda e_k^T \bar{A}(z_{\max} - z_{\min}) = 0$ it follows that

$$\lambda^* = \frac{-e_k^T \bar{A} z_{\min}}{e_k^T \bar{A}(z_{\max} - z_{\min})}.$$

If it holds that $0 < \lambda^* < 1$, that is if the intersection point of N_k and l is a convex combination of $z_{\min}$ and $z_{\max}$, then N_k intersects the interior of $Z_{\tilde{S}}$.

To determine the number of computations necessary for the construction of a discrete-event model for a linear system, it can be seen from the procedure described here, that for coordinate i in one step for $\tilde{n}_i$ discrete states the possible transitions are determined. For each coordinate this has to be done $\prod_{j \neq i} n_j$ times. Furthermore, this computation is performed for all discrete inputs, *i.e.* $\prod_{k=1}^{m} m_k$ times. With this, the total number of computations for a linear system becomes:

$$NC = \sum_{i=1}^{n} \left((\prod_{j \neq i} n_j)(\prod_{k=1}^{m} m_k) \right). \tag{3.14}$$

Compared with the nonlinear case it is observed that besides a reduction of the number of computations also the nature of the computations is different. For the nonlinear case these are (demanding) optimizations whereas

for the linear system these are only simple algebraic computations. Hence, it is extremely beneficial to exploit the linear structure of the system to be modelled.

3.4.2 Hierarchical structure

We will discuss two ways of using a hierarchical structure for reducing the computational effort.

Local refinement

The first method is effective for the situation where a part of the state space is of particular interest, *e.g.* because a high accuracy is needed there for control purposes. To obtain the high accuracy, a fine partitioning of the state space in the region of particular interest is needed. However, introducing a fine partitioning of this region will be at the cost of introducing unnecessary discrete states, since the boundaries that are used for the partitioning (*i.e.* the refinement) also will partition other regions. To illustrate this, see Figure 3.9 (a). By refining the partitioning of the region in the center (*i.e.* the

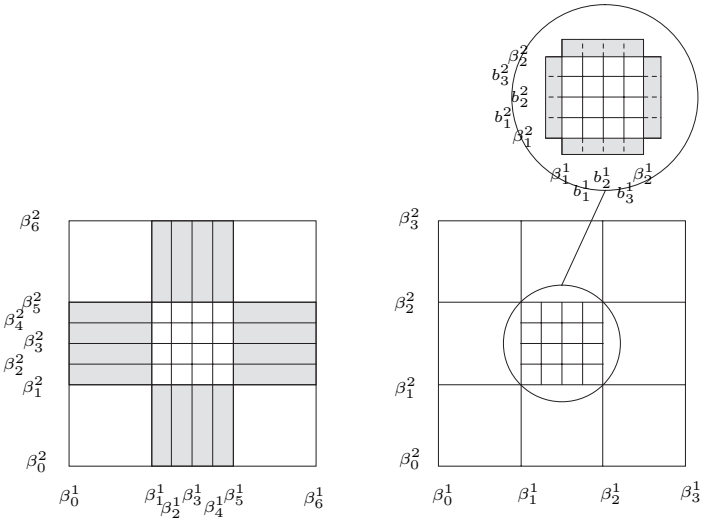

Figure 3.9: *Refinement of the partitioning (a) and using a hierarchical structure with two models (b)*

region of particular interest), besides the 16 desired states in the center, also 16 undesired states are added (in the grey areas) because of the expansion of the number of boundaries.

Instead of using one discrete-event model for the complete state space region, it is convenient if we use two discrete-event models: one (coarse) discrete-event model for the coarse partitioning, and another discrete-event model for the fine partitioning, thus avoiding the introduction of unnecessary discrete states. By using a supervisor which keeps track of the region we are operating in, it is possible to use multiple discrete-event models by switching between them when appropriate, see Figure 3.10.

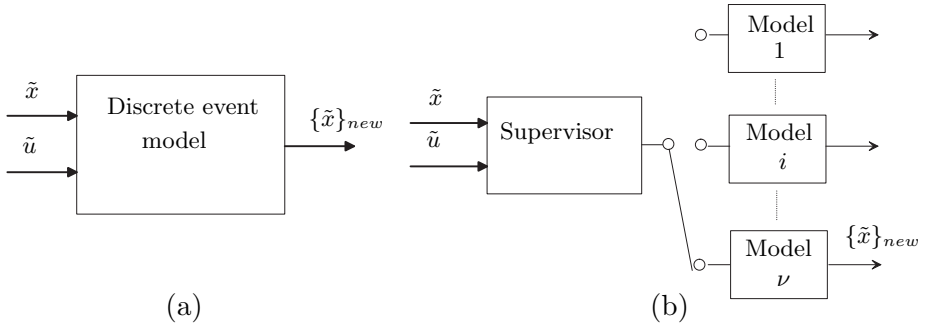

Figure 3.10: *The (a) complete- and (b) hierarchical discrete-event model*

For the partitioning in Figure 3.9 this implies that a coarse discrete-event model (with 9 discrete states) is valid for the off center regions and that for the region in the center a more accurate model is used (with 16 discrete states), see Figure 3.9 (b).

Creating the models for this hierarchical structure is performed in the same manner as for the original system except for some additional book-keeping. The discrete-event model for the coarse partitioning is obtained by applying the procedure described in Section 3.2 with the boundaries defining the coarse partitioning (*e.g.* the boundaries β_0^i, β_1^i, β_2^i, and β_3^i, $i = 1, 2$, in Figure 3.9 (b)). If from this coarse discrete-event model it follows that (given a discrete state $\tilde{x}$) a transition is possible to the discrete state for which the refinement applies, then for the hierarchical model, this means that transitions are possible from $\tilde{x}$ to *all* discrete states of the *accurate* model that are adjacent to $\tilde{x}$. If the accurate discrete-event model is constructed by the method in Section 3.2 with the boundaries defining the fine partitioning (*e.g.* the boundaries β_1^i, b_1^i, b_2^i, b_3^i, and β_2^i, $i = 1, 2$, in Figure 3.9 (b)) then from this model it can not be seen if transitions to discrete states of the coarse model are possible, since these are not states of the accurate model itself. For this reason, additional boundaries are introduced to expand the region of interest of the accurate model, see the grey areas in Figure 3.9 (b). If according to the accurate model with the expanded number of states, a transition is possible from one of the original discrete states to one of the additional discrete states (the grey

states in Figure 3.9 (b)), then for the hierarchical model, this means that a transition is possible to the associated coarse discrete state. Note that the additional discrete states that arise in this manner (the grey states) are not used in the model itself[6], so the accurate model has 16 discrete states instead of 32.

Sparsity

For both nonlinear and linear systems the number of optimizations or computations can be reduced by exploiting the sparsity of the differential equation (3.1) or (3.12). As expressed in (3.11) for our discrete-event modelling algorithm it is assumed that for each of the coordinates x^i all the coordinates of x are of importance for the possibility of a transition (*i.e.* for $\dot{x}^i$). In many cases however, only a part of the state x and the input u influences the derivative $\dot{x}^i$. This makes it profitable to consider sub-systems of the overall system such that the sum of the computations for the individual systems is less than for the overall system.

For each component x^i of the state vector, we compute the sets of indices of the components of the state x and the input u that influence the sign of the derivative $f^i(x,u)$, collected in the sets S_x^i and S_u^i, respectively.

$$S_x^i := \{j \mid \exists x \in \mathbb{R}^n, \exists u \in \mathbb{R}^m \text{ such that } \frac{\partial f^i(x,u)}{\partial x^j} \neq 0\},$$
$$S_u^i := \{j \mid \exists x \in \mathbb{R}^n, \exists u \in \mathbb{R}^m \text{ such that } \frac{\partial f^i(x,u)}{\partial u^j} \neq 0\}.$$

These sets serve as a basis for decomposing the original system into a number of sub-systems. For instance, consider a system without inputs $\dot{x} = f(x)$, $x \in \mathbb{R}^3$, and assume that $S_x^1 = \{1,2\}$, $S_x^2 = \{1\}$, and $S_x^3 = \{2,3\}$. Then a possible decomposition is given by $[\dot{x}^1, \dot{x}^2]^T = f_1([x^1, x^2]^T)$ and $\dot{x}^3 = f_2([x^2, x^3]^T)$. Note that these subsystems are not decoupled, since f_2 still depends on a state in f_1. In general, the state space is partitioned in ν subspaces, such that $\mathbb{R}^n = \mathbb{R}^{n_1} \times ... \times \mathbb{R}^{n_\nu}$, and $\sum_{i=1}^{\nu} n_i = n$. The new state z is a permutation of the original state components and is decomposed as $z = [z_1, ..., z_\nu]^T$. The indices of the components of the original state vector that are elements of the 'sub state' z_j, are collected in the sets X_j, $j = 1, ..., \nu$. So, if $z_1 = [x^1, x^2]^T$ then $X_1 = \{1,2\}$. The partitioning of the state space is such that $X_j \cap X_k = \emptyset$ for $j \neq k$, and $\bigcup_{j=1}^{\nu} X_j = \{1, ..., n\}$.

The differential equations in (3.1) are now partitioned accordingly such that we have ν sub-systems,

$$\dot{z}_i = f_i(w_i, v_i),$$

[6] In fact, transitions between additional states need not to be computed at all, since only transitions from original states to additional states are of interest.

where w_i (v_i) is a vector consisting of those components x^j (u^j) of x (u) that influence $\dot{z}_i$ directly. For each sub-system i these vectors can be characterized by the sets of indices S_w^i (and S_v^i) of the components of the original state-vector x (or input-vector u):

$$S_w^i := \{j \in S_x^k \mid k \in X_i\}, \text{ and } S_v^i := \{j \in S_u^k \mid k \in X_i\}.$$

The computational effort to obtain discrete-event models of all these sub-systems may be significantly less than creating the complete model at once. For each sub-model i the computational effort to produce the discrete-event model is

$$NO_i = \sum_{j \in X_i} \left((n_j - 1)(\prod_{k \in S_w^i \setminus j} n_k)(\prod_{l \in S_v^i} m_l) \right), \tag{3.15}$$

involving $\#(S_v^i)(\#(S_w^i) - 1)$ variables. Note, that for each optimization less (or equal) variables are involved than for the overall system.

The information provided by these sub-models now can be used to reconstruct the complete discrete-event model that would result from the original procedure. For this, each discrete state $\tilde{x}$ and input $\tilde{u}$ is decomposed into $\tilde{w}_i$ and $\tilde{v}_i$ for $i = 1, ..., \nu$. Next, for all the sub-models the next possible states $\{\tilde{z}_i\}_{new}$ are computed given $\tilde{w}_i$ and $\tilde{v}_i$. Given these next new states $\{\tilde{z}_i\}_{new}$ it is easy to reconstruct the set of new states $\{\tilde{x}\}_{new}$ since for example in the tuple presentation, a discrete state $\tilde{x}$ is equal to $(\tilde{z}_1, ..., \tilde{z}_\nu)$ except for a known permutation of elements. This procedure can be performed off-line and from the results a new complete discrete-event model can be built. Since for all the sub-models, the computation of $\{\tilde{z}_i\}_{new}$ can be done in parallel (on different machines) even more computation-time can be gained than only caused by the reduction of the number of computations and variables for each optimization.

By creating a hierarchical structure, it is possible to use the sub-models explicitly instead of building one large model. In this case, a supervisor is used to extract the necessary information for each sub-model and to reconstruct the complete state from the information provided by the sub-models. This requires extracting $\tilde{w}_i$ and $\tilde{v}_i$ from $\tilde{x}$ and $\tilde{u}$ for each of the sub-models, that is for $i = 1, ..., \nu$, as defined by the sets S_w^i and S_v^i, respectively. These parts $\tilde{w}_i, \tilde{v}_i$ provide the information that is necessary for sub-model i to compute (in parallel with the other models) the next possible discrete states $\{\tilde{z}_i\}$, which then are used to reconstruct $\tilde{x}$. Clearly, this is the same procedure as for the construction of the overall discrete-event model. The procedure is depicted in Figure 3.11.

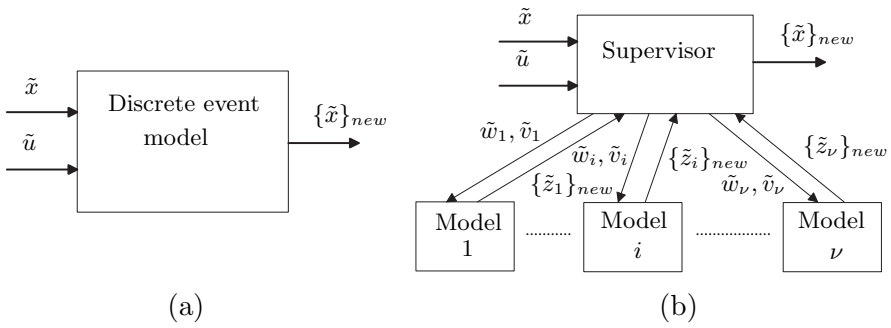

Figure 3.11: *The (a) complete- and (b) hierarchical discrete-event model*

3.5 Example: a three tank system

To clarify the main concepts discussed in the previous sections, a three tank system as in Figure 3.12 serves as an illustration. It consists of three commu-

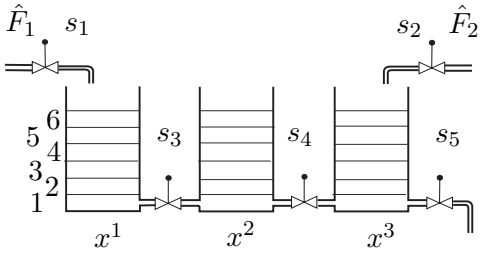

Figure 3.12: *Three tank system (parallel)*

nicating tanks (all with cross section A_t) which are connected through pipes (all with cross-section A_p). The input $u = (s_1, s_2, s_3, s_4, s_5)$ of the system consists of the on/off switches, $s_1, \ldots, s_5$ controlling the valves, where $s_i \in \{0, 1\}$ (closed/open). The first and the last tank can be filled by the flows $\hat{F}_1$ and $\hat{F}_2$ respectively. Only the last tank has a drain. The (continuous) state vector $x = [x^1, x^2, x^3]^T$ is given by the water levels in each tank. Each tank is divided into six parts (discrete states) that are defined by the levels: 0, 0.01, 0.1, 0.2, 0.3, 0.4, and 0.5 $[m]$. It is only observed if a level is reached. This implies that although the dynamics are described by continuous differential equations, only discrete outputs are observed and discrete inputs are applied, so for a controller supervising this system, this tank system acts as a discrete-event system.

The discrete states for the discrete-event model that will be computed are induced by the partitioning of the continuous state space into hypercubes. This partitioning is defined by the boundaries β_j^i following from the level-measurements. By this, the region of interest in the state space is divided

into 216 hypercubes resulting in an equal number of discrete states. Since the input u is given by the position of the switches $s_1, ..., s_5$ it is not necessary to discretize the input space, because the input is already discrete. Each of the five inputs can take two values (0 or 1) resulting in $2^5 = 32$ discrete inputs. The mapping Q is defined as in Section 3.2. Since the input is already discrete, there is no need for a mapping S from the continuous to the discrete domain. The transition function ϕ is computed for a nonlinear and for a linear model of the three tank system. To illustrate the computation effort necessary to obtain these discrete-event models, they are computed without and with using knowledge of the systems structure (sparsity). In conclusion, a discrete-event model of the three-tank system is computed in four different ways. The computations are performed on a Personal Computer with a Pentium III 500 MHz processor and with 128Mb memory.

3.5.1 The nonlinear system

First, the system is modelled by the following set of nonlinear differential equations.

$$\frac{dx^1}{dt} = \frac{1}{A_t}\left(s_1\hat{F}_1 - s_3 A_p\sqrt{2g}\Psi(x^1 - x^2)\right), \tag{3.16a}$$

$$\frac{dx^2}{dt} = \frac{1}{A_t}\left(s_3 A_p\sqrt{2g}\Psi(x^1 - x^2) - s_4 A_p\sqrt{2g}\Psi(x^2 - x^3)\right), \tag{3.16b}$$

$$\frac{dx^3}{dt} = \frac{1}{A_t}\left(s_2\hat{F}_2 + s_4 A_p\sqrt{2g}\Psi(x^2 - x^3) - s_5 A_p\sqrt{2g}\Psi(x^3)\right), \tag{3.16c}$$

where $\Psi(x) := sign(x)\sqrt{|x|}$.

The differential equations describing the system have three coordinates ($n = 3$) and for our discretization each coordinate is partitioned into 6 regions ($\tilde{n}_i = 6$). Furthermore there are 5 inputs ($m = 5$) and each input can take two values ($\tilde{m}_i = 2$). From (3.11) it follows that the number of optimizations is approximately $NO_{nl} = \sum_{i=1}^{3}((6-1)(\prod_{j\neq i} 6)(\prod_{k=1}^{5} 2)) = 12480$. The time to perform these optimizations and to compute the discrete-event model of the complete system is 522.17 [s] (8 minutes and 42 seconds).

With the resulting discrete-event model we can compute the next possible discrete state(s) given an initial discrete state and a discrete input. In Figure 3.13 (a) a continuous trajectory is depicted starting form initial state $x_0 = [0.25, 0.25, 0.25]^T$ and simulated for 250 [s]. The input applied to the system only changes if a boundary (level sensor) is reached. This results in the input

v

$$v_{[0,250]} = \begin{cases} (0,0,0,0,1), & 0 \leq t < 33.2 \\ (0,0,0,1,1), & 33.2 \leq t < 56.2 \\ (0,0,1,1,1), & 56.2 \leq t < 143.8 \\ (1,1,1,1,1), & 143.8 \leq t \leq 250. \end{cases}$$

The corresponding discretized (*i.e.* observed by the level sensors) trajectory is shown in Figure 3.13 (b) and is given by the sequence

$$\begin{aligned} &(4,4,4) \overset{(00001)}{\longrightarrow} (4,4,3) \overset{(00001)}{\longrightarrow} (4,4,2) \overset{(00011)}{\longrightarrow} (4,4,3) \overset{(00011)}{\longrightarrow} \\ &(4,3,3) \overset{(00011)}{\longrightarrow} (4,3,2) \overset{(00111)}{\longrightarrow} (3,3,2) \overset{(00111)}{\longrightarrow} (3,2,2) \overset{(00111)}{\longrightarrow} \\ &(2,2,2) \overset{(11111)}{\longrightarrow} (3,2,2) \overset{(11111)}{\longrightarrow} (3,3,2) \overset{(11111)}{\longrightarrow} (3,3,3) \overset{(11111)}{\longrightarrow} (4,3,3). \end{aligned}$$

The set of discrete states resulting from our discrete-event model with initial discrete state $(4,4,4)$ and the same input sequence as for the continuous case is given in Figure 3.13 (c). Due to the nondeterminism of the discrete-event model the set of discrete states that can be reached (15 states) is more than the original 9 states that are actually reached. Finally, in Figure 3.13 (d) a discrete trajectory is shown that cannot result from the differential equations (3.16) describing the three tank system but that is possible according a sequence of discrete states resulting from our discrete-event model. This sequence is given by

$$\begin{aligned} &(4,4,4) \overset{(00001)}{\longrightarrow} (4,4,3) \overset{(00001)}{\longrightarrow} (4,4,2) \overset{(00011)}{\longrightarrow} (4,4,3) \overset{(00011)}{\longrightarrow} \\ &(4,3,3) \overset{(00011)}{\longrightarrow} (4,3,2) \overset{(00111)}{\longrightarrow} (3,3,2) \overset{(00111)}{\longrightarrow} (3,2,2) \overset{(00111)}{\longrightarrow} \\ &(3,3,2) \overset{(11111)}{\longrightarrow} (3,3,3) \overset{(11111)}{\longrightarrow} (4,3,3) \overset{(11111)}{\longrightarrow} (4,4,3) \overset{(11111)}{\longrightarrow} (5,4,3). \end{aligned}$$

Clearly, for input $u = (0,0,1,1,1)$, *i.e.* only s_1 and s_2 are closed, the level in all tanks has to drop monotonically. Hence the sequence $(3,3,2) \overset{(00111)}{\longrightarrow} (3,2,2) \overset{(00111)}{\longrightarrow} (3,3,2)$ is a spurious solution of our discrete-event model.

Next we exploit the structure of the system. From the differential equations (3.16) it can be seen that a single tank is not influenced by all the inputs or the level of the fluid in all other tanks. In fact, instead of considering the three tank system as one system it is also possible to look at the tanks separately, see Figure 3.14.

By this the original state space is partitioned in 3 sub-spaces and for the new coordinates $z_1 = x^1$, $z_2 = x^2$, and $z_3 = x^3$ we consider the differential

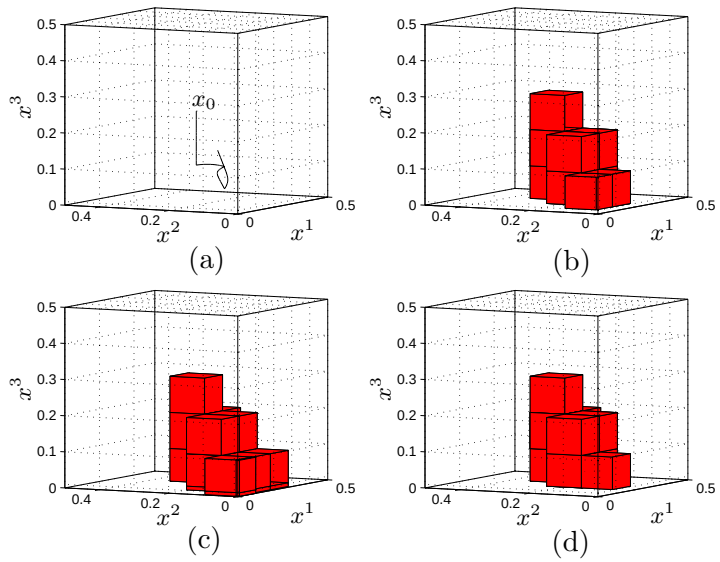

Figure 3.13: *Continuous (a) and discretized (b) simulation result together with discrete simulation (c) and a spurious discrete trajectory (d)*

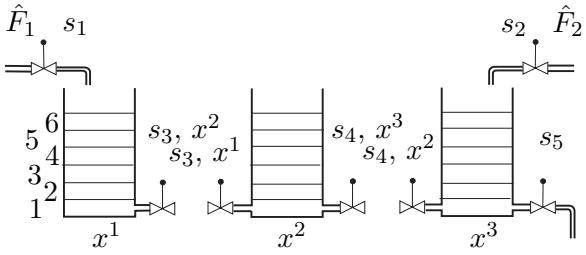

Figure 3.14: *Considering the tanks separately*

equations of the form

$$\begin{aligned}
\dot{z}_1 &= \frac{1}{A_t}\left(s_1\hat{F}_1 - s_3 A_p\sqrt{2g}\Psi(x^1 - x^2)\right) = f_1(x^1, x^2, u^1, u^3) = f_1(w_1, v_1)\\
\dot{z}_2 &= \frac{1}{A_t}\left(s_3 A_p\sqrt{2g}\Psi(x^1 - x^2) - s_4 A_p\sqrt{2g}\Psi(x^2 - x^3)\right)\\
&= f_2(x^1, x^2, x^3, u^3, u^4) = f_2(w_2, v_2)\\
\dot{z}_3 &= \frac{1}{A_t}\left(s_2\hat{F}_2 + s_4 A_p\sqrt{2g}\Psi(x^2 - x^3) - s_5 A_p\sqrt{2g}\Psi x^3\right)\\
&= f_3(x^2, x^3, u^2, u^4, u^5) = f_3(w_3, v_3).
\end{aligned}$$

Using the notation as in Subsection 3.4.2 it is clear that $X_1 = \{1\}$, $X_2 = \{2\}$ and $X_3 = \{3\}$. Since the first tank is only influenced by x^1, x^2, u^1, and u^3,

the sets $S_w^1 = \{1, 2\}$ and $S_v^1 = \{1, 3\}$ result. Similarly we obtain $S_w^2 = \{1, 2, 3\}$, $S_v^2 = \{3, 4\}$, and $S_w^3 = \{2, 3\}$, $S_v^3 = \{2, 4, 5\}$.

If we use this to compute the number of optimizations for each of the systems by (3.15), then we obtain the following results.

$$
\begin{aligned}
NO_{nl,1} &= (6-1)(6)(2 \cdot 2) = 120, \\
NO_{nl,2} &= (6-1)(6 \cdot 6)(2 \cdot 2) = 720, \\
NO_{nl,3} &= (6-1)(6)(2 \cdot 2 \cdot 2) = 240.
\end{aligned}
$$

In conclusion, we need a total of $120 + 720 + 240 = 1080$ optimizations to obtain the complete discrete-event model of the three tank system. The time needed to perform these computations is 2.97 [s] for the first, 26.86 [s] for the second, and 5.55 [s] for the third model. From the number of optimizations it follows that the second system needs 6 times more optimizations than the first system and for the third system 2 times more optimizations are required. From the computation-times we can see that the second system required almost 9 times more computation-time than the first system and the third system needs 1.7 times more time to compute the discrete-event model. The difference of the ratios of computation-times compared with the ratios of the number of optimizations is probably due to the fact that besides the number of optimizations also the number of variables of the optimizations changes. For the second system there are two variables for the optimization (x^1 and x^2), whereas for the first and the third system only one variable (x^2) is involved. Hence the ratios of the computation-times are smaller than the ratios of the number of computations.

The total time to compute the overall model by the hierarchical method is equal to 35.38 [s]. This is $\frac{1}{14.8}$ of the time needed by the previous procedure. The ratio of the number of optimizations of both methods is $\frac{12380}{1080} = 11.5$

3.5.2 The linear system

By assuming that the flow between tanks depends linearly on the difference between the levels instead of the square root of the difference (by Bernoulli's equation) we obtain the following differential equations

$$
\begin{aligned}
\frac{dx^1}{dt} &= \frac{1}{A_t}\left(s_1 \hat{F}_1 - s_3 A_p \beta (x^1 - x^2)\right), \\
\frac{dx^2}{dt} &= \frac{1}{A_t}\left(s_3 A_p \beta (x^1 - x^2) - s_4 A_p \beta (x^2 - x^3)\right), \\
\frac{dx^3}{dt} &= \frac{1}{A_t}\left(s_2 \hat{F}_2 + s_4 A_p \beta (x^2 - x^3) - s_5 A_p \beta x^3\right).
\end{aligned}
$$

Since inputs (switches s_i) are multiplied with states, this is still a nonlinear system. However, because the switches can only take a finite number of values (1 and 0) it is still possible to use the algorithm for linear systems for computing the discrete-event model of this system by fixing the input for each of the computations. This simply means that for each input we consider a different linear model.

Using equation (3.14) it follows that the number of computations for the complete three tank system is $NC = \sum_{i=1}^{3}((6 \cdot 6)(\prod_{k=1}^{5} 2)) = 3456$. The discrete-event model is computed in 40.81 [s].

By exploiting the sparsity of the system as we have seen for the nonlinear case, the number of computations can be reduced even more. For the first tank $(6)(2 \cdot 2) = 24$ computations are necessary. The second tank requires $(6 \cdot 6)(2 \cdot 2) = 144$ computations and for the third tank $(6)(2 \cdot 2 \cdot 2) = 48$ computations are needed which brings the total amount of computations at 216. Computing the discrete-event models for the three systems separately required 0.16, 1.87, and 0.27 [s] respectively. The total computation-time then is 2.30 [s]. This is $\frac{1}{17.7}$ of the computation-time for the complete system at once, whereas the ratio of the number of computations is $\frac{3456}{216} = 16$.

To summarize the results, the following table is presented

		Computations		computation-time [s]	
Nonlinear			12480		522.17
Nonlinear	1	120	1080	2.97	35.38
	2	720		26.86	
	3	240		5.55	
Linear			3456		40.81
Linear	1	24	216	0.16	2.30
	2	144		1.87	
	3	48		0.27	

3.6 Notes and references

Discrete-event models of continuous systems

Approximating the behavior of continuous plants by a finite automaton has already been described in 1970 (Vinogradov 1970), and has been further developed in (Kornoushenko 1975). However (Kornoushenko 1975, Vinogradov 1970) require an explicit solution of the differential equation describing the plant. This state-equation is used to obtain a discrete-time approximation, which in turn is used for the construction of a deterministic automaton. With the growing interest in hybrid systems (Grossman *et al.* 1993, Antsaklis *et al.* 1993, Alur *et al.* 1996, Antsaklis *et al.* 1997, Antsaklis *et al.* 1999) the qual-

itative modelling approach has gained renewed attention as a way to combine discrete with continuous dynamics. In (Puri *et al.* 1996) a method is discussed for computing an arbitrary close approximation of the set of states reached after time t starting from an initial set, for a differential inclusion $\dot{x} \in f(x)$ satisfying a Lipschitz condition. Various formalisms of discrete-event models of continuous systems are given in literature. In (Stiver and Antsaklis 1993) the discrete states are defined by hypersurfaces and the 'interface' between both domains is given by a static map. Transitions are computed by a derivative test similar to the one described in this chapter, but no explicit (automated) algorithm is given. For linear discrete-time systems a method is discussed in (Lunze 1994, Lunze 1996) for obtaining qualitative models. Discrete states are defined by rectangles and the discrete model is represented by stochastic automata. Also a condition for the existence of a deterministic model is given. In (Lunze *et al.* 1997) the continuous dynamics has been transformed into a Petri-net using discrete states based on rectangular sets and also the continuous dynamics (defined for each rectangular set) has been used to obtained additional (time) information. Another approach is followed in (Lunze *et al.* 1999*b*, Lunze *et al.* 1999*a*), where a continuous-time model is first transformed into a discrete-time model. Next, by a procedure based on backward integrating, a stochastic automaton is obtained with discrete states not necessarily based on rectangular sets. Results are obtained for linear systems without inputs. In (Lunze 1999) timed discrete-event abstraction of continuous systems is proposed modelled by a semi-Markov process. Two discretization methods are described in (Stursberg *et al.* 1997). Both methods use discrete states based on rectangular sets and result in a timed automaton. One method checks the derivatives in gridpoints and the other method uses forward/backward integration. Completeness of the model is the main problem for the latter method. In (Raisch and O'Young 1998) it is explained how to obtain a discrete-event model from discrete-time systems by solving linear inequalities based on the system's inverse. The discrete model then uses a recorded string of measurement and control symbols to compute the next possible states. By this, the nondeterminism is reduced. The proposed modelling method is only effective for affine models. The discretization method discussed in this chapter is based on (Preisig 1996*b*), further improved and extended in (Preisig *et al.* 1997, Philips *et al.* 1997). Exploiting sparsity of the continuous system for reducing the computational effort is discussed in (Philips and Weiss 2000). Methods for reducing the number of discrete states when constructing a discrete-event model of a plant are presented in (Preisig 1989, Preisig 1996*a*). The use of a hierarchical structure is related to multigrid methods in numerical analysis (local refinement) and to hierarchical discrete-event systems (Raisch and Itigin 2000).

Isomorphism

The definition of isomorphism of two discrete-event models (Definition 3.1.5) is based on the definition of two automata being isomorphic (Booth 1967) with the difference that we also allow different discrete-input sets and that we compare (transformed) discrete states instead of discrete outputs only.

Mappings Q, R, and S

In the literature, the mappings from the continuous to the discrete domains always are defined to be static. Although the concept and the intention of these mappings is clear, We believe that these static mappings cannot assign a discrete state for all continuous states; at least not in a proper way. For instance, in the three tank example a trajectory can evolve on a boundary plane in the state space. The mappings used in literature will assign no discrete states when open (rectangular) sets are used; multiple discrete states when closed (rectangular) sets are used or they will change from one discrete state to another when semi open (rectangular) sets are used, because the boundary of such set is reached without being crossed.

Systems with outputs

Most systems that are studied in the literature mentioned above are systems without outputs or systems for which the output are components of the state-vector. For simplicity, for the remainder of this thesis systems without outputs are considered. Only in the next chapter, we will describe methods for reconstructing the complete discrete and continuous state for systems with outputs.

Computational effort

It is understood that the number of optimizations NO and the number of computations NC as mentioned in this chapter only serve as a measure for the computational effort, and are not the exact number of optimizations and certainly not the exact number of computations. They serve as a measure for computational effort since they represent the number of times that a sequence of computations (or a procedure) has to be performed.

Chapter 4

State reconstruction

This chapter deals with the problem of state reconstruction. Two types of states are considered: continuous and discrete states. First it is shown how to reconstruct the continuous state of a continuous system from discrete partial measurement data. Next, we will discuss how to reconstruct the unknown (initial) discrete state from a sequence of discrete measurements (events). Both methods are illustrated by examples.

4.1 Continuous-state reconstruction

The partitioning of the state space on which the construction of the discrete-event models is based, is often induced by the placement of 'discrete sensors'. Discrete sensors are only able to detect when a process variable crosses a certain value and occur frequently in industrial practice. For instance, a motor encoder can be seen as a discrete sensor, since the (angular) position measurements of the motor axis become available only when specific positions are reached. As a consequence, the measurement instants are asynchronous in time (Heemels *et al.* 1999). Also, one could think of an automated bus that has to determine its position and speed based on markers (*e.g.* magnets) placed on fixed positions in the road (De Bruin and van den Bosch 1998). Characteristic for these discrete measurements is that the time-instants at which the information becomes available are not equidistant in time. It is clear that a system with discrete sensors provides poorer information on the time-evolution of the state-variables of a plant than a usual continuous sensor. On the other hand, discrete sensors can be far more reliable and robust in operation, or yield a higher resolution. They may also operate on larger ranges than continuous sensors normally would. All these advantages recommend them for the safety tasks for which they are often used. Another typical application field is the implementation of relay control schemes.

It is studied here how to deal with these discretely-observed systems by using a discrete-event model of the original continuous system. For control purposes, it is interesting to consider another approach of dealing with this kind of systems: reconstructing the continuous state from the discrete measurements allows the application of conventional control strategies. Also complete state information can be directly used to enhance the performance of a supervisory control scheme. Another application lies in the area of fault detection.

In real life, discrete sensors are not necessarily used exclusively, but in combination with continuous sensors that provide measurement data throughout the time evolution of the plant. However, in this section we shall consider the situation where only discrete sensors are present. A particular time-evolution of the plant will be recorded as a sequence of state level crossings. The available measurement data consists of the values that have been crossed and the time when the crossings took place.

To formalize the problem, suppose that we have a process that is described by a model of the form

$$\dot{x}(t) = f(x(t), u(t), t), \qquad x(t_0) = x_0 \tag{4.1}$$

and $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$. Furthermore suppose that we have a number of sensors that detect whether a certain state variable x^i has reached a certain value β^i_j. That means that, for each coordinate x^i, a set of values

$$\mathcal{B}^i = \{\beta^i_1, ..., \beta^i_{n_i-1}\}, \qquad \beta^i_1 < ... < \beta^i_{n_i-1} \tag{4.2}$$

is given such that, the situation $x^i(t) = \beta^i_j$ results in the emission of a sensor signal and the recording of the time when the state event occurs. Note that $\mathcal{B}^i$ differs from the set of boundaries (3.3) because we assume that the outer boundaries defining the region of interest (*i.e.* β^i_0 and $\beta^i_{n_i}$) are not measurable. Furthermore, we assume that the input υ applied to the system is known.

A particular trajectory of the system will be observed as a sequence of k time moments $t_0 < t_1 < ... < t_k$, a sequence of state coordinates $i_1, ..., i_k$ ($i_j \in \{1, ..., n\}$) and a sequence of values $b_1, ..., b_k$ satisfying

$$b_j \in \mathcal{B}^{i_j} \tag{4.3}$$

corresponding to the levels reached by the state variable x^{i_j} of the model at the time moments t_j. Our problem can now be formulated as follows.

Problem 4.1.1 (The continuous-state reconstruction problem) *Let $\mathcal{T} = \{t_1, ..., t_k\}$, $\mathcal{I} = \{i_1, ..., i_k\}$, $\mathcal{V} = \{b_1, ..., b_k\}$ satisfying $i_j \in \{1, ..., n\}$ and $b_j \in \mathcal{B}^{i_j}$ for $j = 1, ..., k$ be given. Find all state trajectories of (4.1) with known input υ such that*

$$x^{i_j}(t_j) = b_j, \quad j = 1, ..., k. \tag{4.4}$$

It is assumed that the model of the time-continuous system 4.1 is exactly known. One can regard this problem as a multipoint boundary-value problem for the differential equation (4.1) in the sense of (Meyer 1973). As we put no constraints on the data, it is quite clear that the problem might have no, several, or an infinite number of solutions. The existence and uniqueness for the problem that we have formulated will receive attention in this section.

The setting presented so far is quite general. It includes the case that some state variables are not monitored at all ($\mathcal{B}^i = \varnothing$ and $n_i = 1$).

Assuming that the differential equation is such that the initial value problem has a unique solution, it is obvious that finding the state trajectory of (4.1) satisfying (4.4) is equivalent to determining the initial condition $x_0 = x(t_0)$. This is particularly convenient whenever an analytical expression for the initial-value solution is available. Therefore, we first examine a large class of systems for which this is true: linear time-invariant systems.

4.1.1 The linear, time-invariant case

We concentrate in this subsection on the case that the process is described by

$$\dot{x}(t) = Ax(t) + Bu(t). \tag{4.5}$$

Then, the initial-value solution with $x(t_0) = x_0$ and input $u(t)$ can be expressed analytically by the variation-of-constants formula

$$x(t) = e^{A(t-t_0)}x_0 + \int_{t_0}^{t} e^{A(t-\tau)}Bu\,(\tau)\,d\tau. \tag{4.6}$$

The conditions (4.4) become

$$b_j = x^{i_j}(t_j) = e_{i_j}^T \left(e^{A(t_j-t_0)}\ x_0 + \int_{t_0}^{t_j} e^{A(t_j-\tau)}Bu(\tau)d\tau \right),$$

or

$$b_j - e_{i_j}^T \int_{t_0}^{t_j} e^{A(t_j-\tau)}\ Bu\,(\tau)\,d\tau = e_{i_j}^T\, e^{A(t_j-t_0)}x_0, \tag{4.7}$$

with $j = 1, ..., k$. This is a system of k linear equations in x_0. Introducing

$$P := \begin{bmatrix} e_{i_1}^T\ e^{A(t_1-t_0)} \\ \vdots \\ e_{i_k}^T\ e^{A(t_k-t_0)} \end{bmatrix} \in \mathbb{R}^{k\times n} \tag{4.8}$$

and

$$d := \begin{bmatrix} b_1 \ - e_{i_1}^T \int_{t_0}^{t_1} e^{A(t_1-\tau)} \ Bu\,(\tau)\,d\tau \\ \vdots \\ b_k \ - e_{i_k}^T \int_{t_0}^{t_k} e^{A(t_k-\tau)} Bu\,(\tau)\,d\tau \end{bmatrix} \in \mathbb{R}^k \tag{4.9}$$

the system (4.7) can be written in the compact form

$$Px_0 = d. \tag{4.10}$$

Note, that both P and d are depending on the initial condition $x\,(t_0) = x_0$ and the input $u(t)$ because these determine the time-instants t_j and the indices i_j of the states that cross certain levels. Since both the matrix P and the vector d can be readily computed from the available data, the state reconstruction problem is in principle solvable for this case. Indeed, the problem has a solution if and only if

$$d \in \operatorname{Im} P.$$

If our model is correct, that is if the data is generated by the system (4.1) then this condition must be guaranteed. In particular, if P is surjective, *i.e.* if it has full row rank, the solution exists for every d. On the other hand, the problem has at most one solution if and only if P is injective, *i.e.* if it has full column rank. In general, whenever a solution exists, the set of all solutions can be parameterized by the formula

$$x_0 = P^{\#}d + P_0 w, \tag{4.11}$$

where $P^{\#}$ is a left inverse of P, P_0 is a matrix such that $\operatorname{Ker} P = \operatorname{Im} P_0$ and w is an arbitrary vector parameter. Of course, if $k = n$ and P is invertible, the solution to (4.10) exists and is unique. Otherwise, x_0 is known to be an element of the affine subspace of $\mathbb{R}^n$ through $x_0^* := P^{\#}d$ and parallel to $\ker P$.

4.1.2 The linear, time-varying case

Let us consider now the case that our model is linear time-varying, *i.e.* of the form

$$\dot{x}(t) = A(t)x(t) + B(t)u(t). \tag{4.12}$$

The solution for this differential equation can be written again as

$$x(t) = \Phi(t, t_0)x_0 + \int_{t_0}^{t} \Phi(t, \tau)B(\tau)u(\tau)d\tau,$$

where the transition matrix $\Phi(t, t_0)$ satisfies

$$\dot{\Phi}(t, t_0) = A(t)\Phi(t, t_0), \qquad \Phi(t_0, t_0) = I.$$

In general there is no analytic expression for the transition matrix. However, we can follow the derivation in the previous section in order to obtain at least a numerical procedure to assess reconstructibility and, eventually, to solve for the initial state.

The conditions (4.4) can be written as

$$b_j = x^{i_j}(t_j) = e_{i_j}^T \, [\, \Phi(t_j, t_0)x_0 + \int_{t_0}^{t_j} \Phi(t_j, \tau) \; Bu(\tau)d\tau],$$

or

$$b_j \; - e_{i_j}^T \int_{t_0}^{t_j} \Phi(t_j, \tau) \; Bu\,(\tau)\, d\tau = e_{i_j}^T \; \Phi(t_j, t_0) \; x_0, \tag{4.13}$$

with $j = 1, ..., k$. Introducing

$$P := \begin{bmatrix} e_{i_1}^T & \Phi(t_1, t_0) \\ & \vdots \\ e_{i_k}^T & \Phi(t_k, t_0) \end{bmatrix} \in \mathbb{R}^{k \times n} \tag{4.14}$$

and

$$d := \begin{bmatrix} b_1 \; - e_{i_1}^T \int_{t_0}^{t_1} \; \Phi(t_1, \tau) \;\; Bu\,(\tau)\, d\tau \\ \vdots \\ b_k \; - e_{i_k}^T \int_{t_0}^{t_k} \; \Phi(t_k, \tau) \; Bu\,(\tau)\, d\tau \end{bmatrix} \in \mathbb{R}^k \tag{4.15}$$

the system (4.13) can be written in a compact form as in (4.10). Although there is no general analytic formula for the transition matrix, computing the matrix P and the vector d from the available data amounts to several numerical integrations of the system (4.12), which is for reasonably complex models not a very challenging task.

4.1.3 The general case

Not much is known about the general problem. The theory of multipoint boundary-value problems offers some necessary conditions for the existence of solutions to the state reconstruction problem. For example, (Meyer 1973, Section 4.1) gives two such existence results for the case that the number of recorded events k equals the state space dimension n. The first result,

(Meyer 1973, Theorem 4.1.1), reduces the problem to the existence of a Cauchy problem for a partial differential equation. This leads to a numerical solution for the multipoint boundary value problem by the method of invariant embedding. The second existence result (Meyer 1973, Theorem 4.1.2) states that the solution exists provided that the length of the intervals between two consecutive results is small enough. Quantitative estimates are provided. It is assumed however that information is available about all state variables which is not always the case in our setting, neither it is necessary.

There are at least two other solution methods concerning the numerical solution of the state reconstruction problem besides these invariant embedding methods: the finite difference methods and the shooting methods. See (Roberts and Shipman 1972) and (Keller 1968). For testing purposes we have implemented a variant of Newton's method as one of the shooting methods ((Roberts and Shipman 1972, Chapter 6)), which now will be explained briefly.

Introducing the notation $x(t; t_0, x_0)$ for the solution of (4.1) given input v and initial condition $x(t_0) = x_0$, we try to find the initial value x_0 (in principle there might be more than one minimizer) that minimizes

$$J(x_0) = \sum_{j=1}^{k} \left\| x^{i_j}\left(t_j; t_0, x_0\right) - b_j \right\|^2 .$$

By defining $x(t_{1,...,}t_k, t_0, x_0) := \left[x^{i_1}\left(t_1; t_0, x_0\right), ..., x^{i_k}\left(t_k; t_0, x_0\right)\right]^T$ and $b := [b_1, ..., b_k]^T$ the objective function $J(x_0)$ can be written as

$$J(x_0) = [x(t_{1,...,}t_k, t_0, x_0) - b]^T [x(t_{1,...,}t_k, t_0, x_0) - b].$$

Let x_0^* be an initial state for which $J(x_0^*) \leq J(x_0)$, for $x_0 \neq x_0^*$. In case that the state reconstruction problem is uniquely solvable, this is the state we are looking for. Furthermore, let $\hat{x}_0$ be the current estimate of the optimal initial condition then we can write $x_0^* = \hat{x}_0 + \Delta x$. By taking the Taylor expansion of $x(t_{1,...,}t_k, t_0, \hat{x}_0 + \Delta x)$ and neglecting higher order terms we obtain

$$x(t_{1,...,}t_k, t_0, \hat{x}_0 + \Delta x) \approx x(t_{1,...,}t_k, t_0, \hat{x}_0) + \frac{\partial x(t_1, ..., t_k; t_0, \hat{x}_0)}{\partial x_0} \Delta x.$$

This yields

$$J(x_0) \approx [P\Delta x - d]^T [P\Delta x - d],$$

with $P := \frac{\partial x(t_1,...,t_k;t_0,\hat{x}_0)}{\partial x_0}$ and $d := b - x(t_{1,...,}t_k, t_0, \hat{x}_0)$. Since for the optimal solution x_0^* it must hold that $\frac{\partial J(x_0)}{\partial x_0} = 0$, we obtain that

$$\frac{\partial J(x_0)}{\partial x_0} \approx 2P^T P\Delta x - 2P^T d = 0,$$

from which we get that $\Delta x \approx (P^T P)^{-1} P d$, provided that $P^T P$ is invertible (*i.e.* P is injective). For linear time-invariant systems taking initial guess $\hat{x}_0 = 0$ will immediately lead to the optimal solution in one step and is equal to (4.11). Note that $(P^T P)^{-1} P$ is a left inverse of P. For nonlinear systems the new estimate is obtained by the recursion scheme

$$\hat{x}_0^{new} := \hat{x}_0 + (P^T P)^{-1} P^T d,$$

The elements of the matrix P can be conveniently obtained by integrating the variational system corresponding to (4.1)

$$\frac{d\Phi}{dt} = \frac{\partial f}{\partial x}\left(x\left(t; t_0, \hat{x}_0\right)\right)\Phi, \qquad \Phi\left(t_0\right) = I.$$

In this way, (4.1) is linearized in each point of the (estimated) trajectory $x\left(t; t_0, \hat{x}_0\right)$ such that the linear time-varying system (4.12) results and P can be computed similarly as in (4.14). Actually, it is proven in (Perko 1991, p. 83) that indeed

$$P = \begin{bmatrix} e_{i_1}^T \Phi(t_1; t_0, \hat{x}_0) \\ \vdots \\ e_{i_k}^T \Phi(t_k; t_0, \hat{x}_0) \end{bmatrix}.$$

Numerical results on a concrete example are presented in Section 4.1.5.

4.1.4 Event-based observability and detectability

As we have seen in the previous section the state reconstruction problem can be approached at least numerically. However there are a number of fundamental issues that can be immediately raised in connection to the state reconstruction problem from partial discrete measurements. Answering these issues is a challenging task, and even in the linear time-invariant case this is still an open problem.

If the solution to the state reconstruction problem does not exist, it is of course irrelevant to study which method might be suitable to find it. Hence, it is important to guarantee that every initial state can be reconstructed from a long enough sequence of recorded state events and a suitably chosen input signal. This leads us to consider the following notion.

Definition 4.1.2 (Event-based observability) *A state $x_0 \in \mathbb{R}^n$ is said to be* event-based observable *(or* event-based reconstructible*) for the system (4.1) together with the level sensors (4.2), if there exists an input υ resulting in the sequence of events given by $\mathcal{T} = \{t_1, ..., t_k\}$, $\mathcal{I} = \{i_1, ..., i_k\}$ and $\mathcal{V} = \{b_1, ..., b_k\}$ for which the state reconstruction problem has a unique solution (equal to x_0).*

We have no practical criterion to assess reconstructibility of a state of a system. It is however not difficult to give some situations in which a state is not reconstructible. The extreme case is when the state trajectory of an autonomous system (*i.e.* with no inputs), initialized in the given state never crosses a level boundary. For example, when the respective state is one of two equilibrium states within one hypercube. However, this is not particularly bothersome in many practical situations. Indeed, if the sensor system is used for safety issues, it is important that the critical states that lead to unbounded solutions can be observed.

DEFINITION 4.1.3 (EVENT-BASED DETECTABILITY) *A state $x_0 \in \mathbb{R}^n$ is said to be* event-based detectable *for the system (4.1) together with the level sensors (4.2) if for all $\upsilon \in PC^0[0,\infty)$, a state trajectory initialized in x_0 is either bounded, or the resulting sequence of events given by $\mathcal{T} = \{t_1,...,t_k\}$, $\mathcal{I} = \{i_1,...,i_k\}$ and $\mathcal{V} = \{b_1,...,b_k\}$ is such that the state reconstruction problem has a unique solution (equal to x_0).*

For practical purposes it is often important to be able to determine the initial state out of a known set of possible initial states. Also it can be convenient to know how many observations it at least takes to reconstruct x_0. Specifically, we introduce the following definition.

DEFINITION 4.1.4 (SET RESTRICTED k-STEP OBSERVABILITY) *A state $x_0 \in \mathcal{S} \subset \mathbb{R}^n$ is said to be* set restricted k-step observable *for the system (4.1) together with the level sensors (4.2) if there exists an input $\upsilon \in PC^0[0,\infty)$ such that with the corresponding sequence of events, of length k, $\mathcal{T} = \{t_1,...,t_k\}$, $\mathcal{I} = \{i_1,...,i_k\}$, and $\mathcal{V} = \{b_1,...,b_k\}$ generated by the state trajectory initialized in x_0 with input υ, the state reconstruction problem has a unique solution in the set $\mathcal{S}$, equal to x_0.*

The advantage now is that we can exploit the fact that x_0 is known to be an element of $\mathcal{S}$. The more restrictive $\mathcal{S}$ is, the easier the state reconstruction problem becomes.

The level sensors (4.2) naturally induce a partitioning of the state space. Based on this partitioning we are able to create a discrete-event model of the continuous system as discussed in the previous chapter. Suppose the set $\mathcal{S}$ is equal to the region in the state space corresponding to the particular discrete state $\tilde{x}_0$ which contains x_0, *i.e.* $\mathcal{S} = H_x(\tilde{x}_0)$. Then, it is possible, using the discrete-event model of the system, to determine all possible transition sequences of length k when starting in $\tilde{x}_0$. Since it is possible that more than one adjacent discrete states can be reached from a given discrete state $\tilde{x}$, various transition sequences can occur. Suppose we have l possible transition sequences, then for all sequences the index sets $\mathcal{I}_1,...,\mathcal{I}_l$ and the

measurement sets $\mathcal{V}_1, ...\mathcal{V}_l$ are known. The only missing information are the time sequences $\mathcal{T}_1, ..., \mathcal{T}_l$. However, for linear time-invariant systems it can be decided on this partial information whether a continuous state is *not* set restricted k-step observable. Suppose, for the set of indices $\mathcal{I}_j$ we construct the 'output' matrix $E_{\mathcal{I}_j} = [e_{i_1}, ...e_{i_k}]^T$. Now, it is intuitively clear that if the pair $(A, E_{\mathcal{I}_j})$ is not observable then the state x_0 cannot be reconstructed from the discrete measurements, since even less information is available then is the case for continuous measurements. Hence, all pairs $(A, E_{\mathcal{I}_j})$, $j = 1, ..., l$ being observable is a necessary condition for set restricted k-step observability. To be specific, for the linear time-invariant case, the following holds.

PROPOSITION 4.1.5 *Given an index set $\mathcal{I}$ and a time set $\mathcal{T}$, define the 'output' matrix $E = [e_{i_1}, ...e_{i_k}]^T$. Furthermore let P be defined as in (4.8), then (A, E) not observable implies that P is not injective and hence the state reconstruction problem does not have a unique solution.*

Proof. Suppose (A, E) is not observable, then clearly there exists a vector v such that

$$\begin{bmatrix} E \\ EA \\ EA^2 \\ \vdots \\ EA^{n-1} \end{bmatrix} v = 0.$$

Taking the Taylor expansion of e^{At} and using the Caley-Hamilton theorem we can write $e^{At} = \alpha_1(t)I + \alpha_2(t)A + ... + \alpha_n(t)A^{n-1}$. Using this we obtain for all t that

$$Ee^{At}v = E\left(\alpha_1(t)I + \alpha_2(t)A + ... + \alpha_n A^{n-1}\right) v = 0.$$

From which we get that

$$\begin{bmatrix} Ee^{At_1} \\ Ee^{At_2} \\ \vdots \\ Ee^{At_k} \end{bmatrix} v = 0 \Longrightarrow \begin{bmatrix} e_{i_1}^T e^{At_1} \\ e_{i_2}^T e^{At_2} \\ \vdots \\ e_{i_k}^T e^{At_k} \end{bmatrix} v = Pv = 0,$$

proving that P does not have full column rank (is not injective) and hence the state reconstruction problem does not has a unique solution. ■

Even if it is known that $x_0 \in \mathcal{S}$, where $\mathcal{S}$ is an open set in $\mathbb{R}^n$, then still the state reconstruction problem is not solvable, since in this case for any point

x_0 in $\mathcal{S}$ there exists a point $x_0 + \varepsilon v$ that is also in $\mathcal{S}$ for ε small enough and v satisfying $Pv = 0$. If however $\mathcal{S}$ is a lower dimensional manifold in $\mathbb{R}^n$ then this reasoning no longer holds and the state reconstruction problem might be solvable.

4.1.5 Three tank example

To illustrate the method explained in the foregoing, it is applied to a three tank system depicted in Figure 4.1.

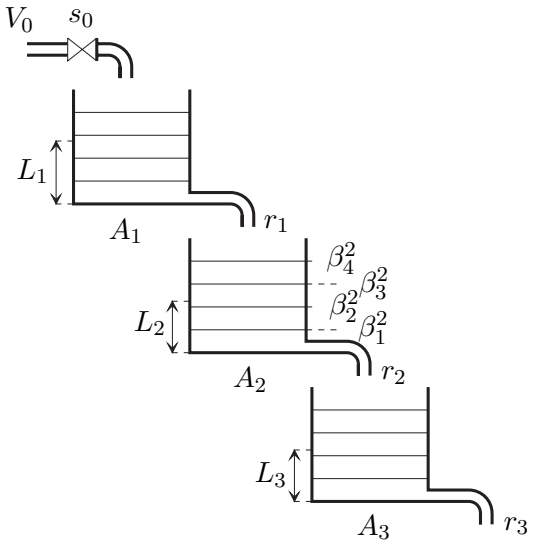

Figure 4.1: *Three tank system (series)*

The system consists of three tanks in cascade. Water enters the first tank with a volume flow V_0 $[m^3/s]$ when the switch $s_0 \in \{0, 1\}$ controlling the valve is on ($s_0 = 1$). The water flows from the first (second) into the second (third) tank. The last tank has a drain. The state vector $x = [L_1, L_2, L_3]^T$ is given by the water levels in each tank. The model of the three tank system used for the simulation is given by

$$\dot{x}^1 = \frac{1}{A_1}(-2\pi r_1^2\sqrt{2g\ x^1} + V_0\ u),$$
$$\dot{x}^2 = \frac{1}{A_2}(2\pi r_1^2\sqrt{2g\ x^1} - 2\pi r_2^2\sqrt{2g\ x^2}),$$
$$\dot{x}^3 = \frac{1}{A_3}(2\pi r_2^2\sqrt{2g\ x^2} - 2\pi r_3^2\sqrt{2g\ x^3}),$$

with $x^i = L_i$ and $u = s_0 \in \{0, 1\}$. The values of the parameters used for

the simulation are: $r_i = 0.01\ [m], g = 9.81\ [m/s^2], A_i = \pi(0.05)^2\ [m^2], V_0 = 0.001\ [m^3/s]$.

The event detectors measuring the levels are mounted at the same positions for each tank, so each state has the same set of boundaries

$$\mathcal{B}^i = \{0.02,\ 0.05,\ 0.10,\ 0.15\}, \qquad i = 1, 2, 3.$$

Starting with the initial condition $x_0 = x(0) = [0.07,\ 0.12,\ 0.03]^T$, the system is simulated during one second. The input is given by

$$u(t) = \begin{cases} 0, & 0 \le t \le \frac{2}{3} \\ 1, & \frac{2}{3} < t \le 1 \end{cases}$$

The first event occurs after 0.165 seconds; the third state variable hits the boundary $x^3 = \beta_2^3 = 0.05$. The time response is depicted in Figure 4.2.

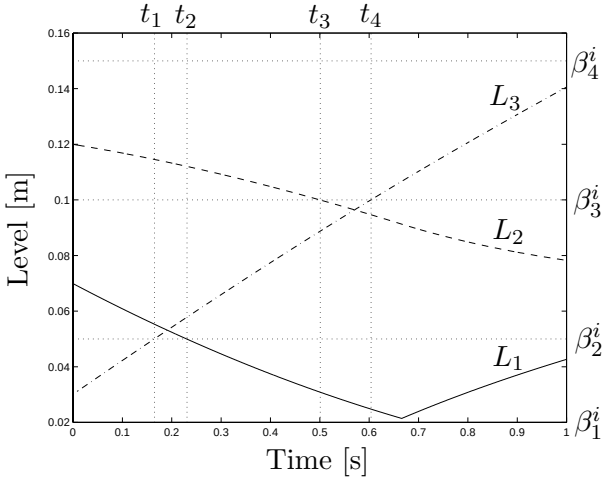

Figure 4.2: *Time response*

After one second four events have happened. The data we now have is

$$\mathcal{T} = \{0.165,\ 0.231,\ 0.501,\ 0.604\},$$
$$\mathcal{I} = \{3,\ 1,\ 2,\ 3\},$$
$$\mathcal{V} = \{0.05\ , 0.05,\ 0.1,\ 0.1\},$$
$$b = [0.05,\ 0.05,\ 0.1,\ 0.1]^T.$$

For the initial guess of x_0 it is reasonable to assume that it is known in which cube, defined by the boundaries, of the state space $x(t)$ is at $t = t_0$. Using the initial guess $\hat{x}_0 = [0.09,\ 0.14,\ 0.01]^T$ the procedure explained in Section 4.1.3 is followed. After six iterations the procedure is stopped when $J(\hat{x}_0) < 1{\cdot}10^{-8}$. The estimation of x_0 then is $\hat{x}_0 = \begin{bmatrix} 0.070 & 0.120 & 0.030 \end{bmatrix}^T$ which is equal to the real x_0 known from the simulation.

4.2 Discrete-state reconstruction

In Chapter 3 it is assumed that the initial continuous state is known to be in the interior of a hypercube to avoid problems with the mapping Q. If this initial hypercube $H_x(\tilde{x}_0)$ is known (and consequently $\tilde{x}_0$ is known) and every component of the continuous state is measured (*i.e.* each boundary β_j^i) then after an arbitrary number of measurements (transitions) the discrete state is still exactly known. If the initial discrete state $\tilde{x}_0$ is not known or if not all components of the continuous state x are observed, the discrete state after k measurements may be unknown. However, from measured transitions (outputs) it may be possible to reconstruct the discrete state. In the sequel it is assumed that the initial discrete state is not given, but at least the continuous state is known to be in the interior of a hypercube associated with the unknown discrete state.

4.2.1 Discrete-state measurement

First we assume that to each discrete state an output is assigned, *i.e.* actual discrete states are measured.

Unknown input

Suppose we are given an output automaton Σ defined by the 5-tuple

$$\Sigma = (\tilde{X}, \tilde{U}, \phi, \tilde{Y}, h),$$

where $\tilde{X}$ is the set of discrete states, $\tilde{U}$ is a finite set of inputs (the input alphabet), $\tilde{Y}$ is a finite set of outputs (the output alphabet), $\phi : \tilde{X} \times \tilde{U} \to 2^{\tilde{X}}$ is the transition function and finally $h : \tilde{X} \to \tilde{Y}$ is the output map.

Problem 4.2.1 (The discrete-state reconstruction problem) *Given an automaton Σ and a measured sequence $\tilde{y}_1\tilde{y}_2...\tilde{y}_k \in \tilde{Y}^k$ of length k, find the state $\tilde{x} \in \tilde{X}$ of the automaton after the k-th measurement.*

First for each measurement $\tilde{y}_l$, $l \in \{1, ..., k\}$ the following sets are defined:

$$\tilde{X}_l := \{\tilde{x} \mid \exists \tilde{u} \in \tilde{U}, \tilde{z} \in \tilde{X}_{l-1} \text{ such that } h(\tilde{x}) = \tilde{y}_l \text{ with } \tilde{x} \in \phi(\tilde{z}, \tilde{u})\}, \tag{4.16}$$

with $\tilde{X}_0 := \tilde{X}$ (the set of all discrete states).

The following proposition is straightforward.

Proposition 4.2.2 *Given an automaton Σ together with a measured sequence $\tilde{y}_1\tilde{y}_2...\tilde{y}_k \in \tilde{Y}^k$ of length k, then after the l-th measurement ($l \in \{0, 1, ..., k)$ we have that $\tilde{x} \in \tilde{X}_l$. Moreover $\tilde{X}_l$ is the smallest set which can contain $\tilde{x}$ based on the first l measurements.*

Proof. Clearly, before the first measurement it holds that $\tilde{x} \in \tilde{X}_0$ which is the smallest set containing $\tilde{x}$ (as we do not have any measurement information). Next, suppose that before the l-th measurement we have that $\tilde{x} \in \tilde{X}_{l-1}$, then after observing $\tilde{y}_l$ it is certain that $\tilde{x}$ must be in the set that can be reached from $\tilde{X}_{l-1}$ with some input $\tilde{u} \in \tilde{U}$, causing the measurement $\tilde{y}_l$. This is exactly the set described in (4.16). ■

Based on this information we have after l observations, that $\tilde{X}_l$ is the smallest set for which we are certain that it contains x.

REMARK 4.2.3 *Using Boolean notation it is easy to compute $\tilde{X}_l$ if we use the output matrix H defined as*

$$\hat{y}^T H \hat{x} = 1 \Longleftrightarrow h(\tilde{x}) = \tilde{y},$$

with $\hat{x}$ and $\hat{y}$ in Boolean vector representation of $\tilde{x} \in \tilde{X}$ and $\tilde{y} \in \tilde{Y}$. The set $\tilde{X}_l$ (in Boolean vector notation represented by the single vector $\hat{x}_l$) is computed by

$$\hat{x}_l = (A_{\tilde{U}} \hat{x}_{l-1}) \otimes (H^T \hat{y}_l).$$

To see this, realize that $\hat{z}_1 := A_{\tilde{U}} \hat{x}_{l-1}$ represents the set of discrete states that can be reached from $\tilde{X}_{l-1}$ with one of the inputs $\tilde{u} \in \tilde{U}$. Furthermore, $\hat{z}_2 := H^T \hat{y}_l$ represents the set of states that will result in the measurement $\tilde{y}_l$. Consequently, the intersection of $\tilde{Z}_1 \cap \tilde{Z}_2$, computed by $\tilde{z}_1 \otimes \tilde{z}_2$, yields all discrete states that can be reached from $\tilde{X}_{l-1}$ and for which $\tilde{y}_l$ is the output.

Known input

Previously it is assumed that the applied input sequence is unknown. The results however are easily extended to the more relevant case where the applied inputs are known.

COROLLARY 4.2.4 *Given an automaton Σ and a measured sequence $\tilde{y}_1 \tilde{y}_2 ... \tilde{y}_k \in \tilde{Y}^k$ of length k, and the applied input sequence $\tilde{u}_1 \tilde{u}_2 ... \tilde{u}_k \in \tilde{U}^k$, then Proposition 4.2.2 still holds if (4.16) is replaced by*

$$\tilde{X}_l = \{\tilde{x} \mid \exists \tilde{z} \in \tilde{X}_{l-1} \text{ such that } h(\tilde{x}) = \tilde{y}_l \text{ with } \tilde{x} \in \phi(\tilde{z}, \tilde{u}_l)\}.$$

Furthermore since $\{\tilde{u}_l\} \subset \tilde{U}$ we have that X_l as computed above has less or an equal number of elements than in the case that no input sequence is known (i.e. when (4.16) is used).

REMARK 4.2.5 *For a known input sequence the set $\tilde{X}_l$ is computed explicitly in the Boolean vector domain by*

$$\hat{x}_l = (A_{\tilde{u}_l} \hat{x}_l) \otimes (H^T \hat{y}_l).$$

REMARK 4.2.6 *For ease of explanation, throughout this section it is assumed that (discrete) inputs only change when a measurement is made. This implies that the measurement sequence and the input sequence have the same length. However, all procedures described in this section can be adapted for the case where inputs change in between measurements.*

4.2.2 Event measurements

Instead of the measurements of discrete outputs, in many practical cases the changes between discrete states are observed. That is, we detect the occurrence of an event. The *continuous*-state reconstruction problem was actually based on the detection of events. The discrete-state reconstruction problem is easily translated to the case were the elements of the observed sequence $\tilde{y}_1\tilde{y}_2...\tilde{y}_l$ are related with events instead of discrete states. For this, only the output-map h must be adapted. Instead of mapping a discrete state to a discrete output, the adapted output map must translate the difference between two discrete states to an output, that is $h : \tilde{X} \times \tilde{X} \to \tilde{Y}$ is given by

$$\tilde{y} = h(\tilde{x}_{new,}\tilde{x}_{old}), \qquad \tilde{x}_{new}, \tilde{x}_{old} \in \tilde{X}.$$

In case of discrete-state measurements we actually used the fact that h was complete (*i.e.* for each state, an output was defined). However, for event measurements this assumption would be too restrictive, because then for all combinations of discrete states h must be defined. For our purposes, it is sufficient to assume that $h(\tilde{x}_{new}, \tilde{x}_{old})$ is defined for all pairs of adjacent discrete states $(\tilde{x}_{old}, \tilde{x}_{new}) \in \tilde{X} \times \tilde{X}$ or for which $\tilde{x}_{new} \in \phi(\tilde{x}_{old}, \tilde{u})$ for some $\tilde{u} \in \tilde{U}$. This is equal to the assumption that all boundaries β^i_j are actually observed or that each crossing of a boundary is measured, respectively. This implies that no transition from one discrete state to another is possible without being observed. Notice that it is not known what the direction of a crossing is, that is $h(\tilde{x}_1, \tilde{x}_2) = h(\tilde{x}_2, \tilde{x}_1)$.

Complete transition measurement

First, we consider the case that each transition (event) is observed. Under this assumption, Proposition 4.2.2 holds if (4.16) is replaced by

$$\tilde{X}_l = \{\tilde{x} \mid \exists \tilde{u} \in \tilde{U}, \tilde{z} \in \tilde{X}_{l-1} \text{ such that } h(\tilde{x}, \tilde{z}) = \tilde{y}_l \text{ with } \tilde{x} \in \phi(\tilde{z}, \tilde{u})\}.$$

In case of known inputs, we can improve upon the above expression by replacing it by

$$\tilde{X}_l = \{\tilde{x} \mid \exists \tilde{z} \in \tilde{X}_{l-1} \text{ such that } h(\tilde{x}, \tilde{z}) = \tilde{y}_l \text{ with } \tilde{x} \in \phi(\tilde{z}, \tilde{u}_l)\}.$$

REMARK 4.2.7 *For describing the output function in case of discrete-event measurements we can use the labelled output matrix* $\mathbf{H} = \mathbf{H}^T$ *defined as*

$$\hat{x}_2^T \mathbf{H} \hat{x}_1 = \tilde{y} \Longleftrightarrow h(\tilde{x}_2, \tilde{x}_1) = \tilde{y},$$

where $\hat{x}_i$ *is the Boolean representation of* $\tilde{x}_i$, $i = 1, 2$. *Furthermore by using the notation* $\tilde{Y} \ni \tilde{y} = 1 \Longleftrightarrow \tilde{y} \in \tilde{Y}$ *the set* $\tilde{X}_l$ *can be computed explicitly (with some abuse of notation) by*

$$\hat{x}_l = (A_{\tilde{U}} \otimes \mathbf{H}) \hat{x}_{l-1} \ni \tilde{y}_l,$$

and in case of a known input by

$$\hat{x}^l = (A_{\tilde{u}_l} \otimes \mathbf{H}) \hat{x}_{l-1} \ni \tilde{y}_l.$$

This should be interpreted as follows. The operation $A_{\tilde{U}} \otimes \mathbf{H}$ *results in a labelled matrix for which the* ij*-th element is* $\tilde{y} \in \tilde{Y}$ *if and only if the* ij*-th elements of* $A_{\tilde{U}}$ *and* $\mathbf{H}$ *are '*1*' and* $\tilde{y}$*, respectively. By multiplication with the Boolean vector* $\hat{x}_{l-1}$*, a vector results for which a component is either zero, or consists of a (set of) symbol(s)* $\tilde{Y}_l \subseteq \tilde{Y}$. *By checking this vector (component wise) for the element* $\tilde{y}_l$ *only a '*1*' arises at coordinates that also contain the symbol* $\tilde{y}_l$. *Otherwise, a '*0*' results. As a consequence the Boolean vector* $\hat{x}_l$ *is computed this way.*

Partial transition measurement

Finally, the more general case is considered where not all transitions from one discrete state to another are measured. This happens when not all boundaries β_j^i defining the partitioning of the state space are actually observed. Now, after each measurement $\tilde{y}_l$ the discrete state can evolve to other states without knowing this. Therefore, after each measurement we have to compute all discrete states that can be reached from a given set without causing events. We only consider the case where the input is known.

The set of states that can be reached from a given initial set $\tilde{S}$ with input $\tilde{u} \in \tilde{U}$ without causing measurements is denoted by $\tilde{M}(\tilde{S}, \tilde{u})$ and is computed by the following recursive scheme

ALGORITHM 4.2.8 *Given input* $\tilde{u}$ *and initial set* $\tilde{S}$, *define* $\tilde{M}_0(\tilde{S}, \tilde{u}) := \tilde{S}$ *and then for* $k \geq 0$, *repeat*

$$\tilde{M}_{k+1}(\tilde{S}, \tilde{u}) = \{\tilde{x} \in \phi(\tilde{z}, \tilde{u}) \mid \tilde{z} \in \tilde{M}_k(\tilde{S}, \tilde{u}),\ h(\tilde{x}, \tilde{z}) \notin \tilde{Y}\} \cup \tilde{M}_k(\tilde{S}, \tilde{u}),$$

until $\tilde{M}_{k+1}(\tilde{S}, \tilde{u}) = \tilde{M}_k(\tilde{S}, \tilde{u})$. *The set* $\tilde{M}(\tilde{S}, \tilde{u})$ *is equal to* $M_k(\tilde{S}, \tilde{u})$.

With this, Proposition 4.2.2 holds if (4.16) is replaced by

$$\tilde{X}_l = \{\tilde{x} \mid \exists \tilde{z} \in M(\tilde{X}_{l-1}, \tilde{u}), \text{ such that } h(\tilde{x}, \tilde{z}) = \tilde{y}_l \text{ with } \tilde{x} \in \phi(\tilde{z}, \tilde{u})\}.$$

REMARK 4.2.9 *Using Boolean vector notation the recursive scheme for computing $\tilde{M}(\tilde{S}, \tilde{u})$ is executed easily. To do so, we will use the Boolean matrix version H of the labelled output matrix $\mathbf{H}$ (i.e. $h_{ij} = 1 \Longleftrightarrow \mathbf{h}_{ij} \in \tilde{Y}$). Then, $\tilde{M}_{k+1}(\tilde{S}, \tilde{u})$ is computed by*

$$\hat{m}_{k+1}(\tilde{S}, \tilde{u}) = (A_{\tilde{u}} \ominus H)\hat{m}_k(\tilde{S}, \tilde{u}) \oplus \hat{m}_k(\tilde{S}, \tilde{u}).$$

Indeed, the set represented by $\hat{z}_1 := A_{\tilde{u}}\hat{m}_k(\tilde{S}, \tilde{u})$ contains all discrete states that can be reached from states in $\tilde{M}_k(\tilde{S}, \tilde{u})$ with input $\tilde{u}$. Furthermore, $\hat{z}_2 := H\hat{m}_k(\tilde{S}, \tilde{u})$ represents the discrete states to which transitions from $\tilde{M}_k(\tilde{S}, \tilde{u})$ would result in some measurement $\tilde{y} \in \tilde{Y}$. Consequently, the set difference $\tilde{Z}_1 \setminus \tilde{Z}_2$ computed by $\hat{z}_1 \ominus \hat{z}_2 = (A_{\tilde{u}} \ominus H)\hat{m}_k(\tilde{S}, \tilde{u})$ contains all discrete states that can be reached from $\tilde{M}_k(\tilde{S}, \tilde{u})$ with input $\tilde{u}$ without causing a measurement. As a consequence, the computation of $\tilde{X}_l$ in Boolean vector notation becomes

$$\hat{x}_l = (A_{\tilde{u}_l} \otimes \mathbf{H})\hat{m}(\tilde{X}_{l-1}, \tilde{u}_l) \textcircled{\ni} \tilde{y}_l.$$

4.2.3 Rolling ball example

Consider a square in a two-dimensional space which is divided into 9 regions defined by the lines b_1, b_2, b_3, and b_4 as depicted in Figure 4.3. Now suppose a ball would roll over the square resulting in the path as shown in Figure 4.3, starting in square 1. This path is only observed when the ball crosses a line b_i. This will result in the observed sequence $b_1b_3b_2b_4$. The purpose now is to determine the region the ball is in after the fourth observation. First,

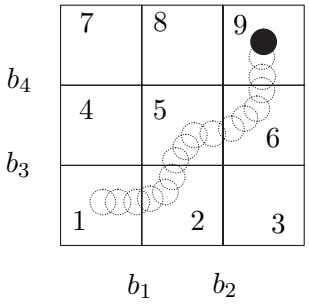

Figure 4.3: *A trajectory in a 2-D plane*

the corresponding automaton Σ has to be constructed. The discrete states are associated to the regions defined by b_1, b_2, b_3, and b_4, and denoted by a number:

$\tilde{x}_{old}/\tilde{x}_{new}$	1	2	3	4	5	6	7	8	9
1	–	b_1	–	b_3	–	–	–	–	–
2	b_1	–	b_2	–	b_3	–	–	–	–
3	–	b_2	–	–	–	b_3	–	–	–
4	b_3	–	–	–	b_1	–	b_4	–	–
5	–	b_3	–	b_1	–	b_2	–	b_4	–
6	–	–	b_3	–	b_2	–	–	–	b_4
7	–	–	–	b_4	–	–	–	b_1	–
8	–	–	–	–	b_4	–	b_1	–	b_2
9	–	–	–	–	–	b_4	–	b_2	–

Table 4.1: *The output map h*

1 for the lower left square, and increasing from the left to the right, from the bottom to the top. Now Σ is defined by $\tilde{X} = \{1, 2, ..., 9\}$, $\tilde{Y} = \{b_1, b_2, b_3, b_4\}$. The transition function ϕ is defined by the directed graph depicted in Figure 4.4.

Note, that we assume the input sequence to be unknown. The output map

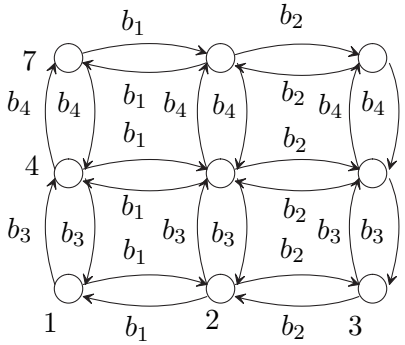

Figure 4.4: *Directed graph with measurements as labels*

h is given in Table 4.1.

This results in the labelled measurement matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} 0 & b_1 & 0 & b_3 & 0 & 0 & 0 & 0 & 0 \\ b_1 & 0 & b_2 & 0 & b_3 & 0 & 0 & 0 & 0 \\ 0 & b_2 & 0 & 0 & 0 & b_3 & 0 & 0 & 0 \\ b_3 & 0 & 0 & 0 & b_1 & 0 & b_4 & 0 & 0 \\ 0 & b_3 & 0 & b_1 & 0 & b_2 & 0 & b_4 & 0 \\ 0 & 0 & b_3 & 0 & b_2 & 0 & 0 & 0 & b_4 \\ 0 & 0 & 0 & b_4 & 0 & 0 & 0 & b_1 & 0 \\ 0 & 0 & 0 & 0 & b_4 & 0 & b_1 & 0 & b_2 \\ 0 & 0 & 0 & 0 & 0 & b_4 & 0 & b_2 & 0 \end{bmatrix}.$$

The discrete states are represented in the Boolean vector domain *e.g.* $\tilde{x} = 3$ is written as $\tilde{x} = [0, 0, 1, 0, 0, 0, 0, 0, 0]^T$. For the outputs we still use the symbolic presentation, *e.g.* $\tilde{y} = b_2$.

Furthermore the adjacency matrix $A \in B^{9\times 9}$of the directed graph shown in Figure 4.4 is given by:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

With this, the following sequence of sets $\tilde{X}_l$ results after the successive observations:

$$\begin{aligned} \hat{x}_1 &= (A \otimes \mathbf{H})\hat{x}_0 \ominus b_1, \\ \hat{x}_2 &= (A \otimes \mathbf{H})\hat{x}_1 \ominus b_3, \\ \hat{x}_3 &= (A \otimes \mathbf{H})\hat{x}_2 \ominus b_2, \\ \hat{x}_4 &= (A \otimes \mathbf{H})\hat{x}_3 \ominus b_4. \end{aligned}$$

The corresponding sets are depicted in Figure 4.5.

From Figure 4.5 it can be seen that after four measurements the correct discrete state is found by the proposed procedure.

4.3 Notes and references

Continuous-state reconstruction

The continuous-state reconstruction procedure discussed in this chapter is based on material presented in (Philips and Weiss 1998) and deals with both linear and nonlinear systems.

In (Raisch 1993) the continuous state is reconstructed from discrete measurements for systems that are piecewise linear for each discrete input. The reconstructed continuous output is used for control purposes. Also an observability condition is given involving controllable and reachable sets. In (Delchamps 1989) the idea is posed to design a feedback control law that helps to gather information on the continuous state. Recently, (Schnabel and

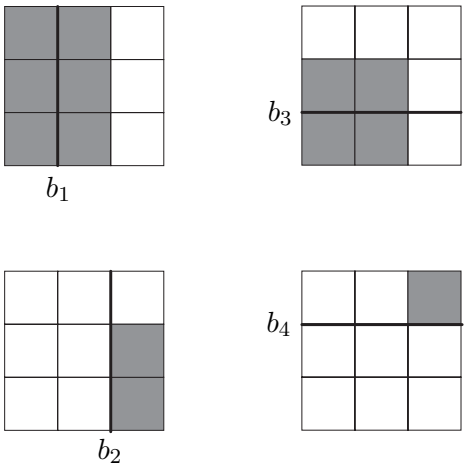

Figure 4.5: *The sets $\tilde{X}_l$ (grey): possible states following from a new measurement*

Krebs 2000) proposed a method for reconstructing the continuous state for affine continuous systems. Their method is based on the construction of sets containing those states that allow evolution without reaching sensors within the observed time. In (Phillips and Tomizuka 1995) a state reconstruction method is presented that utilizes the Luenberger observer framework. For this, it is necessary that always the same state-variables are measured when an event occurs.

Discrete-state reconstruction

The reconstruction of the discrete state as presented in this section is discussed in terms of automata. The fact that an automaton may result from the discretization of a continuous plant is not important (for the algorithm) and is only reflected in the type of measurements that is referred to as 'event measurements'. This type of measurements (*i.e.* the measurement map $h : \tilde{X} \times \tilde{X} \to \tilde{Y}$) is not standard in automata theory, but is especially suited for our purpose. If the automaton results from a discretization of a continuous plant, then completeness of the model implies that results obtained by the observation of the discrete event model imply that the corresponding discretized states can be assumed by the continuous system and, moreover, the other states can be excluded.

In (Booth 1967) the so-called initial and terminal state identification problems are solved by applying a prespecified input sequence such that the automaton is brought to a known state (for the terminal state identification problem) or the initial state can be determined from observed outputs. Ob-

servability of discrete-event systems based on automata is discussed in (Lin and Wonham 1988), where observation of discrete-event systems is combined with supervisory control. For automata without inputs, (Özveren and Willsky 1990) propose an observer represented by an automaton that determines the discrete state at points in time, based on partial measurements of transitions.

Concerning the observation of qualitative (*i.e.* discrete) states of continuous systems, (Lichtenberg and Lunze 1997) present a method for reconstruction of the discrete state based on qualitative (discrete-event) models of continuous systems represented by (nondeterministic) automata or stochastic automata with event measurements. For this, the automaton is described by distinct Boolean matrices for each input-output combination. For the stochastic automaton, the Boolean matrices are replaced by matrices which entries represent the possibility of a transition. It is assumed that each new state results in a new measurement. For discrete-time systems with symbolic observations (*i.e.* discrete measurements) that are used for piecewise smooth feedback or finite state dynamic feedback, (Ramadge 1990) presents conditions under which the symbolic observations become periodic.

Chapter 5

Control strategies

In this chapter methods for controller design are proposed for systems with a 'discretized' state space and input space, as discussed in Chapter 3. Instead of designing the supervisor solely on the basis of the discrete approximation of the continuous plant, also information provided by the continuous system is incorporated in the synthesis. The fact that the underlying system is continuous allows control actions that would not be possible for pure discrete-event systems. As a consequence, the approach pursued here offers additional possibilities compared to methods suited for pure discrete-event systems. The proposed methods allow structured controller design where the computations (based on Boolean vector operations) are given explicitly. As an advantage, no specialistic insight or knowledge of operators of the plant is needed, which is often the case with 'rule-based' control strategies. Also, for complex systems, structured controller synthesis reduces the possibility that the controller is not prepared for all possible situations that can occur.

5.1 Control goals

The problem of controlling a continuous system using only discrete-state information and acting on a finite number of discrete inputs, can be related to various practical situations. For instance, it involves the control of a system that is continuous by nature, but is observed by discrete sensors only (*i.e.* sensors only detecting when a state crosses a certain boundary). It is also of importance for the situation where a continuous plant is to be supervised by some programmable logical controller or computer program which builds on discrete-state information. In such situations, the controller often will be a discrete-event system. Since a discrete-event controller cannot communicate with the system at a continuous level it is necessary to use an interface. This can be modelled as discretizing the continuous state space of the system into

a finite set of symbols to be used by the controller as explained in Chapter 3. The discrete-event models extracted from continuous systems as discussed in Chapter 3 are now used for control purposes.

First, recall that the systems under consideration are of the following form:

$$\dot{x}(t) = f(x(t), u(t)), \qquad x(t_0) = x_0, \tag{5.1}$$

with $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, together with the set of boundaries for each component x^i, $(i = 1, ..., n)$,

$$\beta_0^i < \beta_1^i < \ldots < \beta_{n_i}^i \quad (n_i \geq 1), \tag{5.2}$$

and, when applicable, for each component u_i, $(i = 1, ...m)$,

$$\gamma_0^i < \gamma_1^i < \ldots < \gamma_{m_i}^i \quad (m_i \geq 1). \tag{5.3}$$

Furthermore, these boundaries induce hypercubes $H_x(\tilde{x})$ and $H_u(\tilde{u})$ in the state space and input space, respectively which are associated with discrete state $\tilde{x}$ and discrete input $\tilde{u}$.

The proposed controller design methods are based on discrete inputs $\tilde{u} \in \tilde{U}$ in such a way that any continuous input $u(t) \in H_u(\tilde{u})$ can be chosen when the discrete input $\tilde{u}$ should be applied to the system according to the discrete controller. This means that while applying discrete input $\tilde{u}$, some freedom is left for low-level control; the only requirement is that $u(t) \in H_u(\tilde{u})$. This makes the proposed strategies suitable for supervisory control. In the case where $u(t)$ already takes values from some discrete set $\tilde{U}$, it is understood that $H_u(\tilde{u}) = \{\tilde{u}\}$ in the remainder of this chapter.

The only available information for the controller, is the knowledge of the hypercube $H_x(\tilde{x})$ the continuous trajectory lives in. Loosely speaking, the controller applies a piecewise constant (in terms of discrete inputs) input to the system, which only changes when the trajectory goes (or is about to go) from one hypercube to another. The following definition is given.

DEFINITION 5.1.1 *Given the system (5.1). A solution trajectory $\xi \in C^0[0, T]$ of (5.1) for input $\upsilon \in PC^0[0, T]$ is called* discretely controlled.

In this chapter two different control goals are distinguished:

The reachability problem Given the system (5.1), a set of boundaries (5.2), the discrete initial state $\tilde{x}_0$, the set of target states $\tilde{X}_e$, and a set of discrete inputs $\tilde{U}$. Find a controller that realizes for any continuous initial state $x_0 \in H_x(\tilde{x}_0)$ a discretely controlled trajectory ξ (with $\xi(0) = x_0$) that intersects $int\left(\bigcup_{\tilde{x} \in \tilde{X}_e} H_x(\tilde{x})\right)$.

The reachability problem is the problem of controlling the trajectory from an initial discrete state to a desired final set of discrete states (more precisely, to the corresponding hypercubes in state space).

The stabilization problem Given the system (5.1), a set of boundaries (5.2), the discrete initial state $\tilde{x}_0$, the set of target states $\tilde{X}_e$, and a set of discrete inputs $\tilde{U}$. Find a controller that realizes for any continuous initial state $x_0 \in H_x(\tilde{x}_0)$ a discretely controlled trajectory ξ (with $\xi(0) = x_0$) intersecting $int\left(\bigcup_{\tilde{x}\in\tilde{X}_e} H_x(\tilde{x})\right)$ in such a way that for all $t_e > 0$ with $\xi(t_e) \in int\left(\bigcup_{\tilde{x}\in\tilde{X}_e} H_x(\tilde{x})\right)$ it holds that $\xi(t) \in \bigcup_{\tilde{x}\in\tilde{X}_e} H_x(\tilde{x})$ for all $t \geq t_e$.

The stabilization problem is the problem of controlling the state trajectory from an initial discrete state to a desired set of discrete states and preventing the trajectory from leaving the corresponding set of hypercubes after the state has entered the interior.

REMARK 5.1.2 *Note that this notion of stability implies the more natural stability for which only the existence of t_e such that $x(t) \in \bigcup_{\tilde{x}\in\tilde{X}_e} H_x(\tilde{x})$ for all $t \geq t_e$ is required, as e.g. for qualitative stability* (Lunze 1995). *Realizing the latter definition of stability on the basis of the automaton implies that the definition stated here is obtained.*

At this point, it should be realized that the problem to be solved yields the control of nonlinear systems solely on the basis of discrete partial measurements and discrete inputs. Of course, it is impossible to solve this problem in its full generality, as controlling nonlinear systems with complete state measurements and continuous inputs in general is extremely difficult and solutions are found only for limited classes of systems (*e.g.* affine nonlinear control systems (Nijmeijer and Van der Schaft 1990)). As a consequence (and as a first attempt), we will propose three control methodologies, which will work under an appropriate assumption. To overcome problems arising from violating this assumption, several modifications will be discussed at the end of the chapter.

ASSUMPTION 5.1.3 *A discretely controlled trajectory ξ will never reach a point $x \in \mathbb{R}^n$ that belongs to more than two hypercubes, i.e. for all $t \in \mathbb{R}_+$ it holds that $\xi(t) \notin \bigcap_{i\in I} H_x(\tilde{x}_i)$ for all index sets I with cardinality $\#(I) > 2$.*

This assumption is made because otherwise the Assumption 3.2.1 in Chapter 3 would easily be violated. Furthermore, this somewhat facilitates the solution of the difficult nonlinear control problems we have to solve. Moreover, the computation and explanation of the concepts that form the basis of the

control strategies proposed in this chapter are facilitated. In Section 5.8 it is discussed how to adapt (if possible) the proposed control strategies for cases where Assumption 5.1.3 does not hold.

The following concepts are used for specifying the control strategies in this chapter. First, we loosely introduce these notions after which we formalize them in the next section.

Preventing input An input that prevents a transition from happening, see Figure 5.1 (a). A preventing input is applied immediately when a boundary is reached. As such, it is necessary that it is noticed (measured) that a transition possibly is about to occur. This means that we assume that it is exactly measured when the state trajectory reaches a boundary between two hypercubes. Moreover, it is necessary that instantaneous control action is possible.

Correcting input An input that corrects a transition that just has occurred, see Figure 5.1 (b). To use a correcting input it is sufficient that a transition, *i.e.* the actual change of the discrete state, is measured. Also the assumption of instantaneous action can now be relaxed.

Moving input An input for which it is certain that the state trajectory moves towards a particular boundary plane[1]. Applying this input guarantees for all initial states x_0 in hypercube $H_x(\tilde{x})$ that when the trajectory (started in x_0) leaves $H_x(\tilde{x})$, it is closer to the specified boundary plane than the initial state $x_0 \in H_x(\tilde{x})$, see Figure 5.1 (c).

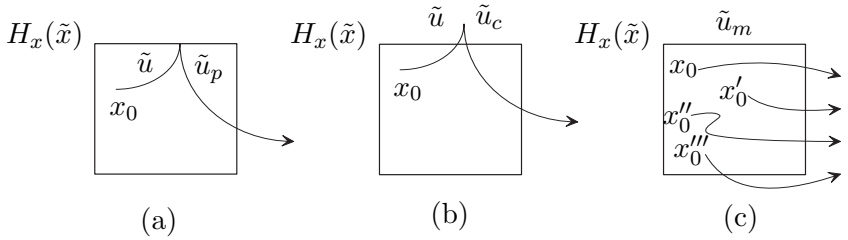

Figure 5.1: *(a) Preserving input $\tilde{u}_p$; (b) correcting input $\tilde{u}_c$; (c) moving input $\tilde{u}_m$*

In the next section, these concepts will be defined more precisely together with methods to decide if a specific input is preventing, correcting or moving for a transition. These types of inputs are the elements from which the proposed controller design methods are built.

[1]Trajectories starting at the specified boundary plane will cross it.

5.2 Preventing, correcting, and moving inputs

5.2.1 Preventing inputs

Let the system (5.1) be given, together with a set of boundaries (5.2), and a finite set of inputs $\tilde{U}$. Fix $\tilde{u} \in \tilde{U}$ and consider the two adjacent states $\tilde{x}_1 = (\tilde{x}^1, ..., \tilde{x}^r, ..., \tilde{x}^n)$ and $\tilde{x}_2 = (\tilde{x}^1, ..., \tilde{x}^r + 1, ..., \tilde{x}^n)$, such that $\{x \in \mathbb{R}^n \mid x^r = \beta_j^r\}$ is the separating hyperplane and $x \in H_x(\tilde{x}_1) \Longrightarrow x^r \leq \beta_j^r$ and $x \in H_x(\tilde{x}_2) \Longrightarrow x^r \geq \beta_j^r$ (as in Condition 3.2.5). Note that j is equal to the r-th coordinate of the n-tuple $\tilde{x}_1$, *i.e.* $j = \tilde{x}^r$.

Definition 5.2.1 (Preventing input) *Given $\tilde{x}_1$ and $\tilde{x}_2$ as in Condition 3.2.5. An input $\tilde{u} \in \tilde{U}$ is* preventing *for the transition $\tilde{x}_1 \to \tilde{x}_2$ if this transition is not possible with the specific input $\tilde{u}$, i.e. $\tilde{x}_2 \notin \phi(\tilde{x}_1, \tilde{u})$.*

This means that $\tilde{u}$ can be used to prevent the state from changing from $H_x(\tilde{x}_1)$ to $H_x(\tilde{x}_2)$.

Proposition 5.2.2 *An input $\tilde{u} \in \tilde{U}$ is preventing for the transition $\tilde{x}_1 \to \tilde{x}_2$ if and only if*

$$f^r(x, u) \leq 0, \forall x \in H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2), \ \forall u \in H_u(\tilde{u}). \tag{5.4}$$

Proof. Follows from Theorem 3.2.6 in Section 3.2. ■

The preventing inputs can be determined directly from the discrete-event model of the discretely-observed continuous system (5.1) that was constructed in Chapter 3. By using the Boolean vector notation $\hat{x}$ for a discrete state $\tilde{x}$ and the set of adjacency matrices $\{A_{\tilde{u}_1}, ..., A_{\tilde{u}_k}\}$ for representing the discrete-event model as explained in Chapter 2, the computation of preventing inputs is straightforward.

Proposition 5.2.3 *Given a discrete state $\tilde{x}_1$, the input $\tilde{u}_i$ is preventing for the transition $\tilde{x}_1 \to \tilde{x}_2$ if and only if $\hat{x}_2^T A_{\tilde{u}_i} \hat{x}_1 = 0$.*

Proof. The set of states $\tilde{X}_3$ that can be reached from $\tilde{x}_1$ with input $\tilde{u}_i$ is represented by $\hat{x}_3 := A_{\tilde{u}_i} \hat{x}_1$. The discrete state $\tilde{x}_2$ is not an element of $\tilde{X}_3$ iff $\hat{x}_2^T \hat{x}_3 = 0$. Clearly, the relation $\hat{x}_2^T A_{\tilde{u}_i} \hat{x}_1 = 0$ is equivalent to $\tilde{x}_2 \notin \phi(\tilde{x}_1, \tilde{u})$. ■

There exists at least one input $\tilde{u} \in \tilde{U} = \{\tilde{u}_1, ..., \tilde{u}_k\}$ that is preventing for $\tilde{x}_1 \to \tilde{x}_2$ if for any of the adjacency matrices $A_{\tilde{u}_l}$ the ij-th element is zero, where i and j are the integer representations of $\tilde{x}_2$ and $\tilde{x}_1$, respectively. Recall that for two Boolean matrices A and B the 'and' operator is defined by $C =$

$A \otimes B$ with $c_{ij} = a_{ij} \cdot b_{ij}$. Now, given a set of discrete inputs $\tilde{U} = \{\tilde{u}_1, ..., \tilde{u}_k\}$, define the Boolean matrix $S_{\tilde{U}} \in \{0,1\}^{p \times p}$ as

$$S_{\tilde{U}} = A_{\tilde{u}_1} \otimes A_{\tilde{u}_2} \otimes ... \otimes A_{\tilde{u}_k} = \bigotimes_{i \in \tilde{U}} A_i \tag{5.5}$$

and notice that $s_{ij} = 0$ implies that $a_{ij} = 0$ for some $A_{\tilde{u}_l}$. With this, there exists at least one input $\tilde{u} \in \tilde{U}$ that is preventing for $\tilde{x}_1 \to \tilde{x}_2$ if and only if $\hat{x}_2^T S_{\tilde{U}} \hat{x}_1 = 0$. Furthermore, $\hat{x}_2 = S_{\tilde{U}} \hat{x}_1$ contains all neighboring states of $\tilde{x}_1$ that *cannot be avoided by any input* of the set $\tilde{U}$.

5.2.2 Correcting inputs

Let the system (5.1) be given, together with a set of boundaries (5.2), and a finite set of inputs $\tilde{U}$.

Definition 5.2.4 (Correcting input) *Given $\tilde{x}_1$ and $\tilde{x}_2$ as in Condition 3.2.5. An input $\tilde{u} \in \tilde{U}$ is correcting for the transition $\tilde{x}_1 \to \tilde{x}_2$ if*

$$f^r(x,u) < 0, \forall x \in H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2),\ \forall u \in H_u(\tilde{u}). \tag{5.6}$$

This means that $\tilde{u}$ can be used to correct the changing of the continuous state from $H_x(\tilde{x}_1)$ to $H_x(\tilde{x}_2)$.

Proposition 5.2.5 *If input $\tilde{u} \in \tilde{U}$ is* correcting *for the transition $\tilde{x}_1 \to \tilde{x}_2$, then there exists an open set Ω, with $int(H_x(\tilde{x}_1) \cup H_x(\tilde{x}_2)) \supseteq \Omega \supseteq int(H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2))$ with the following property. For any state trajectory ξ corresponding to an input v with $v(t) \in H_u(\tilde{u})$ for all $t \geq 0$, and initial condition $\xi(0) = x_0 \in \Omega$ there exists a $t_1 \geq 0$ such that $\xi(t) \in \Omega$ for $0 \leq t \leq t_1$, and $\xi(t_1) \in int(H_x(\tilde{x}_1))$.*

Proof. Pick any point $x_0 \in int(H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2))$ and take $\delta > 0$ such that

- $B(x_0, \delta) := \{x \mid \|x - x_0\| < \delta\} \subseteq int(H_x(\tilde{x}_1) \cup H_x(\tilde{x}_2))$, and;
- there exists an $\varepsilon > 0$ with $f^r(x,u) < -\varepsilon$ for all $x \in B(x_0, \delta)$ and all $u \in H_u(\tilde{u})$.

To see that in particular the last property can be obtained, consider the function $x \mapsto g(x) = \max_{u \in H_u(\tilde{u})} f^r(x,u)$. Since $H_u(\tilde{u})$ is compact, the maximum is attained for some $u_x^* \in H_u(\tilde{u})$. Hence, for all $H_u(\tilde{u})$ we have that $g(x_0) = f^r(x_0, u_{x_0}^*) < 0$ by the hypothesis of the theorem. Since f is continuous, the function g is continuous as well. Hence, there exists an $\varepsilon > 0$ and a

$\delta > 0$ such that for all $x \in B(x_0, \delta)$ it holds that $g(x) \leq -\varepsilon$, which establishes the result mentioned above.

For a trajectory ξ of the system starting in $\xi(0) \in B(x_0, \delta)$ we denote the maximal interval that the trajectory remains in $B(x_0, \delta)$ by (t^1_ξ, t^2_ξ). Formally,

$$t^1_\xi := \sup\{t < 0 \mid \xi(t) \notin B(x_0, \delta)\} \tag{5.7a}$$

$$t^2_\xi := \inf\{t > 0 \mid \xi(t) \notin B(x_0, \delta)\}. \tag{5.7b}$$

Note that both extremes exist and are finite when ξ is a trajectory corresponding to an input v with $v(t) \in H_u(\tilde{u})$ for all t. Indeed, in this case we have inside $B(x_0, \delta)$ that $\dot{\xi}^r(t) = f^r(\xi(t), v(t)) \leq -\varepsilon$. As a consequence, $t^1_\xi \leq \frac{2\delta}{\varepsilon}$ and $t^2_\xi \geq -\frac{2\delta}{\varepsilon}$.

To continue, we take $0 < \delta' < \delta$ and consider $\Omega'_{x_0} := H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2) \cap B(x_0, \delta')$. The set Ω_{x_0} is now defined as the set of all points inside $B(x_0, \delta)$ that can be reached in (either forward or backward time) starting in Ω'_{x_0} with input values lying in $H_u(\tilde{u}$. If the state trajectory corresponding to an input v and initial condition z_0 at time 0 is denoted by $\xi_{z_0,v}$, we can formally define this set as

$$\Omega_{x_0} := \left\{ z \in B(x_0, \delta) \;\middle|\; \begin{array}{l} \exists z_0 \in \Omega'_{x_0} \exists v \text{ such that } \xi_{z_0,v}(t) = z \text{ for some } t, \\ t \in (t^1_{\xi_{x_0,v}}, t^2_{\xi_{x_0,v}}) \text{ and } v(t) \in H_u(\tilde{u}) \text{ for all } t \end{array} \right\}$$

As Ω_{x_0} is open, the union $\Omega := \bigcup_{x_0} \Omega_{x_0}$ is open as well. Since Ω_{x_0} consists of the collection of solutions trajectories crossing the boundary $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2) \cap B(x_0, \delta')$ (note that $f^r(x, u) \leq -\varepsilon$) and lying in $B(x_0, \delta) \subset int(H_x(\tilde{x}_1) \cup H_x(\tilde{x}_2))$, the proposition follows by construction. ■

In words this proposition states that when the transition $\tilde{x}_1$ to $\tilde{x}_2$ has occurred, then we can 'correct' the transition as long as the state $\xi(t)$ is in the open set Ω. Indeed, choosing the discrete input $\tilde{u}$ will steer the state $\xi(t) \in H_x(\tilde{x}_2)$ back to $H_x(\tilde{x}_1)$ (without triggering another transition first).

Note that the proposition for a correcting input differs from the proposition for a preventing input by the strictness of the inequality in (5.4) and (5.6). The strictness of the inequality together with the continuity of $f(x, u)$ allows us to correct a transition: if $f^r(x, u) < 0$ at the boundary $x^r = \beta^r_j$ between two adjacent hypercubes $H_x(\tilde{x}_1)$, and $H_x(\tilde{x}_2)$, then due to continuity $f^r(x, u) < 0$ close to the boundary as well. As long as the continuous state has not left Ω a correcting input $\tilde{u}$ can be used to correct the transition.

Next, inequality (5.6) is used for constructing the Boolean matrix $K_{\tilde{u}} \in \{0, 1\}^{p \times p}$ for a discrete input $\tilde{u}$:

$$(k_{\tilde{u}})_{ij} = \begin{cases} 0 & j \to i \text{ can be corrected} \\ 1 & \text{else.} \end{cases}$$

Notice that $\hat{x}_2 = K_{\tilde{u}}\hat{x}_1$ represents the set of discrete states for which a transition from $\tilde{x}_1$ cannot be corrected. Clearly, the input $\tilde{u}$ is correcting for the transition $\tilde{x}_1 \to \tilde{x}_2$ if and only if $\hat{x}_2^T K_{\tilde{u}}\hat{x}_1 = 0$. At first sight, it might be strange to construct a matrix from which the transitions that cannot be corrected are deduced, however notice that $K_{\tilde{u}}$ is comparable with the matrix $S_{\tilde{u}}$.

The computation of the matrices $K_{\tilde{u}}$ involves elaborate optimizations for deciding whether the sign of the r-th coordinate derivative $f^r(x,u)$ is strictly negative or positive for all (continuous) states on the boundary between two hypercubes. It would be convenient if from the information provided by the discrete-event model the inequality (5.6) could be checked immediately (like for preventing inputs). A necessary condition for a transition $\tilde{x}_1 \to \tilde{x}_2$ to be correctable in terms of the transition function ϕ (or equivalently, of the adjacency matrices $\{A_u\}_{u\in\tilde{U}}$) is given by the following proposition.

PROPOSITION 5.2.6 (NECESSARY COND. FOR A CORRECTING INPUT) *For a transition $\tilde{x}_1 \to \tilde{x}_2$ to be correctable with the input $\tilde{u} \in \tilde{U}$ it is necessary that*

1. *The transition $\tilde{x}_1 \to \tilde{x}_2$ is not possible with input $\tilde{u}$, i.e. $\hat{x}_2^T A_{\tilde{u}}\hat{x}_1 = 0$.*
2. *The transition $\tilde{x}_2 \to \tilde{x}_1$ is possible with input $\tilde{u}$, i.e. $\hat{x}_1^T A_{\tilde{u}}\hat{x}_2 = 1$.*

That this condition is only necessary and not sufficient follows from the construction of our discrete-event model (Chapter 3), where it can be seen that for checking the possibility of a transition the *existence* of a strict inequality is checked (see Theorem 3.2.6). So, if according to the resulting discrete-event model the transition $\tilde{x}_1 \to \tilde{x}_2$ is not possible and the transition $\tilde{x}_2 \to \tilde{x}_1$ is possible, then still there might exists points $x \in H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$ for which the derivative in the particular direction is zero, that is $f^r(x,u) = 0$ such that strictness of the inequality for points on the boundary can not be guaranteed, see Figure 5.2.

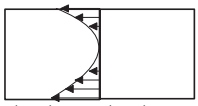

Figure 5.2: *$\tilde{u}$ is not correcting for the transition $\tilde{x}_1 \to \tilde{x}_2$*

It would be convenient if the condition stated in Proposition 5.2.6 also would be sufficient. This is the case if the following assumption is satisfied.

ASSUMPTION 5.2.7 *If for two adjacent states $\tilde{x}_1$, $\tilde{x}_2$ and discrete input $\tilde{u}$ there exists an $x_0 \in int(H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2))$ with $f^r(x_0, u) = 0$ for some $u \in H_u(\tilde{u})$ then one of the following two conditions should be satisfied*

1. *$\exists x_1, x_2 \in int\,(H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2))$ and $\exists u_1, u_2 \in H_u(\tilde{u})$ for which it holds that $f^r(x_1, u_1) > 0$ and $f^r(x_2, u_2) < 0$,*
2. *$f^r(x, u) = 0,\ \forall x \in int\,(H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2))$ and $\forall u \in H_u(\tilde{u})$.*

In this way it is certain that if on the boundary between two hypercubes the derivative $f^r(x, u) = 0$ for some point x and some u (*i.e.* the inequality sign is non-strict) then there must exist both positive and negative derivatives on the boundary, or $f^r(x, u) = 0$ for all points on the boundary and all inputs $u \in H_u(\tilde{u})$. Both situations are reflected in the transition function of the discrete-event model; in the first case, both $\tilde{x}_1 \to \tilde{x}_2$ and $\tilde{x}_2 \to \tilde{x}_1$ are possible, in the second case none of the transitions is possible. One important class of systems for which this condition is satisfied, is the class of linear systems. In Section 3.3.4 it is shown that for linear systems $\dot{x}^r = 0$ defines a $((n+m-1)$-dimensional) hyperplane in the state-input space $\mathbb{R}^{n+m}$ separating two half-spaces: one for which the derivative of the r-th coordinate is positive and one for which it is negative.

If this assumption holds, then the adjacency matrix $A_{\tilde{u}}$ and the neighbor matrix N (see Chapter 2) can be used for the computation of $K_{\tilde{u}}$. First recall that for a discrete state $\tilde{x}_1$ the Boolean vector $\hat{x}_2 := N\hat{x}_1$ represents the set of all discrete states for which the tuple representation differs only one unit in each coordinate from the tuple representation of $\tilde{x}_1$. In the way our discrete states are related to hypercubes in the state space, the set $\hat{X}_2$ contains *all* hypercubes adjacent to the hypercube $H_x(\tilde{x}_1)$, *i.e.* all 'neighbors' of $\tilde{x}_1$. To these states transitions from $\tilde{x}_1$ are allowed, but not necessarily possible according the discrete-event model. Given a discrete state $\tilde{x}_1$ and a discrete input $\tilde{u}$ we know that $\hat{x}_3 := A_{\tilde{u}}\hat{x}_1$ is the set of discrete states to which transitions are possible with the input $\tilde{u}$ according our model. Since $\hat{x}_2 := N\hat{x}_1$ are all adjacent discrete states to $\tilde{x}_1$, the set difference $\hat{x}_4 := \hat{x}_2 \ominus \hat{x}_3$ are all adjacent discrete states of $\tilde{x}_1$ for which a transitions is not possible with the input $\tilde{u}$ according to the discrete-event model. This set can be computed in one step by $\hat{x}_4 = (N \ominus A)\hat{x}_1$.

For computing $K_{\tilde{u}}$, recall that $\hat{x}_2 := K_{\tilde{u}}\hat{x}_1$ is the set of discrete states for which a transition from $\tilde{x}_1$ cannot be corrected. Since Proposition 5.2.6 together with Assumption 5.2.7 is necessary and sufficient, it follows that an input is *not* correcting for a transition $\tilde{x}_1 \to \tilde{x}_2$ if and only if it allows the transition $\tilde{x}_1 \to \tilde{x}_2$ or if the transition $\tilde{x}_2 \to \tilde{x}_1$ is not possible. Given $\tilde{x}_1$ the set of states to which a transition is possible with input $\tilde{u}$ is given by $\hat{x}_2 := A_{\tilde{u}}\hat{x}_1$. Furthermore, the set of states from which a transition to $\tilde{x}_1$ is

possible with input $\tilde{u}$ is given by $\hat{x}_3 := A_{\tilde{u}}^T \hat{x}_1$, so the set of states from which a transition to $\tilde{x}_1$ is not possible is given by $\hat{x}_4 := (N \ominus A_{\tilde{u}}^T)\hat{x}_1$. Combining both yields the set of states for which a transition *cannot* be corrected with input $\tilde{u}$, $\hat{x}_3 \oplus \hat{x}_4 = ((N \ominus A_{\tilde{u}}^T) \oplus A_{\tilde{u}})\hat{x}_1$. Hence the Boolean matrix $K_{\tilde{u}}$ is given by

$$K_{\tilde{u}} := ((N \ominus A_{\tilde{u}}^T) \oplus A_{\tilde{u}}).$$

5.2.3 Moving inputs

From the description of moving inputs in Section 5.1 it is clear that an input is moving for a transition $\tilde{x}_1 \to \tilde{x}_2$ if with this input for all trajectories ξ starting in $H_x(\tilde{x}_1)$ the distance to the boundary plane $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$ is decreased when ξ leaves $H_x(\tilde{x}_1)$. From Figure 5.1 (c) it can be seen that the trajectory starting in x_0'' is moving according to this description. However, to facilitate computations (specifically, to avoid the necessity to integrate) the definition of moving inputs is strengthened.

Let the system (5.1) be given, together with a set of boundaries (5.2), and a finite set of inputs $\tilde{U}$.

Definition 5.2.8 (Moving input) *Given $\tilde{x}_1$ and $\tilde{x}_2$ as in Condition 3.2.5. An input $\tilde{u} \in \tilde{U}$ is* moving *for the transition $\tilde{x}_1 \to \tilde{x}_2$ if*

$$f^r(x,u) > 0, \ \forall x \in H_x(\tilde{x}_1), \ \forall u \in H_u(\tilde{u}). \tag{5.8}$$

This means that the r-th coordinate of x increases as $f^r(x,u) > 0$ and therefore moves towards the boundary between $H_x(\tilde{x}_1)$ and $H_x(\tilde{x}_2)$.

With this new definition the trajectory starting in x_0'' is no longer moving, so the class of moving inputs is contracted. We opt for using Definition 5.2.8 because such a property can be verified automatically.

Fix input $\tilde{u} \in \tilde{U}$. All transitions for which $\tilde{u}$ is a moving input are represented by the *direction matrix* $D_{\tilde{u}} \in \{0,1\}^{p \times p}$ given by

$$(d_{\tilde{u}})_{ij} = \begin{cases} 1 & \tilde{u} \text{ is moving for } j \to i \\ 0 & \text{else.} \end{cases}$$

The computation of $D_{\tilde{u}}$ can be automated by using optimizations for checking (5.8). Using $\lambda_{\min} := \min_{x \in H_x(j)} f^r(x,u)$ and $\lambda_{\max} := \max_{x \in H_x(j)} f^r(x,u)$ we have that $f^r(x,u) > 0$, $\forall x \in H_x(\tilde{x}_1) \iff \lambda_{\min} > 0$ and $f^r(x,u) < 0$, $\forall x \in H_x(\tilde{x}_1) \iff \lambda_{\max} < 0$. Note that $(d_{\tilde{u}})_{ij} = 1 \implies (d_{\tilde{u}})_{ji} = 0$. Furthermore, for linear systems the same techniques can be used as for the computation of the transition function as discussed in Section 3.3.4. In fact, for linear systems all information needed can be extracted from the adjacency matrices $\{A_{\tilde{u}_i}\}$.

Note, that $\hat{x}_2 := D_{\tilde{u}}\hat{x}_1$ represents the set of all discrete states adjacent to $\tilde{x}_1$ for which the input $\tilde{u}$ is moving. With this, it is clear that the input $\tilde{u}$ is moving for the transition $\tilde{x}_1 \rightarrow \tilde{x}_2$ if and only if $\hat{x}_2^T D_{\tilde{u}}\hat{x}_1 \neq 0$, since this is equivalent to $\hat{x}_2 \otimes D_{\tilde{u}}\hat{x}_1 \neq 0$.

If $\hat{x}_2$ represents the set $\tilde{X}_2$ then $\hat{x}_2^T D_{\tilde{u}}\hat{x}_1 \neq 0$ implies that there exists $\tilde{x}_2' \in \tilde{X}_2$ such that the input $\tilde{u}$ is moving for $\tilde{x}_1 \rightarrow \tilde{x}_2'$. In this case we say that $\tilde{u}$ is moving for the set $\tilde{X}_2$ from $\tilde{x}_1$.

Furthermore, for a set of inputs $\tilde{U}$ the direction matrix D is given by

$$D = \bigoplus_{\tilde{u}\in\tilde{U}} D_{\tilde{u}}.$$

Obviously, there exists an input $\tilde{u} \in \tilde{U}$ which is moving for the transition $\tilde{x}_1 \rightarrow \tilde{x}_2$ if and only if $\hat{x}_2^T D\hat{x}_1 \neq 0$.

These three types of inputs form the basis for the controller design methods that will be proposed in the next section. To use the preventing inputs, some assumptions have to be made on the sensors and the control action. Therefore, first the three proposed controller design methods are explained with the use of preventing and moving inputs under the following assumption.

Assumption 5.2.9 *It is measured whenever the continuous state trajectory ξ reaches a boundary plane $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$. When this is observed, immediate control action is possible.*

In Section 5.7, it is discussed how to adapt the design method for the situation where the aforementioned assumption is relaxed in which case the correcting inputs will replace the preventing inputs.

5.3 Discretely controlled invariant sets

Since the proposed controller design methods depend heavily on the construction of discretely controlled invariant sets, this notion is formalized first.

Definition 5.3.1 (Discretely controlled invariant set) *The set $\Omega := \bigcup_{\tilde{x}\in\tilde{Z}} H_x(\tilde{x})$ (and the corresponding set of discrete states $\tilde{Z}$) is called a discretely controlled invariant set, if for each initial state $x_0 \in \Omega$, a discretely controlled trajectory ξ (with $\xi(0) = x_0$) exists, such that $\xi(t) \in \Omega$ for all $t \geq 0$.*

Proposition 5.3.2 *The set $\tilde{Z} \subseteq \tilde{X}$ is discretely controlled invariant for the set of inputs $\tilde{V} \subseteq \tilde{U}$ iff (in Boolean vector notation) $S_{\tilde{V}}\hat{z} \ominus \hat{z} = 0$, where $S_{\tilde{V}} = \bigotimes_{i\in\tilde{V}} A_i$.*

Proof. Recall that $\tilde{z}' := S_{\tilde{V}}\tilde{z}$ represents the set of states to which transitions from states in the set $\tilde{Z}$ cannot be avoided. If $\tilde{Z}'$ is a subset of $\tilde{Z}$ or equivalently $\hat{z}' \ominus \hat{z} = 0$), then transitions to states outside the set $\tilde{Z}$ can be prevented with inputs from the set $\tilde{V}$, hence any trajectory starting in $\bigcup_{\tilde{x}\in\tilde{Z}} H_x(\tilde{x})$ can be kept from leaving this set. If $S_{\tilde{V}}\hat{z} \ominus \hat{z} \neq 0$, then clearly transitions to states outside $\tilde{Z}$ cannot be prevented, hence $\tilde{Z}$ is not discretely controlled invariant. ■

DEFINITION 5.3.3 *Given the set $\tilde{Z}$, then $\tilde{X}_{inv}$ is the largest discretely controlled invariant set in $\tilde{Z}$ if*

i) $\tilde{X}_{inv} \subseteq \tilde{Z}$,

ii) $\tilde{X}_{inv}$ *is discretely controlled invariant,*

iii) *If $\tilde{X}'_{inv}$ satisfies i) and ii) then $\tilde{X}'_{inv} \subseteq \tilde{X}_{inv}$.*

PROPOSITION 5.3.4 *Given a set $\tilde{Z} \subseteq \tilde{X}$, the largest discretely controlled invariant set $\tilde{X}_{inv}$ for the set of inputs $\tilde{V}$ which is contained in $\tilde{Z}$ is computed from the sequence of sets given by $\tilde{Z} = \tilde{X}_0 \supset \tilde{X}_1 \supset \tilde{X}_2 \supset ... \supset \tilde{X}_j$, with*

1. $\hat{x}_0 := \hat{z}$,
2. $\hat{x}_{k+1} = \hat{x}_k \ominus S^T_{\tilde{V}}(S_{\tilde{V}}\hat{x}_k \ominus \hat{x}_k)$.

When $\hat{x}_{j+1} = \hat{x}_j$ then $\hat{x}_{inv} := \hat{x}_j$.

Proof. i) Note that from step 2 we have that $\tilde{X}_{k+1} \subseteq \tilde{X}_k$, since only states are removed from $\tilde{X}_k$ to obtain $\tilde{X}_{k+1}$. Hence $\tilde{X}_k \subseteq \tilde{Z}$ and i) is satisfied.

ii) $\tilde{X}_k = \tilde{X}_{k+1}$ is equivalent to $\hat{x}_k = \hat{x}_k \ominus S^T_{\tilde{V}}(S_{\tilde{V}}\hat{x}_k \ominus \hat{x}_k)$ implying that $0 = \hat{x}_k^T S^T_{\tilde{V}}(S_{\tilde{V}}\hat{x}_k \ominus \hat{x}_k) = (S_{\tilde{V}}\hat{x}_k)^T(S_{\tilde{V}}\hat{x}_k \ominus \hat{x}_k)$. Since the set represented by $S_{\tilde{V}}\hat{x}_k \ominus \hat{x}_k$ is a subset of the set represented by $S_{\tilde{V}}\hat{x}_k$ it holds that $(S_{\tilde{V}}\hat{x}_k)^T(S_{\tilde{V}}\hat{x}_k \ominus \hat{x}_k) = 0 \Longleftrightarrow S_{\tilde{V}}\hat{x}_k \ominus \hat{x}_k = 0$ thus proving that $\tilde{X}_k$ is controlled invariant.

To prove iii) we claim that for any set $\tilde{Z}' \subseteq \tilde{Z}$ it holds that $\tilde{X}'_k \subseteq \tilde{X}_k$ where $\{\tilde{X}'_k\}$ is the sequence of sets generated by the algorithm as given in the proposition with $\tilde{X}'_0 := \tilde{Z}'$. Starting with $\tilde{X}'_0 := \tilde{Z}'$ and $\tilde{X}_0 := \tilde{Z}$ we have that $\tilde{X}'_0 \subseteq \tilde{X}_0$, so the claim holds for $k = 0$. We proceed by induction, *i.e.* suppose $\tilde{X}'_k \subseteq \tilde{X}_k$ is true. Now we will prove that if $\tilde{x} \in \tilde{X}'_k \cap \tilde{X}_k$ is removed from $\tilde{X}_k$ on the basis of the algorithm, then it is also removed from $\tilde{X}'_k$ implying that $\tilde{X}'_{k+1} \subseteq \tilde{X}_{k+1}$. Suppose that $\tilde{x} \in \tilde{X}'_k \cap \tilde{X}_k$ has to be removed from $\tilde{X}_k$, then this implies that $\hat{x}^T S^T_{\tilde{V}}(S_{\tilde{V}}\hat{x} \ominus \hat{x}_k) = 1$. Recall that $S_{\tilde{V}}\hat{x} \ominus \hat{x}_k$ represents the set of states to which transitions cannot be prevented from $\tilde{x}$ that are not in $\tilde{X}_k$, and $S^T_{\tilde{V}}(S_{\tilde{V}}\hat{x} \ominus \hat{x}_k)$ represents the set of states (from which $\tilde{x}$ is

an element) that lead to those states. Since $\tilde{X}'_k \subseteq \tilde{X}_k$, the set represented by $S_{\tilde{V}}\hat{x} \ominus \hat{x}_k$ is a subset of the set represented by $S_{\tilde{V}}\hat{x} \ominus \hat{x}'_k$. Consequently, $\hat{x}^T S^T_{\tilde{V}}(S_{\tilde{V}}\hat{x} \ominus \hat{x}'_k) = 1$. Since $\tilde{x} \in \tilde{X}'_k$ it is certain that $\tilde{x}$ will be removed from $\tilde{X}'_k$. Hence, $\tilde{X}'_{k+1} \subseteq \tilde{X}_{k+1}$. To prove iii) take an arbitrary $\tilde{X}'_{inv}$ satisfying i) and ii). Set $\tilde{Z}'$ in the claim above equal to $\tilde{X}'_{inv}$ and observe that controlled invariance yields that $\tilde{X}'_k = \tilde{X}'_{inv}$ for all k. Since $\tilde{X}'_{inv} \subseteq \tilde{X}_k$ for all k the result follows as $\tilde{X}_k$ is equal to $\tilde{X}_{inv}$ for sufficient large k. ■

Starting with the set $\tilde{X}_0 = \tilde{Z}$, all discrete states are removed from $\tilde{X}_0$ that cause unpreventable transitions to states outside $\tilde{X}_0$. From $\tilde{X}_0$ unpreventable transitions occur to states outside $\tilde{X}_0$, represented by $\hat{x}' := S_{\tilde{V}}\hat{x}_0 \ominus \hat{x}_0$. The discrete states in $\tilde{X}_0$ from which these transitions happen are given by $\hat{x}'' := S^T_{\tilde{V}}\hat{x}' \otimes \hat{x}_0$. Removing these states from $\tilde{X}_0$ yields $\hat{x}_1 := \hat{x}_0 \ominus (S^T_{\tilde{V}}\hat{x}' \otimes \hat{x}_0) = \hat{x}_0 \ominus S^T_{\tilde{V}}\hat{x}'$ (we can only extract states from $\tilde{X}_0$ that actually are elements of $\tilde{X}_0$). This is repeated until $\tilde{X}_k$ is controlled invariant (or equivalently, $\tilde{X}_k = \tilde{X}_{k-1}$). Since $\tilde{X}_0$ has a finite number of elements and only states are removed, the algorithm is finite.

DEFINITION 5.3.5 *Given the set $\tilde{Z}$, then $\tilde{X}_{inv}$ is the smallest discretely controlled invariant set containing $\tilde{Z}$, if*

i) $\tilde{Z} \subseteq \tilde{X}_{inv}$,

ii) $\tilde{X}_{inv}$ *is discretely controlled invariant,*

iii) *If $\tilde{X}'_{inv}$ satisfies i) and ii) then $\tilde{X}_{inv} \subseteq \tilde{X}'_{inv}$.*

PROPOSITION 5.3.6 *Given a set $\tilde{Z} \subseteq \tilde{X}$, the smallest discretely controlled invariant set $\tilde{X}_{inv}$ for the set of inputs $\tilde{V}$ containing $\tilde{Z}$ is computed from the sequence of sets given by $\tilde{Z} = \tilde{X}_0 \subset \tilde{X}_1 \subset \tilde{X}_2 \subset ... \subset \tilde{X}_j$, with*

1. $\hat{x}_0 := \hat{z}$,
2. $\hat{x}_{k+1} = \hat{x}_k \oplus S_{\tilde{V}}\hat{x}_k$.

When $\hat{x}_{j+1} = \hat{x}_j$ then $\hat{x}_{inv} := \hat{x}_j$.

Proof. i) Note that we have $\tilde{X}_k \subseteq \tilde{X}_{k+1}$ as only states are added to obtain $\tilde{X}_{k+1}$. Hence, $\tilde{Z} \subseteq \tilde{X}_k$ for all k and i) is satisfied.

ii) $\tilde{X}_k = \tilde{X}_{k+1}$ is equivalent to $\hat{x}_k = \hat{x}_k \oplus S_{\tilde{V}}\hat{x}_k \Longleftrightarrow S_{\tilde{V}}\hat{x}_k \ominus \hat{x}_k = 0$ (since $S_{\tilde{V}}\hat{x}_k$ must represent a subset of $\tilde{X}_k$), proving the controlled invariance of $\tilde{X}_k$.

iii) We claim that for any set $\tilde{Z}' \supseteq \tilde{Z}$ it holds that $\tilde{X}'_k \supseteq \tilde{X}_k$, where $\{\tilde{X}'_k\}$ is generated by the algorithm with $\tilde{X}'_0 = \tilde{Z}'$. Starting with $\tilde{X}'_0 := \tilde{Z}'$ and

$\tilde{X}_0 := \tilde{Z}$ we have that $\tilde{X}'_0 \supseteq \tilde{X}_0$, proving that the claim holds for $k = 0$. To proceed by induction, assume that $\tilde{X}'_k \supseteq \tilde{X}_k$. Correspondingly we can write $\hat{x}'_k = \hat{x}_k \oplus \Delta\hat{x}'_k$. With this, it follows that $\hat{x}_{k+1} = \hat{x}_k \oplus S_{\tilde{V}}\hat{x}_k$ and $\hat{x}'_{k+1} = \hat{x}_k \oplus \Delta\hat{x}'_k \oplus S_{\tilde{V}}(\hat{x}_k \oplus \Delta\hat{x}'_k) = \hat{x}_k \oplus \Delta\hat{x}'_k \oplus S_{\tilde{V}}\hat{x}_k \oplus S_{\tilde{V}}\Delta\hat{x}'_k$. Hence, $\tilde{X}'_{k+1} \supseteq \tilde{X}_{k+1}$. Similar reasoning as for the proof of iii) for Proposition 5.3.4 gives the desired result ■

Starting with the set $\tilde{X}_0 = \tilde{Z}$, all discrete states are added to $\tilde{X}_0$ to which transitions from states in $\tilde{X}_0$ cannot be prevented (represented by $\hat{x}' := S_{\tilde{V}}\hat{x}_0$). This is repeated until $\tilde{X}_k$ is discretely controlled invariant. Since only states are added to which transitions cannot be avoided, the smallest discretely controlled invariant set containing $\tilde{Z}$ results. Since, starting from $\tilde{X}_0$, only states are added and the total number of discrete states in $\tilde{X}$ is finite, the algorithm also is finite.

For the sake of the reachability problem it is convenient to have an extension of the concept of controlled invariant set.

DEFINITION 5.3.7 (Γ-CONTROLLED INV. SET) *Let the sets $\Omega := \bigcup_{\tilde{x}\in\tilde{Z}} H_x(\tilde{x})$ and $\Gamma := \bigcup_{\tilde{x}\in\tilde{Z}_1} H_x(\tilde{x})$ be given. Ω is called Γ-controlled invariant (and $\tilde{Z}$ is $\tilde{Z}_1$-controlled invariant), if for each initial state $x_0 \in \Omega$, a discretely controlled trajectory ξ (with $\xi(0) = x_0$) exists, such that either $\xi(t) \in \Omega$ for all $t \geq 0$, or $\xi(t_e) \in \Gamma$, where $t_e = \inf\{t > 0 \mid \xi(t) \notin \Omega\}$.*

Loosely speaking, Ω is Γ-controlled invariant, if we leave the set via Γ in case the state cannot remain in Ω. Note that discretely controlled invariance is equivalent to $\emptyset$-controlled invariance.

PROPOSITION 5.3.8 *$\tilde{Z} \subseteq \tilde{X}$ is $\tilde{Z}_1$-controlled invariant for the set of inputs $\tilde{V} \subseteq \tilde{U}$ iff (in Boolean vector notation) $S_{\tilde{V}}(\hat{z} \ominus \hat{z}_1) \ominus \hat{z} = 0$, where $S_{\tilde{V}} = \bigotimes_{\tilde{u}\in\tilde{V}} A_{\tilde{u}}$.*

Proof. From all the discrete states in $\tilde{Z}$ that are not in $\tilde{Z}_1$ (*i.e.* $\tilde{Z} \setminus \tilde{Z}_1$ represented by $\hat{z} \ominus \hat{z}_1$) transitions cannot be prevented to the set $\hat{z}' = S_{\tilde{V}}(\hat{z} \ominus \hat{z}_1)$. If $\hat{z}' \ominus \hat{z} = 0$, then $\tilde{Z}' \subseteq \tilde{Z}$ and we can only leave $\tilde{Z}$ via $\tilde{Z}_1$. If $S_{\tilde{V}}(\hat{z} \ominus \hat{z}_1) \ominus \hat{z} \neq 0$, then clearly transitions to states outside $\tilde{Z}$ cannot be prevented from a discrete state which is in $\tilde{Z}$ but not in $\tilde{Z}_1$. ■

5.4 'Forceable state-transition' control strategy

The first controller design method, that we propose, is most easily explained for the reachability problem, although it is also applicable to the stabilizing problem with some minor extensions. Recall that Assumption 5.2.9 holds.

From the transition function ϕ it can be seen that a transition from one state to another is possible with a particular input. However, due to non-determinism, often also other transitions are possible with this input. As a consequence, it is not known which of the possible transitions actually will happen, since this depends on the unknown continuous state. The idea of the first controller synthesis is to find a way to guarantee that a transition from one discrete state to another is possible and all other transitions can be ruled out. To do so, we introduce the following concept.

DEFINITION 5.4.1 (FORCEABLE TRANSITION) *Given two (adjacent) discrete states $\tilde{x}_1$, $\tilde{x}_2$ the transition $\tilde{x}_1 \to \tilde{x}_2$ is called* forceable*, if from each initial state $x_0 \in H_x(\tilde{x}_1)$ there exists a discretely controlled trajectory ξ (with $\xi(0) = x_0$) for which there exists a $t_1 > 0$ such that $\xi(t) \in H_x(\tilde{x}_1) \cup H_x(\tilde{x}_2)$ for all $t \in [0, t_1]$ and $\xi(t_1) \in int(H_x(\tilde{x}_2))$.*

The forceable transitions are defining a directed graph on the set of discrete states that we call the *forceability graph.* It is obvious that the reachability problem is easily solvable if the forceability graph is known:

PROPOSITION 5.4.2 *The reachability problem is solvable for $\tilde{x}_0$ to $\tilde{x}_e$, if the forceability graph contains a path from $\tilde{x}_0$ to $\tilde{x}_e$.*

A solution to the reachability problem can be found by constructing the forceability graph. Unfortunately, it is very difficult in general to check whether a given transition is forceable. In the next section we give a sufficient condition for forceability and we show how it can be systematically checked using the continuous model. Because our condition is not necessary we are only able to construct a subgraph of the forceability graph. Hence, if there is no path from the initial discrete state to the target discrete state in this subgraph the reachability problem may still have a solution. However, if such a path exists then it is certain that the problem is solvable and a control strategy can be constructed that realizes the desired path.

5.4.1 Sufficient condition for the forceability of a transition

Given the system (5.1), together with a set of boundaries (5.2), and a finite set of inputs $\tilde{U}$, let $\tilde{u} \in \tilde{U}$ be fixed and let $\tilde{x}_1$ and $\tilde{x}_2$ be two adjacent states.

Using the concept of moving and preventing inputs, we can formulate the following proposition.

PROPOSITION 5.4.3 *The transition $\tilde{x}_1 \to \tilde{x}_2$ is* forceable*, if for each transition $\tilde{x}_1 \to \tilde{x}_3 \neq \tilde{x}_2$ to a state $\tilde{x}_3$ adjacent to $\tilde{x}_1$, there exists an input that is preventing for $\tilde{x}_1 \to \tilde{x}_3$ among the inputs that are moving for $\tilde{x}_1 \to \tilde{x}_2$.*

Proof. Suppose that the set of inputs that are moving for the transition $\tilde{x}_1 \rightarrow \tilde{x}_2$ is given by $\tilde{U}_m$. If we pick any $\tilde{u}_1 \in \tilde{U}_m$ it is guaranteed that the continuous state $x(t)$ moves towards the common boundary $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$. The only phenomenon that might obstruct $x(t)$ from crossing the boundary $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$, is that another boundary $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_3)$, $\tilde{x}_3 \neq \tilde{x}_2$ is reached first. In this case the transition $\tilde{x}_1 \rightarrow \tilde{x}_3$ will be prevented by an input $\tilde{u}_2$. If we choose $\tilde{u}_2$ from the set $\tilde{U}_m$, then still it is guaranteed that $x(t)$ is moving towards the desired boundary. By construction, it is certain that the transition $\tilde{x}_1 \rightarrow \tilde{x}_2$ will occur, and all other transitions $\tilde{x}_1 \rightarrow \tilde{x}_3$, $\tilde{x}_3 \neq \tilde{x}_2$, will not (and thus $\tilde{x}_1 \rightarrow \tilde{x}_2$ is forceable). ■

Since the condition in Proposition 5.4.3 is only sufficient it is used to construct a subgraph of the forceability graph, as will be explained in the next section.

5.4.2 Forceability graph

Now we present a practical method to determine the subgraph of the forceability graph based upon Proposition 5.4.3. Given a set of discrete inputs $\tilde{U} = \{\tilde{u}_i\}$, the corresponding transition matrices $\{A_{\tilde{u}_i}\}$ and the direction matrices $\{D_{\tilde{u}_i}\}$ the conditions of Proposition 5.4.3 guaranteeing the transition $\tilde{x}_1 \rightarrow \tilde{x}_2$ to be forceable can be checked by performing the following steps:

1. Determine all inputs for which $\tilde{x}_1$ is moving to $\tilde{x}_1 \rightarrow \tilde{x}_2$, *i.e.* compute

$$\tilde{U}_m = \{\tilde{u} \in \tilde{U} \mid \hat{x}_2^T D_{\tilde{u}} \hat{x}_1 \neq 0\}. \tag{5.9}$$

2. Check if, transitions to discrete states other than $\tilde{x}_2$ can be prevented by some input $\tilde{u} \in \tilde{U}_m$, *i.e.* verify that

$$S_{\tilde{U}_m} \hat{x}_1 \ominus \hat{x}_2 = 0,$$

with $S_{\tilde{U}_m} := \bigotimes_{\tilde{u} \in \tilde{U}_m} A_{\tilde{u}}$.

By computing this for all discrete states (that is, for all transitions $j \rightarrow i$), the forceability matrix P is constructed:

$$p_{ij} = \begin{cases} 1 & S_{\tilde{U}_m} \hat{x}_1 \ominus \hat{x}_2 = 0,\ \tilde{U}_m = \{\tilde{u} \in \tilde{U} \mid \hat{x}_2^T D_{\tilde{u}} \hat{x}_1 \neq 0\} \\ 0 & \text{else,} \end{cases}$$

where j and i are the integer presentations of $\tilde{x}_1$ and $\tilde{x}_2$, respectively.

The Boolean vector $\hat{x}_2 := P\hat{x}_1$ represents a set that contains discrete states to which a transition from $\tilde{x}_1$ is forceable. The forceability matrix P is the adjacency matrix of a subgraph of the forceability graph. If we label each

edge in the forceability graph with the set of inputs $\tilde{U}_m$ as defined in (5.9) then we have the *labelled forceability graph.* The labels indicate which inputs we can choose to prevent undesirable transitions and still guarantee movement towards a desired transition.

5.4.3 Control strategy

The reachability problem requires to discretely control the state from the initial discrete state $\tilde{x}_0$ to the target set of discrete states $\tilde{X}_e$ ($\tilde{x}_0 \notin \tilde{X}_e$). For this we have to search for a path in the forceability subgraph from $\tilde{x}_0$ to $\tilde{X}_e$. The following algorithm provides all these paths with minimal length (*i.e.* number of discrete states) by first searching backward (from $\tilde{X}_e$) and then forward (from $\tilde{x}_0$).

step 1 Set $\hat{x}_1 := \hat{x}_e$ and iterate

$$\hat{x}_{i+1} = P^T \hat{x}_i$$

until $\tilde{x}_0 \in \tilde{X}_{i+1}$, *i.e.* $\hat{x}_0^T \hat{x}_{i+1} = 1$. Set $r = i + 1$. Now we have computed all forceable paths of length $r - 1$ that lead to $\tilde{X}_e$, and at least one of these paths starts from $\tilde{x}_0$, see Figure 5.3 (a).

step 2 Set $\hat{z}_1 = \hat{x}_0$ and for $i = 1, ..., r - 1$ compute

$$\hat{z}_{i+1} = P\hat{z}_i \otimes \hat{x}_{r-i}.$$

So, the set $\tilde{Z}_i$ (represented by the Boolean vector $\hat{z}_i$) is the intersection of the forceable paths starting from $\tilde{x}_0$ of length $i-1$, see Figure 5.3 (b) and the forceable paths that are $r-i-1$ steps away from $\tilde{X}_e$, see Figure 5.3 (a). The resulting set $\{\tilde{Z}_1, \tilde{Z}_2, ..., \tilde{Z}_r\}$ contains the discrete states on forceable paths originating from $\tilde{x}_0$ and ending in $\tilde{X}_e$, see Figure 5.3 (c). It is always possible to go from $\tilde{Z}_i$ to $\tilde{Z}_{i+1}$ such that, starting from $\tilde{x}_0$ we reach $\tilde{X}_e$.

The actual control algorithm consists of choosing a moving input $\tilde{u} \in \tilde{U}_m$ from the labelled forceability graph such that a transition from $\tilde{x}_1 \in \tilde{Z}_i$ to $\tilde{x}_2 \in \tilde{Z}_{i+1}$ is forced until the target state is reached. Each time a sensor detects that the continuous state reaches a boundary $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_3)$ with $\tilde{x}_3 \notin \tilde{Z}_{i+1}$ the particular transition is prevented by choosing an input $\tilde{u} \in \tilde{U}_m$ such that $\tilde{x}_3 \notin \phi(\tilde{x}_1, \tilde{u})$, *i.e.* $\hat{x}_3^T A_{\tilde{u}} \hat{x}_1 = 0$. Whenever a sensor detects that the continuous state reaches a boundary $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$, $\tilde{x}_2 \in \tilde{Z}_{i+1}$, then the current input is maintained for τ time units to make sure that the boundary actually is crossed. After τ time units, an input is chosen that is moving for $\tilde{Z}_{i+2}$. However, in case a boundary of an undesired discrete state (*i.e.* a state not in $\tilde{Z}_{i+2}$) is

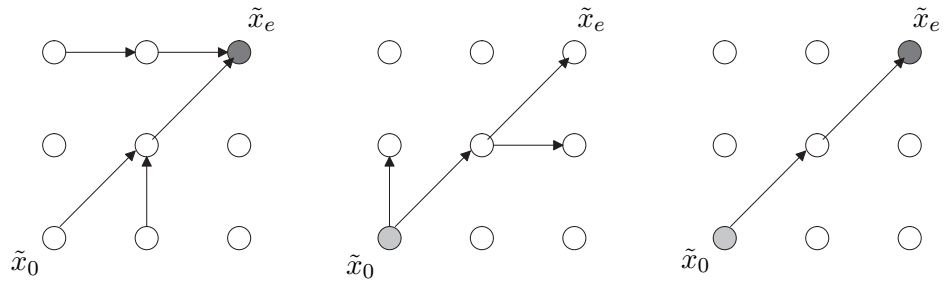

Figure 5.3: *(a) Forceable paths of length 2 ending in $\tilde{x}_e$, (b) forceable paths starting in $\tilde{x}_0$, and (c) forceable paths starting in $\tilde{x}_0$ and ending in $\tilde{x}_e$*

reached before τ time units have expired, then a preventing input is directly applied which is moving for $\tilde{Z}_{i+2}$ to realize a suitable continuation and the action with respect to 'τ' is cancelled. The parameter $\tau > 0$, can be chosen freely.

REMARK 5.4.4 *If a control action triggered by a time event (the elapse of a time period of τ units) is not included and the actions would be synchronous with the measurements, then two problems might occur. First, if we would wait until the next boundary is hit (i.e. $\tau = \infty$) for choosing a new preventing input, then in case of an equilibrium for the current input (which was moving for $\tilde{x}_1 \to \tilde{Z}_{i+1}$ but not necessarily for $\tilde{x}_2 \to \tilde{Z}_{i+2}$) in $\tilde{x}_2 \in \tilde{Z}_{i+1}$ the continuous state may not leave $\tilde{x}_2$ and no further improvement is possible. Second, if we would directly change the input after a boundary was reached (i.e. $\tau = 0$), then the new moving input could bring the trajectory back to $\tilde{x}_1$, because this input needs not to be preventing for the transition $\tilde{x}_2 \to \tilde{x}_1$.*

REMARK 5.4.5 *The usage of forceability subgraph is exploited for solving the reachability problem. To solve the stabilization problem one should solve the reachability problem for the largest discretely controlled invariant set contained in $\tilde{X}_e$. If this set is empty, then the stabilization problem cannot be solved. In this case, the best one can do is solving the reachability problem for the smallest discretely controlled invariant set containing $\tilde{X}_e$.*

5.4.4 Example: two tank system

Consider the two tank system depicted in Fig. 5.4.

It consists of two communicating tanks which are connected through a small pipe. Both tanks can be filled by controlling the valves in the respective feed lines (s_1, s_2). The pipe between the tubes can be blocked by means of the

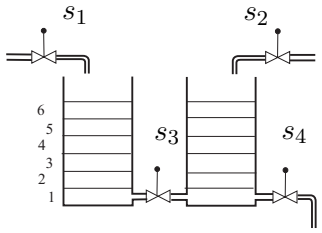

Figure 5.4: *The two tank system*

switch s_3. Only the second tank has a drain s_4. The input $\tilde{u} = (s_1, s_2, s_3, s_4)$ consists of the vector of switch positions (open or closed) for the four valves controlling the flows in the system. The input set has $2^4 = 16$ elements. The state vector $x = [x^1, x^2]^T$ is given by the water levels in each tank. The continuous model can easily be deduced from the three tank example in Section 3.5 by leaving out the equation for the second (middle) tank and renaming the elements of the state and input vector when necessary.

Each tank is divided into six parts (states) defined by the levels (β_j^i): 0, 0.01, 0.1, 0.2, 0.3, 0.4, and 0.5 $[m]$.

After computing the transition- and direction matrices we obtain the forceability graph as depicted in Figure 5.5, where the discrete states are represented by their tuple representation.

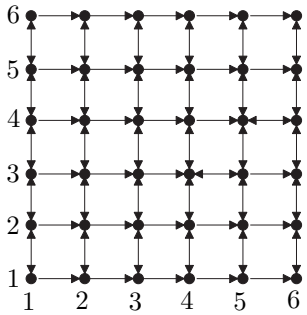

Figure 5.5: *Forceability graph for the two tank system*

A possible time trajectory from $\tilde{x}_0 = (6, 6)$ to $\tilde{x}_e = (4, 6)$ is shown in Figure 5.6.

For this example, τ was chosen equal to 0.1 [s]. From the forceability graph it can be seen that a possible path is given by the sequence $(6, 6) \rightarrow (6, 5) \rightarrow (6, 4) \rightarrow (5, 4) \rightarrow (5, 3) \rightarrow (4, 3) \rightarrow (4, 4) \rightarrow (4, 5) \rightarrow (4, 6)$. From Fig. 5.6 it can be seen that this indeed is the path the controller forces the system to follow. Furthermore, at the top of Figure 5.6 it is shown at which time instants the input changes. Starting with s_4 open and all other

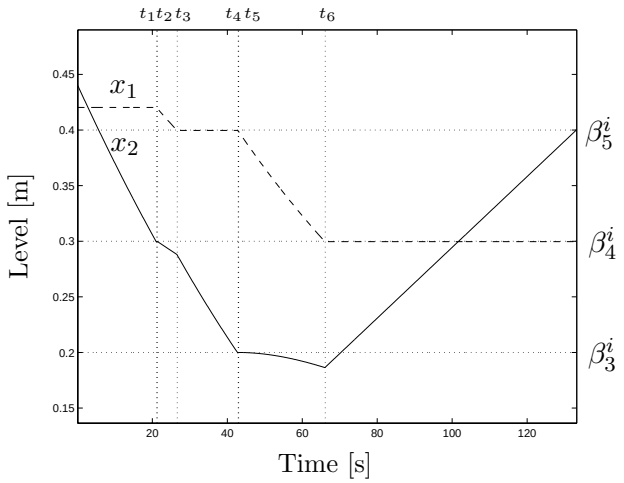

Figure 5.6: *Trajectory for* $\tilde{x}_0 = (6, 6)$ *and* $\tilde{x}_e = (4, 6)$

switches closed, after approximately 21 seconds the boundary β_5^2 is reached and $\tau = 0.1$ seconds later, at t_1 the controller opens s_3 (and closes s_4) in order to lower the level in the first tank. As a result, the level of the second tank starts to rise, which would cause x_2 to cross boundary β_4^2. This is prevented immediately by opening s_4 at time t_2. At t_3, discrete state $(5, 4)$ is reached and s_3 is closed (s_4 is still open) such that the level in the second tank can drop in order to reach $(5, 3)$. Then s_4 is switched off and s_3 is switched on at t_4, allowing the level in the first tank to drop again. However, since the level in the second tank starts to rise, the transition $(5, 3) \rightarrow (5, 4)$ has to be prevented by opening s_4 again at t_5. Finally, at t_6 the valves s_3 and s_4 are turned off and s_2 is turned on such that the level in the second tank rises until the desired discrete state is reached. In summary, the following sequence is generated: $(6, 6) \stackrel{0001}{\longrightarrow} (6, 5) \stackrel{0001}{\longrightarrow} (6, 4) \stackrel{0010}{\longmapsto} (6, 5) \stackrel{0011}{\longrightarrow} (5, 4) \stackrel{0001}{\longrightarrow} (5, 3) \stackrel{0010}{\longmapsto} (5, 4) \stackrel{0011}{\longrightarrow} (4, 3) \stackrel{0100}{\longrightarrow} (4, 4) \stackrel{0100}{\longrightarrow} (4, 5) \stackrel{0100}{\longrightarrow} (4, 6)$. Here $(6, 6) \stackrel{0001}{\longrightarrow} (6, 5)$ denotes the transition $(6, 6) \rightarrow (6, 5)$ with input $\tilde{u} = (0, 0, 0, 1)$, and $(6, 4) \stackrel{0010}{\longmapsto} (6, 5) \stackrel{0011}{\longrightarrow} (5, 4)$ indicates that the transition $(6, 4) \rightarrow (5, 4)$ is realized after preventing $(6, 4) \stackrel{0010}{\longrightarrow} (6, 5)$ with input $\tilde{u} = (0, 0, 1, 1)$. From the simulation it follows that Assumption 5.1.3 is not violated.

It is important to realize that the proposed controller design method is systematic, *i.e.* the computation of the forceability graph (in fact a subgraph) is performed in an automated way. The advantage is that no specialistic insight or knowledge of operators of the plant is needed, which is often the case with 'rule-based' control strategies.

5.5 'Forceable set-transition' control strategy

The previous controller design method is restrictive in the sense that for the existence of a forceable transition it is necessary that *all* undesired transitions have to be preventable. The controller design method discussed in this section relaxes this. Even if a transition to a certain discrete state cannot be prevented, this will not be a problem when from the new state a desired transition to another state is again possible. Instead of forceable transitions between *states*, we are now considering forceable transitions between *sets*. The key idea is to compute a finite sequence of nested ('target set'-)controlled invariant sets (see Section 5.3). Each set contains its predecessor. The first set contains the target set and the last set contains the initial state. Furthermore, as these sets are ('target set'-)controlled invariant, the state trajectory can only move to the predecessor of the current set, which is smaller and 'closer' to the target set.

It is no longer possible to construct a forceability graph, which has the advantage that all paths are available for each initial state $\tilde{x}_0$ and target state $\tilde{x}_e$ beforehand. Instead, for the given initial state $\tilde{x}_0$ and the set of target states $\tilde{X}_e$, the following procedure is performed to obtain the nested sequence of sets.

1. **(Initialization)** For the reachability problem, set $\hat{z}_1 := \hat{x}_1 := \hat{x}_e$ (which can be a Boolean representation of a set). For the stabilizing problem we compute the largest controlled invariant set $\tilde{X}_{inv}$ for the set of inputs $\tilde{U}$ contained in $\tilde{X}_e$. If there is no non-trivial solution, then the stabilizing problem cannot be solved. The best alternative is to compute the smallest controlled invariant set $\tilde{X}_{inv}$ for the set of inputs $\tilde{U}$ containing the target set of discrete states $\tilde{X}_e$. Next, we set $\hat{z}_1 := \hat{x}_1 := \hat{x}_{inv}$ ($\tilde{Z}_1 := \tilde{X}_1 := \tilde{X}_{inv}$) and $k := 1$.

2. **(Moving neighbors)** Next, the set of discrete states $\tilde{X}_2$ containing all $\tilde{x}_2' \notin \tilde{X}_1$ for which there exist moving inputs for some transition $\tilde{x}_2' \rightarrow \tilde{x}_1' \in \tilde{X}_1$ is computed, *i.e.*

$$\hat{x}_2 = (D^T \hat{x}_1) \ominus \hat{x}_1.$$

3. **(Check controlled invariance)** Check for each single discrete state $\tilde{x}_2' \in \tilde{X}_2$, whether transitions to states $\tilde{x} \notin \tilde{X}_1 \cup \tilde{X}_2$ can be prevented by at least one of the moving inputs for some $\tilde{x}_2' \rightarrow \tilde{x}_1' \in \tilde{X}_1$. That is, compute

$$\hat{x}_3 = S_{\tilde{U}_m} \hat{x}_2' \ominus (\hat{x}_2 \oplus \hat{x}_1),$$

with $\tilde{U}_m = \{\tilde{u} \in \tilde{U} \mid \hat{x}_1^T D_{\tilde{u}} \hat{x}_2' \neq 0\}$. Consider the following two cases.

i) (Is controlled invariant) If $\hat{x}_3 = 0$ for all $\tilde{x}'_2 \in \tilde{X}_2$, then we set $\tilde{Z}_{k+1} := \tilde{X}_1 \cup \tilde{X}_2$, which is $\tilde{Z}_1$-controlled invariant[2]. If the initial state $\tilde{x}_0 \in \tilde{Z}_{k+1}$, then the algorithm has successfully finished. Otherwise, set $k := k + 1$, rename $\hat{x}_1 := \hat{x}_1 \oplus \hat{x}_2$ and execute step 2 again.

ii) (Make controlled invariant) If $\hat{x}_3 \neq 0$, then this implies that we cannot prevent the transition $\tilde{x}'_2 \rightarrow \tilde{x}'_3 \in \tilde{X}_3$ by any input $\tilde{u} \in \tilde{U}_m$. The discrete state $\tilde{x}'_2$ is removed from $\tilde{X}_2$, *i.e.* set $\hat{x}_2 := \hat{x}_2 \ominus \hat{x}'_2$. If $\hat{x}_2 = \hat{x}_1$, then no further improvement is possible and the strategy fails. Otherwise, perform step 3 again for the new $\tilde{X}_2$.

If finished successfully, the procedure results in a sequence of nested, $\tilde{Z}_1$-controlled invariant sets, $\tilde{x}_0 \in \tilde{Z}_r \supset \tilde{Z}_{r-1} \supset ... \supset \tilde{Z}_2 \supset \tilde{Z}_1$.

The connection with the first controller design method can be explained by using the directed graph presented in Figure 5.7.

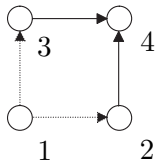

Figure 5.7: *Forceable states versus forceable sets*

In this figure, the solid arrows indicate forceable transitions whereas the dashed arrows represent transitions that cannot be forced but for which moving inputs exist. As can be seen, there is no forceable path from discrete state 1 to state 4. Hence, the first controller design method would fail to solve the reachability problem (from 1 to 4) for this situation. However, we are certain that a transition from discrete state 1 to one of the states in the set $\{2, 3\}$ will happen. Hence, the transition between the sets $\{1\}$ and $\{2, 3\}$ is forceable. From the set $\{2, 3\}$ we then finally can force a transition to the discrete state 4. Consequently the methodology discussed in this section yields a solution in this case.

5.5.1 Control strategy

Instead of forcing transition between states, this controller design method is based on forcing transitions between sets. The aim of the controller strategy is to let the continuous state $x(t)$ evolve from one $\tilde{Z}_1$-controlled invariant set $\tilde{Z}_k$

[2]In case $\tilde{Z}_1$ is controlled invariant (as *e.g.* in the stabilization problem), then a $\tilde{Z}_1$-controlled invariant set containing $\tilde{Z}_1$ is controlled invariant as well.

to the smaller set $\tilde{Z}_{k-1}$, which is 'closer' to the target set $\tilde{Z}_1$. This is repeated until the set of target discrete states $\tilde{Z}_1$ (equal to $\tilde{X}_e$ in the reachability problem) is entered. Initially, from $\tilde{x}_0 \in \tilde{Z}_r$ the set of inputs $\tilde{U}_m$ is computed which are moving to the set $\tilde{Z}_{r-1}$. This set is given by $\tilde{U}_m = \{\tilde{u} \in \tilde{U} \mid \hat{z}_{r-1}^T D_{\tilde{u}} \hat{x}_0 \neq 0\}$. Three situations are now distinguished.

- Whenever it is observed that a transition is about to occur to a discrete state $\tilde{x}_{new} \notin \tilde{Z}_r$ (*i.e.* the situation would be worse after the transition), then an input is chosen from $\tilde{U}_m$, which is preventing for the transition $\tilde{x} \rightarrow \tilde{x}_{new}$. That is choose $\tilde{u} \in \tilde{U}_m$ for which $\hat{x}_{new}^T A_{\tilde{u}} \hat{x}_0 = 0$.
- If $\tilde{x}_{new} \in \tilde{Z}_r \setminus \tilde{Z}_{r-1}$ (*i.e.* the 'distance' to the target set stays 'equal'), then the current input is maintained for τ time units to ensure that the transition actually occurs (similar as in Section 5.4.3). After τ time units an input is chosen that is moving for the transition $\tilde{x}_{new} \rightarrow \tilde{Z}_{r-1}$.
- In case $\tilde{x}_{new} \in \tilde{Z}_{r-1}$, the current moving input will be maintained for τ time units as well, to guarantee that the set $\tilde{Z}_{r-1}$ is entered, which is 'closer' to $\tilde{Z}_1$ (*i.e.* the target set). Next, the strategy aims at moving from $\tilde{Z}_{r-1}$ to $\tilde{Z}_{r-2}$[3] by switching to an input that is moving for $\tilde{Z}_{r-2}$ after τ time units.

If in the latter two cases a boundary of an undesired discrete state is reached before τ time units have expired, then a preventing input is directly applied which is moving for $\tilde{Z}_{r-1}$ or $\tilde{Z}_{r-2}$, respectively. The parameter $\tau > 0$, can be chosen freely.

By these two control actions, we ensure that the trajectory can only leave the set $\tilde{Z}_r$ to a set $\tilde{Z}_{r-1}$. Next, the strategy aims at moving from $\tilde{Z}_{r-1}$ to $\tilde{Z}_{r-2}$ in a similar manner until $\tilde{Z}_1$ is actually reached.

5.5.2 Troublesome situation

In general, it is not possible to guarantee that the state will evolve to a smaller set. Although moving inputs are used to try to let the state evolve in the direction of the desired state, it is not certain that the state will actually reach a smaller controlled invariant set as can be shown by looking at the following example.

EXAMPLE 5.5.1 *Consider the discrete states $\tilde{x}_1$, $\tilde{x}_2$, $\tilde{x}_3$, and $\tilde{x}_4$ and their corresponding hypercubes, depicted in Figure 5.8. The objective is to reach the set $\{\tilde{x}_1, \tilde{x}_3, \tilde{x}_4\}$ from $\tilde{x}_2$. Suppose we start in $\xi(t_0)$ with an input $\tilde{u}$ that*

[3] In case $\tilde{x}_{new} \in \tilde{Z}_k$ for some $k < r-1$ then one might improve by choosing an input that is moving from $\tilde{Z}_k$ to $\tilde{Z}_{k-1}$.

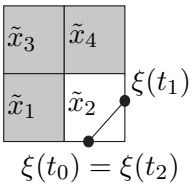

Figure 5.8: *A cyclic trajectory in* $H_x(\tilde{x}_2)$

is moving for $\tilde{x}_2 \rightarrow \tilde{x}_4$*. However, in order to prevent the state from leaving* $H_x(\tilde{x}_2)$*, at time* t_1 *a preventing input is applied that is moving for* $\tilde{x}_2 \rightarrow \tilde{x}_1$*. It might be the case that at time* t_2 *the point* $x(t_0)$ *is reached again and the input* $\tilde{u}$ *is applied again, since it is preventing and moving. Thus, a cyclic trajectory arises that prevents us from reaching the set* $\{\tilde{x}_1, \tilde{x}_3, \tilde{x}_4\}$*.*

There is no proof that the continuous trajectory will reach a smaller controlled invariant set in general. However, it is guaranteed that the situation will never become worse. A possible (heuristic) solution is to keep on aiming at the same state (*e.g.* $\tilde{x}_4$ in the example) as long as possible.

5.5.3 Example: two tank system

Again, we consider the two tank system discussed in Section 5.4.4. For this example the aim is to reach the discrete state $\tilde{x}_e = (4, 6)$ from the initial state $\tilde{x}_0 = (6, 6)$. If we compute the $\tilde{x}_e$-controlled invariant sets by exploiting the procedure discussed in this section then this results in 9 sets (including the target state $\tilde{x}_e$), $\{\tilde{Z}_k\}_{k=1,...,9}$. It turns out that these sets are actually controlled invariant instead of only $\tilde{x}_e$-invariant. In Figure 5.9 the sets $\tilde{Z}_{k+1} \setminus \tilde{Z}_k$ are depicted. Each such set has the same color or shading in the figure and consist of discrete states which are k steps remote from the target state $\tilde{x}_e$, where $k = 1, ..., 8$. In Figure 5.9 also all paths are shown for which moving inputs exist and that will lead to the target state $\tilde{x}_e = (4, 6)$ within 8 steps. Starting in the initial discrete state $\tilde{x}_0 = (6, 6) \in \tilde{Z}_9$ which is 8 transitions remote from the target state $\tilde{x}_e$, it is certain (for this example) that we can move to one of the states in the invariant set $\tilde{Z}_8$ consisting of states being 7 transitions remote from the target state. This is repeated until the target state $\tilde{x}_e$ is reached. If this control strategy is applied to the two tank system, exactly the same time-trajectory and input sequence is obtained as for the example in Section 5.4.4. Further details of the time-trajectory have already been given in Section 5.4.4.

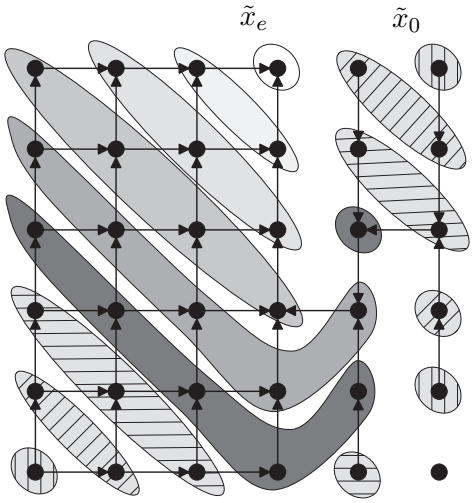

Figure 5.9: *Digraph of all paths for which moving inputs exist ending in* $\tilde{x}_e$

5.6 'Invariant sets' control strategy

The third design method can also be used for both the stabilization problem and the reachability problem; there are just some minor differences between the applied strategies for each of the problems, as will be pointed out. Recall that Assumption 5.2.9 holds.

This last design method resembles the procedure described in the previous section in the sense that it constructs a sequence of nested ('target set'-)controlled invariant sets. However, instead of making a set controlled invariant by deleting states, it is tried to make the set controlled invariant by adding states. As a consequence, steps 1 and 2 remain the same as the first two steps of the previous synthesis method, while step 3 differs.

1. **(Initialization)** For the reachability problem, set $\hat{z}_1 := \hat{x}_1 := \hat{x}_e$ (which can be a Boolean representation of a set). For the stabilization problem we compute the largest controlled invariant set $\tilde{X}_{inv}$ for the set of inputs $\tilde{U}$ contained in $\tilde{X}_e$. If there is no non-trivial solution, then the stabilization problem cannot be solved. The best alternative is to compute the smallest controlled invariant set $\tilde{X}_{inv}$ for the set of inputs $\tilde{U}$ containing the target discrete state $\tilde{X}_e$, see the shaded hypercubes in Figure 5.10 (a). Next, we set $\hat{z}_1 := \hat{x}_1 := \hat{x}_{inv}$ ($\tilde{Z}_1 := \tilde{X}_1 := \tilde{X}_{inv}$) and $k := 1$.

2. **(Moving neighbors)** Next, the set of discrete states $\tilde{X}_2$ containing all $\tilde{x}'_2 \notin \tilde{X}_1$ for which there exist moving inputs for some transition $\tilde{x}'_2 \rightarrow \tilde{x}'_1 \in \tilde{X}_1$ is computed, *i.e.*

$$\hat{x}_2 = (D^T \hat{x}_1) \ominus \hat{x}_1$$

(described by the dark Hypercubes in 5.10 (b)).

3. **(Check controlled invariance)** Check for each single discrete state $\tilde{x}'_2 \in \tilde{X}_2$ whether transitions to states $\tilde{x} \notin \tilde{X}_1 \cup \tilde{X}_2$ can be prevented by at least one of the moving inputs for $\tilde{x}'_2 \to \tilde{x}'_1$ for some $\tilde{x}'_1 \in \tilde{X}_1$. That is, compute

$$\hat{x}_3 = S_{\tilde{U}_m} \hat{x}'_2 \ominus (\hat{x}_2 \oplus \hat{x}_1),$$

with $\tilde{U}_m := \{\tilde{u} \in \tilde{U} \mid \hat{x}_1^T D_{\tilde{u}} \hat{x}'_2 \neq 0\}$. Consider the following two cases.

   i) (Is controlled invariant) If $\hat{x}_3 = 0$ for all $\tilde{x}'_2 \in \tilde{X}_2$ then $\tilde{X}_1 \cup \tilde{X}_2$ is $\tilde{Z}_1$-controlled invariant[4] and will be denoted by $\tilde{Z}_{k+1}$, see Figure 5.10 (c). If the initial state $\tilde{x}_0 \in \tilde{Z}_{k+1}$, then the algorithm has finished successfully, see Figure 5.10 (d). Otherwise, set $k := k+1$, rename $\hat{x}_1 := \hat{x}_1 \oplus \hat{x}_2$ and execute step 2 again.

   ii) (Make controlled invariant) If $\hat{x}_3 \neq 0$ for some $\tilde{x}'_2 \in \tilde{X}_2$, then this implies that for all $\tilde{x}'_3 \in \tilde{X}_3$ the transition $\tilde{x}'_2 \to \tilde{x}'_3$ cannot be prevented by any input $\tilde{u} \in \tilde{U}_m$. Instead of deleting $\tilde{x}'_2$ as in Section 5.5 to (try to) construct a $\tilde{Z}_1$-controlled invariant set $\tilde{Z}_{k+1}$, $\tilde{X}_3$ is now added to $\tilde{X}_2$ under the condition that for each $\tilde{x}'_3 \in \tilde{X}_3$ there exists a moving input for $\tilde{x}'_3$ back to $\tilde{X}_2$. This means that if

$$\hat{x}_2^T D \hat{x}'_3 \neq 0, \tag{5.10}$$

   for each $\tilde{x}'_3 \in \tilde{X}_3$, then we rename $\tilde{X}_1$ and $\tilde{X}_2$ such that

$$\hat{x}_1 := \hat{x}_1 \oplus \hat{x}_2, \ \hat{x}_2 := \hat{x}_3$$

   and repeat step 3. In case (5.10) is not satisfied, the design method fails and the procedure is stopped.

If successfully finished, the procedure results in a sequence of nested, $\tilde{Z}_1$-controlled invariant sets, $\tilde{x}_0 \in \tilde{Z}_r \supset \tilde{Z}_{r-1} \supset ... \supset \tilde{Z}_2 \supset \tilde{Z}_1$.

5.6.1 Control strategy

Similar as for the previous control strategy, the aim is to let the continuous state $x(t)$ evolve from one $\tilde{Z}_1$-controlled invariant set $\tilde{Z}_k$ to the smaller set $\tilde{Z}_{k-1}$. This is repeated until the set of target discrete states $\tilde{Z}_1$ (equal to $\tilde{X}_e$ in the reachability problem) is entered. Initially, from $\tilde{x}_0 \in \tilde{Z}_r$ the set of inputs $\tilde{U}_m$ is computed which are moving to the set $\tilde{Z}_{r-1}$. This set is given by $\tilde{U}_m = \{\tilde{u} \in \tilde{U} \mid \hat{z}_{r-1}^T D_{\tilde{u}} \hat{x}_0 \neq 0\}$. Again, three situations are distinguished.

[4]Recall that in case $\tilde{Z}_1$ is controlled invariant (as *e.g.* in the stabilization problem), then a $\tilde{Z}_1$-controlled invariant containing $\tilde{Z}_1$ set is controlled invariant as well.

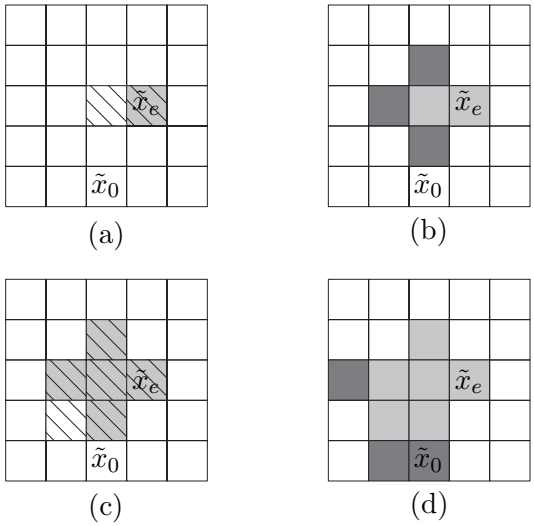

Figure 5.10: *The construction of nested, controlled invariant sets*

- Whenever it is observed that a transition is about to occur to a discrete state $\tilde{x}_{new} \notin \tilde{Z}_r$ (*i.e.* the situation would be worse after the transition), then an input is chosen from $\tilde{U}_m$, which is preventing for the transition $\tilde{x} \to \tilde{x}_{new}$. That is choose $\tilde{u} \in \tilde{U}_m$ for which $\hat{x}_{new}^T A_{\tilde{u}} \hat{x}_0 = 0$.

- If $\tilde{x}_{new} \in \tilde{Z}_r \setminus \tilde{Z}_{r-1}$ (*i.e.* the 'distance' to the target set stays 'equal'), then the current input is maintained for τ time units to ensure that the transition actually occurs (similar as in Section 5.4.3). After τ time units an input is chosen that is moving for the transition $\tilde{x}_{new} \to \tilde{Z}_{r-1}$.

- In case $\tilde{x}_{new} \in \tilde{Z}_{r-1}$, the current moving input will be maintained for τ time units as well, to guarantee that the set $\tilde{Z}_{r-1}$ is entered, which is 'closer' to $\tilde{Z}_1$ (*i.e.* the target set). Next, the strategy aims at moving from $\tilde{Z}_{r-1}$ to $\tilde{Z}_{r-2}$[5] by switching to an input that is moving for $\tilde{Z}_{r-2}$ after τ time units.

If in the latter two cases a boundary of an undesired discrete state is reached before τ time units have expired, then a preventing input is directly applied which is moving for $\tilde{Z}_{r-1}$ or $\tilde{Z}_{r-2}$, respectively. The parameter $\tau > 0$, can be chosen freely.

By these two control actions, we ensure that the trajectory can only leave the set $\tilde{Z}_r$ to a set $\tilde{Z}_{r-1}$. Next, the strategy aims at moving from $\tilde{Z}_{r-1}$ to $\tilde{Z}_{r-2}$ in a similar manner until $\tilde{Z}_1$ is actually reached.

[5]In case $\tilde{x}_{new} \in \tilde{Z}_k$ for some $k < r-1$ then one might improve by choosing an input that is moving from $\tilde{Z}_k$ to $\tilde{Z}_{k-1}$.

5.6.2 Pathological case

An important issue of this method is that in principle no guarantee can be given that the state will evolve to a smaller set. Moving inputs are used to try to let the state evolve in the direction of the desired state. In spite of this aim, it is not certain that the state will actually reach a smaller controlled invariant set, as the same situation can occur as discussed in Example 5.5.1. For the controller strategy discussed in this section there might arise another troublesome situation as shown in the example depicted in Figure 5.11.

Example 5.6.1 (Pathological case) *For the grey states in Figure 5.11 there exist moving inputs to the target state (dark grey in the center), indicated by the solid arrows. This set is made controlled invariant by extending it with states such that the shaded area results (step 3.ii) of the algorithm). However, for the grey states transitions to the added states cannot be prevented, as indicated by the dotted arrows. For the added states there exist moving inputs to the grey states. It is clear from the picture that it is not guaranteed that the target state is reached since it is possible that cyclic trajectories exist.*

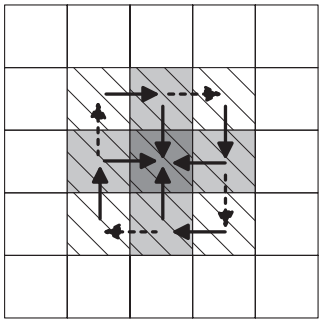

Figure 5.11: *Set of states (grey states) which are moving for the target state (dark grey) and the corresponding controlled invariant set (shaded)*

As indicated by the example, no rigorous proof can be given that the proposed synthesis method will succeed. However, it can be guaranteed that the controller will never worsen the current situation.

5.6.3 Example: two tank system

Consider the two tank system as defined in Section 5.4.4. This time, the aim is to control the continuous state from the initial state $x_0 = [0.42, 0.44]^T \in H_x(\tilde{x}_0)$, that is $\tilde{x}_0 = (6, 6)$ to the target discrete state $\tilde{x}_e = (3, 6)$. The $\tilde{x}_e$-controlled invariant sets that are computed by performing the steps discussed in this section actually turn out to be controlled invariant, as $\tilde{x}_e$ is

controlled invariant. The resulting sequence $\{\tilde{Z}_i\}_{i=1,\ldots 10}$ is depicted in Figure 5.12, where the sets are pictured in the forceability graph of the two tank system as explained in Section 5.4.2.

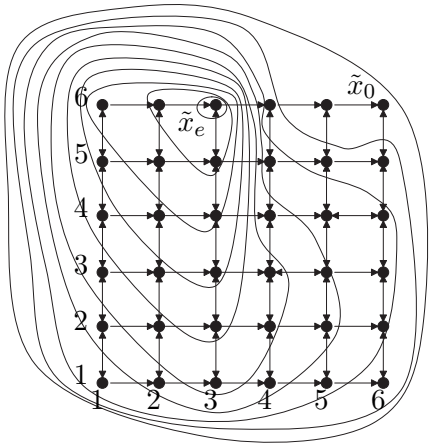

Figure 5.12: *Controlled invariant sets originating from* $\tilde{x}_e = (3, 6)$*, depicted in the forceability graph of the two tank system*

From this forceability graph, it can be seen that there is no path ending in $\tilde{x}_e = (3, 6)$ and originating in $\tilde{x}_0 = (6, 6)$, therefore the 'forceable state-transitions' controller design method will not succeed. From Figure 5.12 it can be seen that there exists a sequence of nested $\tilde{x}_e$-controlled invariant sets, the first beginning in $\tilde{x}_e = (3, 6)$, and the last covering all discrete states and thus including $\tilde{x}_0 = (6, 6)$. Note, that since the $\tilde{Z}_{10}$ covers all discrete states, we obtained a global controller (and stabilizer) to $\tilde{x}_e$. Applying the proposed control strategy with $\tau = 0.1$ [s] results in the time-response as shown in Figure 5.13.

In Figure 5.13 also the time instants are shown for which the input changes. Starting with all switches closed except for s_4, the second tank drops its level until after approximately 21 seconds the discrete state $(6, 4)$ is reached. After τ seconds, at t_1 switch s_3 opens and s_4 closes in order to reach state $(5, 4)$. However, this causes the level in the second tank to rise which would result in a transition back to the state $(6, 5)$. This is prevented at time t_2 by closing s_3 and opening s_4. As a consequence, the level in the second tank lowers again, until at t_3 the transition $(6, 4) \rightarrow (6, 3)$ is prevented by closing s_4 and opening s_3. Eventually, the sequence of discrete states, which results from the control actions is: $(6,6) \xrightarrow{0001} (6,5) \xrightarrow{0001} (6,4) \xmapsto{0010} (6,5) \xmapsto{0001} (6,3) \xrightarrow{0010} (5,4) \xrightarrow{0001} (5,3) \xmapsto{0010} (5,4) \xrightarrow{0011} (4,3) \xmapsto{0001} (4,2) \xrightarrow{0100} (3,3) \xrightarrow{0100} (3,4) \xrightarrow{0100} (3,5) \xrightarrow{0100} (3,6)$. Here, the notation is similar as in Section 5.4.4. From the simulation it can be seen that Assumption 5.1.3 is not violated.

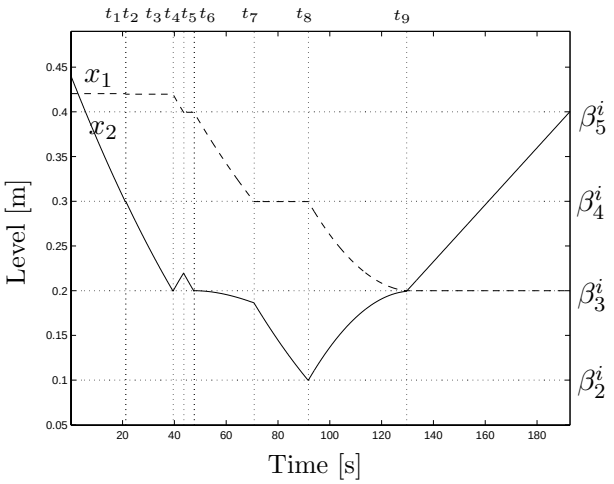

Figure 5.13: *Trajectory for* $\tilde{x}_0 = (6, 6)$ *and* $\tilde{x}_e = (3, 6)$

5.7 Transition measurements

So far, the proposed controller design methods are explained for the case that it is detected when the continuous state reaches the boundary between two states and instantaneous control action is possible. This is valid, when, for instance, an optical sensor is used that emits a signal whenever a beam of light is interrupted, and a (infinitely) fast actuator is available. This assumption allows the usage of preventing inputs. For many practical situations these assumptions may however be too strong. Some mechanical sensors, *e.g.* switches are constructed such that a signal is emitted when the 'state' already has changed. Furthermore, instantaneous control action may be impossible due to large computation-times or slow actuators. The proposed controller design methods can be adapted such that they are suitable for this last situation as well. Note that the definition of a discretely controlled trajectory is still valid: a discretely controlled trajectory is a trajectory for which the applied (discrete) input only changes after each measurement (transition).

5.7.1 Strong forceability graph

In Section 5.4.1 a sufficient condition for forceability was based on preventing undesired transitions (Proposition 5.4.3). As already stated, preventing transitions requires the exact detection of the continuous state reaching the boundary between two adjacent hypercubes and instantaneous control actions (see Assumption 5.2.9). To relax these requirements, the concept of a strongly forceable transition is introduced. For a strongly forceable transition it is demanded that an undesired transition can be corrected instead of prevented.

The transition actually has occurred but can be corrected; as long as the continuous time trajectory ξ is in the open set Ω corresponding to the particular transition, see Proposition 5.2.5. To prevent technicalities, the next definition is in some sense an extension of Definition 5.4.1 based on the sufficient condition in Proposition 5.4.3 for the case of correcting inputs.

DEFINITION 5.7.1 *The transition $\tilde{x}_1 \to \tilde{x}_2$ is* strongly forceable, *if there exists a set $\tilde{U}'_m$ of discrete inputs such that, for each transition $\tilde{x}_1 \to \tilde{x}_3 \neq \tilde{x}_2$ to a state $\tilde{x}_3$ adjacent to $\tilde{x}_1$ with an input in $\tilde{u}_1 \in \tilde{U}'_m$, there exists another input $\tilde{u}_2 \in \tilde{U}'_m$ that is correcting for $\tilde{x}_1 \to \tilde{x}_3$ among the inputs that are moving for $\tilde{x}_1 \to \tilde{x}_2$.*

Now, again a graph can be computed by using the definition. The strong forceability graph can be constructed in a similar way as the forceability graph. Given a set of discrete inputs $\tilde{U} = \{\tilde{u}_1, ..., \tilde{u}_k\}$, the matrices $\{K_{\tilde{u}}\}$ and the direction matrices $\{D_{\tilde{u}}\}$ it can be verified whether a transition $\tilde{x}_1 \to \tilde{x}_2$ is forceable by the following steps.

1. Compute all inputs for which $\tilde{x}_1$ is moving to $\tilde{x}_1 \to \tilde{x}_2$. This set is given by

$$\tilde{U}_m = \{\tilde{u} \in \tilde{U} \mid \hat{x}_2^T D_{\tilde{u}} \hat{x}_1 \neq 0\}. \tag{5.11}$$

2. Check if there exists a set $\tilde{U}'_m \subseteq \tilde{U}_m$, such that by choosing some input $\tilde{u} \in \tilde{U}'_m \neq \emptyset$ we can correct for transitions to discrete states other than $\tilde{x}_2$. That is, check if

$$(A_{\tilde{U}'_m} \otimes K_{\tilde{U}'_m})\hat{x}_1 \ominus \hat{x}_2 = 0,$$

with $A_{\tilde{U}'_m} = \bigoplus_{\tilde{u} \in \tilde{U}'_m} A_{\tilde{u}}$, and $K_{\tilde{U}'_m} = \bigotimes_{\tilde{u} \in \tilde{U}'_m} K_{\tilde{u}}$.

To understand the second computation, realize that $\hat{x}_3 := K_{\tilde{U}'_m} \hat{x}_1$ are all states for which a transition from $\tilde{x}_1$ cannot be corrected with one of the inputs in the set $\tilde{U}'_m$. However, it must be possible that a transition can actually occur before we require a transition to be correctable. The set $\hat{x}_4 := A_{\tilde{U}'_m} \hat{x}_1$ are all states that can be reached from $\tilde{x}_1$ with one of the inputs in the set $\tilde{U}'_m$. So, the intersection of both sets, *i.e.* $\hat{x}_5 := \hat{x}_3 \otimes \hat{x}_4$ gives all states that can be reached from $\tilde{x}_1$ and for which the transition cannot be corrected. This can also be written as $\hat{x}_5 = (A_{\tilde{U}'_m} \otimes K_{\tilde{U}'_m})\hat{x}_1$. The set difference $\hat{x}_5 \ominus \hat{x}_2$ are now all states unequal to $\tilde{x}_2$ that can be reached from $\tilde{x}_1$ and for which the transition is not correctable

Notice that the computation of 1) and 2) corresponds to checking the condition stated in Definition 5.7.1. By performing the computations for all transitions $j \rightarrow i$ the strong forceability matrix P^s is constructed:

$$p^s_{ij} = \begin{cases} 1 & (A_{\tilde{U}'_m} \otimes K_{\tilde{U}'_m})\hat{x}_1 \ominus \hat{x}_2 = 0 \\ 0 & \text{else,} \end{cases}$$

where $\tilde{x}_1 = j$, and $\tilde{x}_2 = i$.

Hence $\hat{x}_2 = P^s\hat{x}_1$ represents all discrete states to which a transition from $\tilde{x}_1$ is strongly forceable. If we label each edge in the strong forceability graph with the set of inputs $\tilde{U}'_m$ as defined in (5.11) then we have the *labelled strong forceability graph.* The labels indicate which inputs we can choose from to correct transitions and still guarantee movement to a desired transition. The control strategy discussed in Section 5.4.3 can now be used, based on the strong forceability graph P^s.

5.7.2 Correcting versus preventing inputs

The 'invariant sets' and the 'forceable set-transition' controller design methods are easily adapted for the situation where it is detected that a transition already has occurred. The only change is the replacement of the Boolean matrix $S_{\tilde{U}_m}$ which is used for determining the set of states to which possible transitions cannot be prevented with inputs from the set $\tilde{U}_m$, by the Boolean matrix

$$(A_{\tilde{U}'_m} \otimes K_{\tilde{U}'_m}),$$

with $A_{\tilde{U}'_m} = \bigoplus_{\tilde{u}\in\tilde{U}'_m} A_{\tilde{u}}$, and $K_{\tilde{U}'_m} = \bigotimes_{\tilde{u}\in\tilde{U}'_m} K_{\tilde{u}}$. Here $\tilde{U}'_m \subseteq \tilde{U}_m$ is a set for which the number of discrete states in $\tilde{X}_2$ computed by $\hat{x}_2 = (A_{\tilde{U}'_m} \otimes K_{\tilde{U}'_m})\hat{x}_1$ is minimal. Recall that $\tilde{X}_2$ is the set of states to which transitions from $\tilde{x}_1$ cannot be corrected with inputs from the set $\tilde{U}'_m$. Indeed, $\tilde{U}'_m$ needs not to be unique; there might be different subsets of $\tilde{U}_m$ resulting in the same number of discrete states in $\tilde{X}_2$. In this case, we just have to choose one, noticing that different sets might result in different controlled invariant sets.

5.8 Relaxing Assumption 5.1.3

For the correctness of the controller design methods discussed previously, it is necessary that Assumption 5.1.3 holds. Unfortunately, it is not possible to guarantee a priori that Assumption 5.1.3 will not be violated by the proposed control strategies. First, we will show how the controllers can cause the violation and what problems result from it. Next, we will reduce the possibility

that the assumption will be violated by eliminating one of the causes. Finally, it is shown how, by additional computations, one can handle violation of Assumption 5.1.3.

One of the basic ideas behind all three controller design methods is to prevent undesired transitions from occurring (if possible). This is done by using either preventing inputs or correcting inputs. In Figure 5.14 it is shown how preventing inputs can result in trajectories ξ that reach points that belong to three hypercubes. In Figure 5.14 (a) the continuous-time trajectory evolves along the boundary plane separating the two corresponding hypercubes due to a preventing input. But, if a trajectory evolving on a boundary plane reaches a different boundary plane (see Figure 5.14 (a)), then Assumption 5.1.3 is no longer satisfied. Another reason that might result in violating the assumption is the switching of inputs that are used for preventing or correcting transitions, as depicted in Figure 5.14 (b). Here, it is tried to force the transition $\tilde{x}_1 \rightarrow \tilde{x}_2$. However, both inputs that are preventing (or correcting) undesired transitions move the trajectory ξ to the same corner. This will result in very fast switching and, eventually, a point will be reached that is an elements of more than two hypercubes, thus violating Assumption 5.1.3.

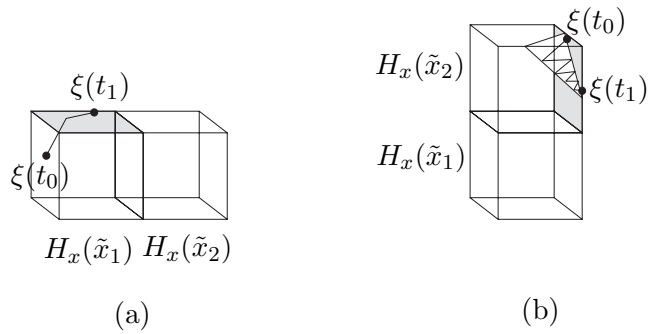

Figure 5.14: *(a) Trajectory ξ evolving along boundary plane due to preserving input; (b) switching inputs let the trajectory ξ evolve to an edge*

It is clear that, if Assumption 5.1.3 does not hold, then Assumption 3.2.1 in Chapter 3 will easily be violated, which means that the modelling method of Chapter 3 cannot be used without additional transitions. Furthermore, if Assumption 5.1.3 is violated, then an input $\tilde{u}$ which is preventing or correcting for a transition $\tilde{x}_1 \rightarrow \tilde{x}_2$ no longer guarantees that the transition indeed is prevented or corrected. Indeed, it is still guaranteed that the trajectory ξ will not enter (or will leave) $H_x(\tilde{x}_2)$, but it is not clear that ξ will 'return' to $H_x(\tilde{x}_1)$, because other hypercubes might be reached (see Figure 5.15).

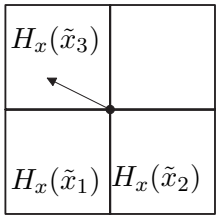

Figure 5.15: *The transition $\tilde{x}_1 \to \tilde{x}_2$ is prevented, but the trajectory will not return to $H_x(\tilde{x}_1)$*

5.8.1 Using correcting inputs

The problem of preventing inputs which resulting in continuous-time trajectories evolving on boundary planes between hypercubes can be solved by utilizing correcting inputs instead of preventing inputs. Indeed, by using correcting inputs, we are always certain (due to the strictness of inequality (5.6) that a trajectory will not evolve along a boundary, but moves away from it.

5.8.2 Additional computations

If nevertheless Assumption 5.1.3 is violated, additional computations are necessary to obtain a controller that guarantees that a transition can indeed be prevented (or corrected). For instance, for a point x on the boundary $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2) \cap H_x(\tilde{x}_3)$ for different $\tilde{x}_1, \tilde{x}_2$, and $\tilde{x}_3$, it must be shown that preventing the transition $\tilde{x}_1 \to \tilde{x}_2$ will not result in the (possibly undesired) transition $\tilde{x}_1 \to \tilde{x}_3$, see Figure 5.16.

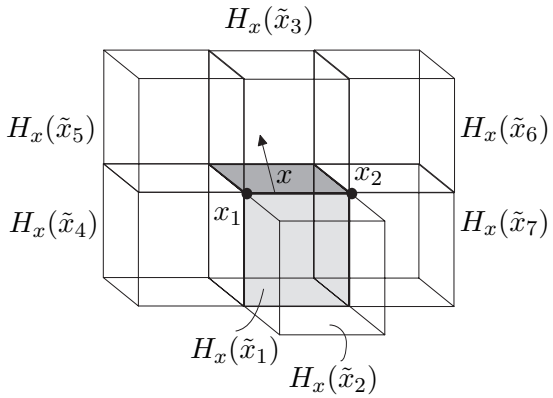

Figure 5.16: *Points on the boundary plane $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$ that also belong to other hypercubes*

To guarantee that applying an input $\tilde{u}$ which is preventing for $\tilde{x}_1 \to \tilde{x}_2$ will

not result in an undesired other transition, it must be checked for all points x that also belong to lower dimensional boundaries of $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$, that the derivative in x, given by $f(x,u)$, points inside the interior of $H_x(\tilde{x}_1)$ for all $u \in H_u(\tilde{u})$. If it is measured on which intersection of boundary planes (*i.e.* on which lower dimensional boundary) the continuous-time trajectory is (as we assume) then it is sufficient that this holds for all points on the intersection at hand. For example, for all points on the line-segment given by the intersection $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2) \cap H_x(\tilde{x}_3)$ excluded by points x_1 and x_2 in Figure 5.16 the derivative has to point inside $int(H_x(\tilde{x}_1))$ for all $u \in H_u(\tilde{u})$. If it is measured that the trajectory is on this particular line-segment, then this input $\tilde{u}$ can be applied to prevent the transition $\tilde{x}_1 \rightarrow \tilde{x}_2$ when necessary (correcting inputs can be applied as long as we are in the neighborhood of the line-segment). If it is measured that the trajectory is in one of the points x_1 or x_2 then for these points there also have to exist inputs $\tilde{u}_1$ and $\tilde{u}_2$ such that for all $u_1 \in H_u(\tilde{u}_1)$ and $u_2 \in H_u(\tilde{u}_2)$ it holds that $f(x_1,u_1)$ and $f(x_2,u_2)$ point inside the interior of $H_x(\tilde{x}_1)$. If such inputs do not exist, then the transition $\tilde{x}_1 \rightarrow \tilde{x}_2$ cannot be prevented (or corrected) while guaranteeing that the continuous state will remain in $H_x(\tilde{x}_1)$ and will not go to $H_x(\tilde{x}_3)$ from points on (lower dimensional) boundaries of $H_x(\tilde{x}_1) \cap H_x(\tilde{x}_2)$.

To be precise, the following steps have to be executed for computing the inputs that are preventing for the transition $\tilde{x}_1 \rightarrow \tilde{x}_2$,

1. Compute all preventing inputs as usual, *i.e.* compute the set $\{\tilde{u} \in \tilde{U} \mid \hat{x}_2^T A_u \hat{x}_1 = 0\}$. If this set is empty, then $\tilde{x}_1 \rightarrow \tilde{x}_2$ cannot be prevented.

2. Determine all lower dimensional boundaries that arise from $H_x(\tilde{x}_1)$ intersecting other hypercubes $H_x(\tilde{x}_i)$ adjacent to $H_x(\tilde{x}_2)$. For the n-dimensional case this results in $n-2$, $n-3$, ..., 0 dimensional faces. For each face, the lower dimensional faces are excluded (*i.e.* only consider the relative interior of the faces). So, for the 3-dimensional case the objects we obtain are lines (without beginning and end point) and points (which are the beginning and end points of the lines, see *e.g.* Figure 5.16).

3. For all these objects, determine whether an $\tilde{u} \in \tilde{U}$ exists for which $f(x,u)$ points to $int(H_x(\tilde{x}_1))$ for all $u \in H_u(\tilde{u})$ and all points x on these objects. If there does not exist such an input $\tilde{u}$, then it is said that $\tilde{x}_1 \rightarrow \tilde{x}_2$ cannot be prevented.

The actual implementation of this last step can be performed by using optimizations and involves cumbersome book-keeping.

5.9 Notes and references

The 'forceable state-transition' controller design method has been published in (Philips *et al.* 1999*c*). The 'forceable set-transition' and 'invariant set' strategies are based on discretely controlled invariant sets first used (in this context) in (Philips *et al.* 1999*b*). In (Philips *et al.* 1999*b*) a strategy is proposed that resembles the invariant set based methods discussed in this chapter, but is less transparent and does not use moving inputs which reduces the chance that a smaller controlled invariant set will be reached. This method has proven to work for a three tank system. The latter two methods discussed in this chapter are improvements in comparison to this preliminary work. The approach followed for both methods is analogous to well known concepts in control theory, such as the controllability set (also used in dynamic programming) and the Lyapunov approach to stability (also used by computer scientists), see *e.g.* (Passino *et al.* 1994, Sontag 1990).

In this chapter only sufficient conditions were given that guarantee a specific discrete transition (so-called forceability). Necessary and sufficient conditions are not available in the context considered here and is an important problem that deserves further attention. The corresponding problem for affine systems on simplices and rectangles has been treated in (Habets and van Schuppen 2000).

In this chapter methods for controller design are proposed for continuous systems with a 'discretized' state space and a discrete (or discretized) input space. Given a discrete-event model of a such continuous plant, usually one forgets about the continuous origin of the model and just applies the existing theory on designing supervisors for discrete-event systems as developed in (Ramadge and Wonham 1982, Ramadge and Wonham 1987*b*, Ramadge and Wonham 1987*a*) or *e.g.* (Heymann and Lin 1998). However, we believe that the discrete-event models under study in general have a high degree of nondeterminism for which (for our purpose) the existing discrete-event controller strategies are not suited. Moreover, the specification of the control goals as defined in this chapter seems to be difficult to translate into terms of legal languages, as is necessary for the aforementioned discrete controller design methods. By only focussing on the discrete-event model, important information provided by the underlying continuous system (such as derivative information) is neglected. Here, we try to incorporate such additional knowledge to obtain stronger control strategies. In contrast to most of the following controller design methods, our strategies are based on *transitions* between discrete states (which are prevented or corrected) rather than the discrete states itself (*i.e.* where the discrete input is a function of the discrete state $\tilde{u} = f_{con}(\tilde{x})$).

The supervisory control theory of (Ramadge and Wonham 1987*b*) is suited

(in a slightly modified version) for the framework presented in (Moor and Raisch 1999) and used for hierarchical control in (Raisch and Itigin 2000, Raisch *et al.* 2000). This framework uses the behavior approach (Willems 1991) for solving the control problem of satisfying 'acceptable' trajectories.

In (Niinomi *et al.* 1995, Cury *et al.* 1998) continuous systems are studied for which the events are generated by a zero-detector. The discrete-event control problem is solved using the approach presented in (Thistle and Wonham 1994). For this, an outer approximation of the exact original discrete-event model (which may not be finite) is constructed, such that the synthesis of the supervisor can be performed for the resulting finite representation.

Necessary and sufficient conditions for the existence of a stabilizing qualitative controller (*i.e.* the input depends on the discrete state) for discrete time systems are given in (Lunze 1995). Also, it is shown how to determine the control law by constructing the graph of the qualitative model and choosing controls such that a certain vertex, which represents an invariant set (containing the equilibrium state), is the only end vertex of the graph and for all other vertices it holds that they are not strongly connected and no self-circles are possible.

In (Drakunov *et al.* 1993) the nondeterministic automaton resulting from the discretization of the continuous system is represented as a deterministic automaton with unknown disturbances. Next, a sliding mode control strategy for finite automata is used resulting in high frequency switching of the input. The approach followed in (Delchamps 1988, Delchamps 1990) for linear discrete time systems is based on the exact continuous description of the plant, rather than using a discrete-event model. In (Hsu 1985) an optimal control method is proposed on the basis of a deterministic 'cell-to-cell mapping' as an approximation of the continuous dynamics (see also (Papa *et al.* 1997)). In (Kowalewski *et al.* 1999) existing controllers are verified by using discrete approximation of the continuous plant.

Another example of a control strategy that is based on a partitioning of the continuous state space is given by (Angelis and Philips 1999), where a rectangular partitioning is assumed and to each hypercube a constant input is assigned which is computed by solving a set of linear matrix inequalities.

Chapter 6

Fault detection and isolation

Fault detection and isolation based on a discrete-event model of a continuous system is the subject of this chapter. Propositions are given that can be used directly for the purpose of fault detection and isolation, and that can serve as a tool to analyze detection and isolation properties. Next, it is studied if it is possible to reduce the number of discrete states of the discrete-event model without losing information regarding fault detection and isolation. We give a computation method for reducing the complexity of the discrete-event model such that it can be checked if transition and fault detection properties are preserved. The results are illustrated by means of an example.

6.1 Fault detection and isolation

Fault detection and isolation in industrial processes are becoming increasingly important. Main reasons for this are the growing demands for higher product quality, safety and operational reliability. Whereas most approaches for detecting and isolating faults are based on the observation in the continuous measurement space, we will use a fault detection and isolation method which is based on the observation of events. For this, first a discrete-event model is constructed, which is based on the continuous model of the process, such that a complete model is obtained capturing all possible discrete-state transitions as is explained in Chapter 3. This procedure is repeated for all models with faults of interest. Next, by observing events from the real system, it is decided whether or not a particular event must have been caused by a fault and, if possible, by which fault. For this approach it is necessary that a fault persists sufficiently long as to be detectable (*i.e.* long enough to cause an event to occur).

Consider a system and a set of faults $\tilde{D} := \{\tilde{d}_1, ..., \tilde{d}_r\}$, such that if no fault

occurs the dynamics are described by

$$\dot{x}(t) = f(x(t), u(t)), \qquad x(t_0) = x_0, \tag{6.1}$$

with $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$. We will refer to (6.1) as the nominal model. For a fixed fault $\tilde{d} \in \tilde{D}$ the dynamics are described by

$$\dot{x}(t) = f_{\tilde{d}}(x(t), u(t)), \qquad x(t_0) = x_0.$$

Furthermore, the same sets of boundaries $\{\beta_j^i\}$ and $\{\gamma_j^i\}$ apply as for the nominal system. Note, that in this way changes in the process and actuator failures can be modelled, however sensor failures cannot be incorporated.

For the nominal model and for each model with fixed fault $\tilde{d}$, the adjacency matrices are determined following the approach described in Chapter 3. Let $A_{\tilde{u}}$ denote the nominal adjacency matrix for input $\tilde{u}$, that is, *without* faults and let $\{A_{\tilde{u}}^{\tilde{d}}\}$ denote the set of adjacency matrices for $\tilde{u}$ when a fault $\tilde{d} \in \tilde{D}$ occurs.

For our fault detection method we are interested whether a measured transition $\tilde{x}_1 \to \tilde{x}_2$ corresponding to input $\tilde{u}$ cannot happen according to the nominal discrete-event model, but is only caused by some fault $\tilde{d} \in \tilde{D}$. Indeed, the most straightforward approach would be to check if an observed transition $\tilde{x}_1 \to \tilde{x}_2$ cannot be caused by the applied input $\tilde{u}$ according to our nominal model, in which case it would hold that $\tilde{x}_2 \notin \phi(\tilde{x}_1, \tilde{u})$ or, equivalently $\hat{x}_2^T A_{\tilde{u}} \hat{x}_1 = 0$. However, since $\hat{x}_2^T A_{\tilde{u}} \hat{x}_1 = 0$ also could be caused by unmodelled dynamics (outside the modelled faults $\tilde{d} \in \tilde{D}$), we will assume here that we are only interested in the occurrence of faults from the set $\tilde{D}$, which contains all reasonably expectable faults. This implies that given a measured transition $\tilde{x}_1 \to \tilde{x}_2$ which cannot be caused by the applied input $\tilde{u}$ solely, it must be checked if this transition might be caused by (at least) one of the faults $\tilde{d} \in \tilde{D}$ in combination with input $\tilde{u}$. We assume that only one fault at a time can occur.

Our fault detection scheme is based on the following observation:

CONDITION 6.1.1 (FAULT DETECTION CONDITION) *Given a measured transition $\hat{x}_1 \to \hat{x}_2$ and an input $\tilde{u}$ applied at the time the transition occurs, then a fault in $\tilde{D}$ has occurred if*

1. *the observed transition is possible with at least one of the faults $\tilde{d} \in \tilde{D}$ in combination with input $\tilde{u}$ and*

2. *the observed transition is not possible with the applied input $\tilde{u}$ solely (i.e. according the nominal model).*

By using the adjacency matrices $\{A_{\tilde{u}}\}$ and $\{A_{\tilde{u}}^{\tilde{d}}\}$ of the nominal system and the systems with faults $\tilde{d}$, respectively, Condition 6.1.1 is easily checked for an observed transition $\hat{x}_1 \to \hat{x}_2$.

PROPOSITION 6.1.2 (FAULT DETECTION) *Given a measured transition $\hat{x}_1 \to \hat{x}_2$ and an input $\tilde{u}$ applied at the time the transition occurs, then a fault $\tilde{d} \in \tilde{D}$ has occurred if*

$$\hat{x}_2^T (F_{\tilde{u}}^{\tilde{d}}) \hat{x}_1 = 1$$

for some $\tilde{d} \in \tilde{D}$, where $F_{\tilde{u}}^{\tilde{d}} := A_{\tilde{u}}^{\tilde{d}} \ominus A_{\tilde{u}}$ is the fault adjacency matrix for input $\tilde{u}$ and fault $\tilde{d}$.

Proof. Note that $\hat{x}_3 := A_{\tilde{u}}^{\tilde{d}} \hat{x}_1 \ominus A_{\tilde{u}} \hat{x}_1 = (A_{\tilde{u}}^{\tilde{d}} \ominus A_{\tilde{u}}) \hat{x}_1$ represents all states to which transitions are possible from $\tilde{x}_1$ with the fault $\tilde{d}$ and input $\tilde{u}$ reduced by the set of states to which transitions are possible with the nominal model and input $\tilde{u}$. Since $\tilde{x}_2 \in \tilde{X}_3$ if and only if $\hat{x}_2^T \hat{x}_3 = 1$, the proposition holds. ■

For each input $\tilde{u} \in \tilde{U}$ and each fault $\tilde{d} \in \tilde{D}$ the corresponding fault adjacency matrix $F_{\tilde{u}}^{\tilde{d}}$ can be computed. Then, after having observed the transition $\tilde{x}_1 \to \tilde{x}_2$ under input $\tilde{u}$, it can be checked for each fault $\tilde{d}$, whether it could have caused the transition or the transition complies with the nominal model. However, instead of constructing a fault adjacency matrix for each of the faults $\tilde{d}$, it can be convenient for the purpose of fault detection to construct the fault adjacency matrix for input $\tilde{u}$ and the *set* of faults $\tilde{D} := \{\tilde{d}_1, ..., \tilde{d}_r\}$:

$$F_u^{\tilde{D}} := F_u^{\tilde{d}_1} \oplus F_u^{\tilde{d}_2} \oplus ... \oplus F_u^{\tilde{d}_r} = A_{\tilde{u}}^{\tilde{d}_1} \oplus A_{\tilde{u}}^{\tilde{d}_2} \oplus ... \oplus A_{\tilde{u}}^{\tilde{d}_r} \ominus A_{\tilde{u}}.$$

By attaching to each fault $\tilde{d}$ a label, it is possible to directly deduce the set of faults (if any) that could have caused the observed transition $\tilde{x}_1 \to \tilde{x}_2$ with input $\tilde{u}$. For enabling computations, a label is a unique number such that even from the sum of these numbers the set of original labels can be restored. One possibility is to assign a number 2^{j-1} to the fault $\tilde{d}_j$. In this way the labeled fault adjacency matrix for input $\tilde{u}$ and the set of faults $\tilde{D} := \{\tilde{d}_1, ..., \tilde{d}_r\}$ is given by

$$\mathbf{F}_{\tilde{u}}^{\tilde{D}} := 2^0 F_{\tilde{u}}^{\tilde{d}_1} + 2^1 F_{\tilde{u}}^{\tilde{d}_2} + ... + 2^{r-1} F_{\tilde{u}}^{\tilde{d}_r},$$

where the addition is applied in the conventional way.

Finally, the fault adjacency matrix for the *set* of inputs $\tilde{U} = \{\tilde{u}_1, ..., \tilde{u}_k\}$ and the fault $\tilde{d} \in \tilde{D}$ is defined by

$$F_{\tilde{U}}^{\tilde{d}} := F_{\tilde{u}_1}^{\tilde{d}} \oplus F_{\tilde{u}_2}^{\tilde{d}} \oplus ... \oplus F_{\tilde{u}_k}^{\tilde{d}}.$$

Using these matrices, the following results are immediate.

PROPOSITION 6.1.3 (FAULT DETECTION AND ISOLATION) *Given a measured transition $\tilde{x}_1 \to \tilde{x}_2$ and an input $\tilde{u}$ applied at the time the transition occurs, then some fault $\tilde{d} \in \tilde{D}$ has occurred if*

$$\hat{x}_2^T(F_{\tilde{u}}^{\tilde{D}})\hat{x}_1 = 1,$$

or equivalently

$$\hat{x}_2^T(\mathbf{F}_{\tilde{u}}^{\tilde{D}})\hat{x}_1 = q \neq 0.$$

Notice, that for the computations with the labeled fault adjacency matrix the standard algebra applies. The set of possible faults can be derived from q because each fault was labelled in such a way that from the sum of labels the original labels can be deduced by the procedure explained in Chapter 2. For example, assume that $\hat{x}_2^T(\hat{\mathbf{F}}_{\tilde{u}}^{\tilde{D}})\hat{x}_1 = 11$, then it can be seen that the fault $\tilde{d}_1$, $\tilde{d}_2$, or $\tilde{d}_4$ has caused the transition, since $2^{1-1} + 2^{2-1} + 2^{4-1} = 11$.

If we want to detect faults by observing transitions then it is interesting to know whether or not a fault $\tilde{d}_j$ can be observed at all. This is formalized by the definition of detectability of a fault.

DEFINITION 6.1.4 (DETECTABILITY OF A FAULT) *The fault $\tilde{d}$ is said to be detectable, if there exists an input $\tilde{u} \in \tilde{U}$ and a corresponding transition $\tilde{x}_1 \to \tilde{x}_2$ from which the fault can be detected (i.e. $\hat{x}_2^T(F_{\tilde{u}}^{\tilde{d}})\hat{x}_1 = 1$).*

The following observation is straightforward.

PROPOSITION 6.1.5 *The fault $\tilde{d}$ is detectable if*

$$F_{\tilde{U}}^{\tilde{d}} \neq 0.$$

Indeed, if the Boolean matrix $F_{\tilde{U}}^{\tilde{d}} \neq 0$, then there exists an input $\tilde{u} \in \tilde{U}$ and a transition $\tilde{x}_1 \to \tilde{x}_2$ such that $\hat{x}_2^T F_{\tilde{U}}^{\tilde{d}} \hat{x}_1 = 1$.

So far, it was tried to determine which fault has occurred by a single observation. The fault isolation method is easily extended for the case that multiple transitions are measured.

PROPOSITION 6.1.6 *Given two subsequent transitions $\tilde{x}_1 \to \tilde{x}_2$ and $\tilde{x}_2 \to \tilde{x}_3$ together with corresponding inputs $\tilde{u}_1$ and $\tilde{u}_2$, respectively, then the set of faults $\tilde{D}' \subseteq \tilde{D}$ that could have caused the transitions is given by*

$$\begin{aligned}\tilde{D}' &= \{\tilde{d} \in \tilde{D} \mid \hat{x}_2^T F_{\tilde{u}_1}^{\tilde{d}} \hat{x}_1 = \ \hat{x}_3^T F_{\tilde{u}_2}^{\tilde{d}} \hat{x}_2 = 1\} \\ &= \{\tilde{d} \in \tilde{D} \mid \hat{x}_2^T F_{\tilde{u}_1}^{\tilde{d}} \hat{x}_1 = 1\} \cap \{\tilde{d} \in \tilde{D} \mid \hat{x}_3^T F_{\tilde{u}_2}^{\tilde{d}} \hat{x}_2 = 1\}.\end{aligned}$$

If several subsequent observations are made, then for each of the transitions the corresponding sets of faults that could possibly have caused the transition can be computed. Next, the set of possible faults can be reduced by taking the intersection of these sets (as done for two observations in the proposition above). This is only possible if it is assumed (as already mentioned) that only one fault has occurred during the time interval of interest. This way, the actual fault that has occurred can be identified (*i.e.* isolated), if the subsequent sets of possible faults decrease until only one element is left. In some cases it is possible to directly identify the fault that caused a detection from only one observation.

PROPOSITION 6.1.7 (UNIQUENESS OF A FAULT DETECTION) *Given a fault $\tilde{d}_j$, there exists an input $\tilde{u} \in \tilde{U}$ and a corresponding transition $\tilde{x}_1 \to \tilde{x}_2$ from which the fault $\tilde{d}_j$ can be determined uniquely if*

$$\left(F_{\tilde{u}_1}^{\tilde{d}_j} \ominus \bigoplus_{i \neq j} F_{\tilde{u}_1}^{\tilde{d}_i}\right) \oplus \left(F_{\tilde{u}_2}^{\tilde{d}_j} \ominus \bigoplus_{i \neq j} F_{\tilde{u}_2}^{\tilde{d}_i}\right) \oplus ... \oplus \left(F_{\tilde{u}_k}^{\tilde{d}_j} \ominus \bigoplus_{i \neq j} F_{\tilde{u}_k}^{\tilde{d}_i}\right) \neq 0.$$

Proof. Notice, that $\hat{x}_3 := F_{\tilde{u}_1}^{\tilde{d}_j} \hat{x}_1$ represents all states to which transitions from $\tilde{x}_1$ could be caused by the fault $\tilde{d}_j$ while applying $\tilde{u}_1$. Similarly, $\hat{x}_4 := (\bigoplus_{i \neq j} F_{\tilde{u}_1}^{\tilde{d}_i}) \hat{x}_1$ is the set of states to which transitions from $\tilde{x}_1$ could be caused by one of the faults $\tilde{d}_i \in \tilde{D}$, $i \neq j$, while applying $\tilde{u}_1$. Consequently, the set difference $\hat{x}_5 := \hat{x}_3 \ominus \hat{x}_4 = (F_{\tilde{u}_1}^{\tilde{d}_j} \ominus \bigoplus_{i \neq j} F_{\tilde{u}_1}^{\tilde{d}_i}) \hat{x}_1$ are the states to which transitions from $\tilde{x}_1$ could only be caused by the fault $\tilde{d}_j$ while applying $\tilde{u}_1$. There exists an input $\tilde{u}$ and a transition $\tilde{x}_1 \to \tilde{x}_2$ that only could be caused by the fault $\tilde{d}_j$ if at least one of the matrices $(F_{\tilde{u}}^{\tilde{d}_j} \ominus \bigoplus_{i \neq j} F_{\tilde{u}}^{\tilde{d}_i}) \neq 0$. ■

6.2 Merging states

For real-time applications it is often important to keep the time to compute $\hat{x}_2^T (\mathbf{F}_{\tilde{u}_i}^{\tilde{d}}) \hat{x}_1$ as small as possible. For this purpose it may be useful to check if the number of discrete states can be reduced without losing information in the sense that the same faults still can be detected and isolated. Given a set of boundaries $\{\beta_j^i\}$ for each coordinate $i \in 1, ..., n$, it is possible to delete one (or more) boundaries or even parts of boundaries and define a new set of discrete states $\tilde{X}^*$. This merging of states can be expressed by the mapping $M \in \{0,1\}^{p^* \times p}$:

$$\hat{x}_1^* = M \hat{x}_1,$$

where $\tilde{x}_1 \in \tilde{X}$, $\tilde{x}_1^* \in \tilde{X}^*$, and $p^* < p$, with $p = \#(\tilde{X})$ and $p^* = \#(\tilde{X}^*)$ are then number of states in $\tilde{X}$ and $\tilde{X}^*$, respectively. The new state $\tilde{x}^* = i$ represents the union of old states satisfying $\{j \mid m_{ij} = 1\}$, where i and j are the integer representations of the new and old states, respectively. Each state $\tilde{x} \in \tilde{X}$ can only be included in one new state $\tilde{x}^* \in \tilde{X}^*$. Consequently, each column of M only contains exactly one '1'. Furthermore, we have that

$$\hat{x}_1 = M^T \hat{x}_1^*$$

is the Boolean representation of the set $\tilde{X}_1$ which contains all (old) states that have been merged to obtain $\tilde{x}_1^*$. Note that $MM^T = I$.

For the new set of states it is possible to construct the corresponding adjacency matrices directly from the original ones:

$$A_{\tilde{u}}^* = MA_{\tilde{u}}M^T \ominus I.$$

Indeed, given the state $\tilde{x}_1^* \in \tilde{X}^*$ we know that we are in one of the original states of the set represented by $\hat{x}_1 = M^T \hat{x}_1^*$. From $\tilde{X}_1$ transitions are possible to $\{\phi(\tilde{x}, \tilde{u}) \mid \tilde{x} \in \tilde{X}_1\}$, computed by $\hat{x}_2 = A_{\tilde{u}}\hat{x}_1$. Hence, after one transition we are in the set $\tilde{X}_2^*$ represented by $\hat{x}_2^* = M\hat{x}_2$ of new states. Since self-loops are not modelled in our setting it is not allowed that $(\hat{x}_2^*)^T \hat{x}_1^* = 1$, so all possible 1's on the diagonal of $MA_{\tilde{u}}M^T$ are removed, resulting in the Boolean substraction by the identity matrix I. For a given discrete-event model of a continuous system $\{\Sigma, Q, S\}$, merging states results in a new discrete-event model $\Sigma = \{\tilde{X}^*, \tilde{U}, \phi^*\}$ of the continuous system, where $Q^* := M \circ Q$ and $S^* := S$.

Suppose that by merging states the discrete-event model of the continuous system now is represented by $\tilde{X}^*$, $\tilde{U}$, and the adjacency matrices $\{A_{\tilde{u}}^*\}$. By construction of $A_{\tilde{u}}^*$ it can be seen that if a transition $\tilde{x}_1 \to \tilde{x}_2$ is possible, then the transition $\tilde{x}_1^* \to \tilde{x}_2^*$, $\hat{x}_i^* = M\hat{x}_i$, $i = 1, 2$ is not possible if it would result in a self-loop. This means that $\tilde{x}_1$ and $\tilde{x}_2$ are merged into the same new state $\tilde{x}_1^* = \tilde{x}_2^*$. Moreover, due to the agglomeration of states, it might happen that the transition $\tilde{x}_1^* \to \tilde{x}_2^*$ is possible, whereas the transition $\tilde{x}_1 \to \tilde{x}_2$, where $\tilde{x}_1$ and $\tilde{x}_2$ are elements of the sets represented by $M^T \hat{x}_i^*$, $i = 1, 2$, is not possible. Obviously, $\tilde{x}_1$ and $\tilde{x}_2$ must be 'neighbors' (*i.e.* $H_x(\tilde{x}_1)$ and $H_x(\tilde{x}_2)$ must be adjacent) for a transition to be possible. This reasoning indicates that information of the original discrete-event system can be lost by merging states. To summarize, there are two causes for loosing information:

1. A transition $\tilde{x}_1 \to \tilde{x}_2$ is possible, where $\tilde{x}_1$ and $\tilde{x}_2$ are merged into the same new state $\tilde{x}_1^* = \tilde{x}_2^*$ and consequently the transition $\tilde{x}_1^* \to \tilde{x}_2^*$ is not possible.

2. A transition $\tilde{x}_1^* \rightarrow \tilde{x}_2^*$ is possible, but there does not exist a transition $\tilde{x}_1 \rightarrow \tilde{x}_2$ for two adjacent discrete states $\hat{x}_1 \in M^T\hat{x}_1^*$ and $\hat{x}_2 \in M^T\hat{x}_2^*$.

This second cause indicates that, if the transition $\tilde{x}_1^* \rightarrow \tilde{x}_2^*$ is possible, then for all states in the set $M^T\hat{x}_1^*$, the corresponding transitions to adjacent states in the set $M^T\hat{x}_2^*$ also must be possible in order to represent the same transition information. This is shown in Figure 6.1.

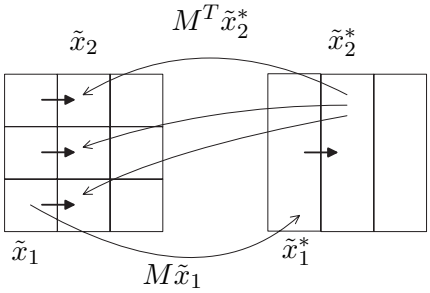

Figure 6.1: *Merging states without loss of information*

To formalize the discussion, the following definition is made.

DEFINITION 6.2.1 *The adjacency matrices $A_{\tilde{u}}^*$ and $A_{\tilde{u}}$ of the merged and original model, respectively, contain the same information if for all $\tilde{x}_1 \in \tilde{X}$ and all neighbors $\tilde{x}_2$ of $\tilde{x}_1$(represented by $N\hat{x}_1$) it holds that*

$$\hat{x}_2^T A_{\tilde{u}} \hat{x}_1 = (\hat{x}_2^*)^T A_{\tilde{u}}^* \hat{x}_1^*,$$

with $\hat{x}_i^* = M\hat{x}_i$, $i = 1, 2$.

PROPOSITION 6.2.2 *$A_{\tilde{u}}^*$ and $A_{\tilde{u}}$ contain the same information if and only if*

$$A_{\tilde{u}} = M^T A_{\tilde{u}}^* M \otimes N.$$

Proof. For the *if* part, assume that $A_{\tilde{u}} = M^T A_{\tilde{u}}^* M \otimes N$, then $\hat{x}_2^T A_{\tilde{u}} \hat{x}_1 = \hat{x}_2^T M^T A_{\tilde{u}}^* M \hat{x}_1 \otimes \hat{x}_2^T N \hat{x}_1$. Since $\hat{x}_2^T N \hat{x}_1 = 1$ in case $\tilde{x}_1$ and $\tilde{x}_2$ are neighbors, this implies that $\hat{x}_2^T A_{\tilde{u}} \hat{x}_1 = (\hat{x}_2^*)^T A_{\tilde{u}}^* \hat{x}_1^*$. For the *only if* part, assume that $\hat{x}_2^T A_{\tilde{u}} \hat{x}_1 = (\hat{x}_2^*)^T A_{\tilde{u}}^* \hat{x}_1^* = \hat{x}_2^T M^T A_{\tilde{u}}^* M \hat{x}_1$ for all $\tilde{x}_1$ and all neighbors $\tilde{x}_2$ of $\tilde{x}_1$, that is for all $\tilde{x}_1$ and $\tilde{x}_2$ for which $\hat{x}_2^T N \hat{x}_1 = 1$. If $\hat{x}_2^T N \hat{x}_1 = 0$ then the equality does not necessarily have to hold. This condition can be written as $\hat{x}_2^T A_{\tilde{u}} \hat{x}_1 \otimes \hat{x}_2^T N \hat{x}_1 = \hat{x}_2^T M^T A_{\tilde{u}}^* M \hat{x}_1 \otimes \hat{x}_2^T N \hat{x}_1$ which must hold for all $\tilde{x}_1, \tilde{x}_2 \in \tilde{X}$ (not necessarily neighbors). Clearly this can only be true if $A_{\tilde{u}} \otimes N = M^T A_{\tilde{u}}^* M \otimes N$. Since $A_{\tilde{u}} \otimes N = A_{\tilde{u}}$ the conclusion follows. ■

With respect to fault detection, it is also interesting to know if information is lost when merging the states or, stated otherwise, if a fault adjacency matrix

of the new system $F_{\tilde{u}}^{*\tilde{d}} := (A_{\tilde{u}}^{*\tilde{d}} \ominus A_{\tilde{u}}^{*})$ contains the same information as the original fault adjacency matrix $F_{\tilde{u}}^{\tilde{d}}$ for fault detection and isolation purposes. It is clear that every transition $\tilde{x}_1 \to \tilde{x}_2$ from which a fault could be detected should result in a transition $\tilde{x}_1^* \to \tilde{x}_2^*$ from which the fault can be observed as well. Conversely, for each transition $\tilde{x}_1^* \to \tilde{x}_2^*$ all corresponding transitions $\tilde{x}_1 \to \tilde{x}_2$, $\tilde{x}_i \in M^T \tilde{x}_1^*$, $i = 1, 2$ should be possible, since otherwise a fault is expected, while in reality a common transition $\tilde{x}_1 \to \tilde{x}_2$ has occurred. Clearly, the same definition and condition as discussed for the adjacency matrices $\{A_{\tilde{u}}\}$ apply for fault detection.

DEFINITION 6.2.3 *The fault adjacency matrices $F_{\tilde{u}}^{*\tilde{d}}$ and $F_{\tilde{u}}^{\tilde{d}}$ of the merged and original model, respectively, contain the same information if for all $\tilde{x}_1 \in \tilde{X}$ and all neighbors $\tilde{x}_2$ of $\tilde{x}_1$(represented by $N\hat{x}_1$) it holds that*

$$\hat{x}_2^T F_{\tilde{u}}^{\tilde{d}} \hat{x}_1 = (\hat{x}_2^*)^T F_{\tilde{u}}^{*\tilde{d}} \hat{x}_1^*,$$

with $\hat{x}_i^* = M\hat{x}_i$, $i = 1, 2$.

PROPOSITION 6.2.4 *$F_{\tilde{u}}^{*\tilde{d}}$ and $F_{\tilde{u}}^{*\tilde{d}}$ contain the same information if and only if*

$$F_{\tilde{u}}^{\tilde{d}} = M^T F_{\tilde{u}}^{*\tilde{d}} M \otimes N. \tag{6.2}$$

For the proof the same reasoning can be followed as for the proof of Proposition 6.2.2.

COROLLARY 6.2.5 *After a merging of states, the* detection and isolation *information is preserved iff*

$$F_{\tilde{u}}^{\tilde{d}} = M^T F_{\tilde{u}}^{*\tilde{d}} M \otimes N, \qquad \forall \tilde{u} \in \tilde{U}, \forall \tilde{d} \in \tilde{D}.$$

COROLLARY 6.2.6 *After a merging of states, the* detection *information is preserved iff*

$$F_{\tilde{u}}^{\tilde{D}} = M^T F_{\tilde{u}}^{*\tilde{D}} M \otimes N, \qquad \forall \tilde{u} \in \tilde{U}.$$

It is possible that after a merging of states the information is preserved only for some specific inputs. Even then, it is computationally worthwhile to use the reduced number of states for these specific inputs and to use the original discrete states for the other inputs.

6.3 Example

To illustrate the possibilities of the results presented here, they are applied to a process that is studied in (Ramkumar *et al.* 1998) and (Ramkumar *et al.* 1999). A schematic picture of the process is shown in Figure 6.2. A process liquid is pumped at a preset flow rate from one of the two storage tanks to an indirect plate heat exchanger (HE), which raises the temperature of the process liquid to a predetermined value. The process requires the liquid stream to be maintained at this temperature for a given period of time. This is achieved by the use of a holding tube followed by a temperature activated diverter valve which allows only fluid of the correct temperature to progress through the process (the remainder being rejected or returned to the feed tank). Next, the process fluid is then cooled to the lowest possible temperature by firstly exchanging otherwise wasted heat with incoming feed regeneration and subsequently by the use of externally supplied cooling water. The plate heat exchanger consists therefore of three separate but interconnected sections: feed preheat/regeneration, heating and cooling. The heating section is supplied with circulatory hot water pumped with the heating pump (HP) from an electrically heated reservoir (HU). This kind of processes are used in industry for *e.g.* continuous, high temperature short time pasteurization.

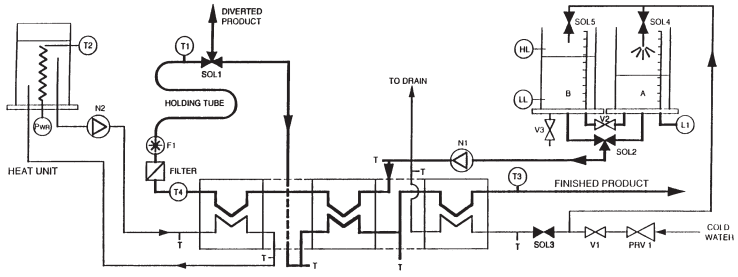

Figure 6.2: *The heat exchanger pilot plant*

Following the notation from (Ramkumar *et al.* 1998), the system under study is described by:

$$
\begin{aligned}
\text{HU} \quad & \frac{dT_2}{dt} = \frac{1}{MC_p}(PWR - \dot{m}_2 C_p (T_2 - T_5)) \\
\text{HE} \quad & \frac{dT_5}{dt} = (1.63\frac{N_2}{0.35})^{0.078}(0.2394T_2 - 13.119)\frac{dT_2}{dt} \\
& \frac{dT_4}{dt} = (0.87\frac{N_2}{0.5})^{0.085}(0.98T_2)\frac{dT_2}{dt} \\
\text{HP} \quad & \dot{m}_2 = \rho_w K_1 (1173N_2 - 169.9)
\end{aligned}
$$

where the following notation is used:

T_2	Heating water inlet temp. $[^\circ C]$ to HE
T_4	Product temp. $[^\circ C]$ at exit HE
T_5	Heating water exit temp. $[^\circ C]$ from HE
PWR	Heat input $[W]$
N_2	Speed of pump N_2 $[1/\min]$
M	Mass of water in heating tank $= 4.67$ $[kg]$
C_p	Heat capacity of water $= 4180$ $[J/kg^\circ C]$
$\dot{m}_2$	Mass flow rate $[kg/\sec]$ of heating water
ρ_w	Density of water $= 1000$ $[kg/m^3]$
K_1	Convert $ml/\min$ to $m^3/\sec$: $10^{-6}/60$

The state-vector is given by $x = [T_2, T_4, T_5]^T$ and the input-vector is $u = [PWR, N_2]^T$. The following possible faults are considered:

1. Failure of the heater ($PWR = 0$)
2. Pump N_2 is malfunctioning ($N_2 = 0$)
3. Leakage in hot-water circulation ($\dot{m}_2$ reduces with 50%)

These faults are denoted $\tilde{d}_1, \tilde{d}_2, \tilde{d}_3$, respectively. The system, including the possible faults, is modelled by

$$\dot{x}_1 = \frac{-\rho_w K_1}{M}((1-\tilde{d}_2)1173u_2 - 169.9)(1-0.5\tilde{d}_3)(x_1 - x_2) + \frac{(1-\tilde{d}_1)u_1}{Mc_p}$$

$$\dot{x}_2 = (1.63\frac{(1-\tilde{d}_2)u_2}{0.35})^{0.078}(0.2397x_1 - 13.119)\cdot \left(\frac{-\rho_w K_1}{M}((1-\tilde{d}_2)1173u_2 - 169.9)(1-0.5\tilde{d}_3)(x_1 - x_2) + \frac{(1-\tilde{d}_1)u_1}{Mc_p}\right)$$

$$\dot{x}_3 = (0.87\frac{(1-\tilde{d}_2)u_2}{0.5})^{0.085}(0.98x_1)\cdot \left(\frac{-\rho_w K_1}{M}((1-\tilde{d}_2)1173u_2 - 169.9)(1-0.5\tilde{d}_3)(x_1 - x_2) + \frac{(1-\tilde{d}_1)u_1}{Mc_p}\right)$$

Next, the various adjacency matrices have to be computed. Since the input set $\mathcal{U}$ is not a discrete set, the input space is discretized. This is done in the same manner as for the state space, *i.e.* by choosing boundaries that generate the discrete-input regions (see Chapter 3). To check for a given discrete input, whether a transition is possible amounts to verifying if there exists an input in the corresponding hypercube such that the particular transition is possible. The following boundaries are chosen in accordance with (Ramkumar *et al.* 1998).

Boundaries				
T_2	T_4	T_5	PWR	N_2
55	45	48	50	0.20
58	49	50	120	0.25
60	52	54	150	0.30
62	54	57	200	0.40
64	57	59	270	0.50

From this it follows that there are 64 discrete states and 16 discrete inputs. The adjacency matrices $A_{\tilde{u}_i}$ and $A_{\tilde{u}_i}^{\tilde{d}_j}$, $1 \leq i \leq 16$, and $1 \leq j \leq 3$ are generated automatically by the procedure as explained in Chapter 3 and are used to construct the fault adjacency matrices $F_{\tilde{u}_i}^{\tilde{d}_j} \in \{0,1\}^{64\times 64}$ as defined in Proposition 6.1.2. To see if a certain fault $\tilde{d}_j$ is detectable with an input $\tilde{u}$ it follows from Proposition 6.1.5 that $F_{\tilde{u}_i}^{\tilde{d}_j} \neq 0$. In the following table a '1' denotes that $F_{\tilde{u}_i}^{\tilde{d}_j} \neq 0$ and a '0' denotes that $F_{\tilde{u}_i}^{\tilde{d}_j} = 0$.

$F_{\tilde{u}_i}^{\tilde{d}_j}$	$\tilde{u}_1$	$\tilde{u}_2$	$\tilde{u}_3$	$\tilde{u}_4$	$\tilde{u}_5$	$\tilde{u}_6$	$\tilde{u}_7$	$\tilde{u}_8$
$\tilde{d}_1$	1	1	1	1	1	1	1	1
$\tilde{d}_2$	0	0	0	0	0	0	0	0
$\tilde{d}_3$	0	0	0	0	0	0	0	0
$F_{\tilde{u}_i}^{\tilde{d}_j}$	$\tilde{u}_9$	$\tilde{u}_{10}$	$\tilde{u}_{11}$	$\tilde{u}_{12}$	$\tilde{u}_{13}$	$\tilde{u}_{14}$	$\tilde{u}_{15}$	$\tilde{u}_{16}$
$\tilde{d}_1$	1	1	1	1	1	1	1	1
$\tilde{d}_2$	1	1	0	0	1	1	1	0
$\tilde{d}_3$	1	1	0	0	1	1	1	0

From this it can be seen that $F_{\tilde{u}_i}^{\tilde{d}_j} \neq 0$, for $j = 1, 2, 3$ implying that for all possible faults there exists an input and a transition resulting in a fault detection. Furthermore, checking the condition stated in Proposition 6.1.7 shows that only for fault $\tilde{d}_1$ there exists a transition such that $\tilde{d}_1$ uniquely can be determined. Also it can be shown that *e.g.* merging the discrete-states $(i,1,4),(i,2,4),(i,3,4)$ and $(i,4,4)$ to $(i,1,4)$ for $i = 1,2,3,4$ (which means a reduction from 64 to 52 states) does not affect the detection and isolation properties of $\tilde{d}_2$ and $\tilde{d}_3$. This means that it holds that if $\tilde{d}_2$ and $\tilde{d}_3$ can be detected by a transition between the original states with a specific input $\tilde{u}$, then $\tilde{d}_2$ and $\tilde{d}_3$ will also be detected by a transition between the merged states with the input $\tilde{u}$. Merging the states $(1,i,1),(2,i,1),(3,i,1),(4,i,1),(1,i,2),(2,i,2),(3,i,2)$, and $(4,i,2)$ to the new state $(i,1,1)$ for $i = 1,2,3,4$ (which is a reduction from 64 states to 36 states) does not affect the detection properties of $F_{\tilde{u}_1}^{\tilde{d}}, F_{\tilde{u}_5}^{\tilde{d}}, F_{\tilde{u}_{11}}^{\tilde{d}}$, and $F_{\tilde{u}_{16}}^{\tilde{d}}$. This means that when applying one of the inputs $\tilde{u}_1, \tilde{u}_5, \tilde{u}_{11}$, or $\tilde{u}_{16}$

the merged states can be used for checking if a fault has occurred when the transition $\tilde{x}_1^* \rightarrow \tilde{x}_2^*$ is observed.

In (Ramkumar *et al.* 1998) and (Ramkumar *et al.* 1999) the process under study is discussed in more detail. Also, experimental results are presented using a fault detection scheme similar to the one discussed in this section.

6.4 Notes and references

This chapter is based on material presented in (Philips *et al.* 1999*a*). A similar approach for detecting faults as discussed here is used in (Ramkumar *et al.* 1998) where also a case study is included. The main difference is that we use a different representation of the discrete-event models (Boolean matrices) that eases the analysis of fault detection and isolation properties.

Although the merging of states also applies to the reduction of the number of discrete states for the discrete-event model itself, it is treated in this chapter (rather than in Chapter 3) because it is particularly relevant to fault detection. In comparison with adjacency matrices, the fault adjacency matrices in general contain little information (there are a few elements equal to '1'), which makes the merging more effective.

In (Förstner *et al.* 2000) discrete-event models of continuous systems with asynchronous input and state events are derived for the purpose of fault diagnoses (see the references therein). A timed discrete-event representation of a discretely-observed (*i.e.* quantized) system which is modelled by a semi-Markov process, is the basis for fault diagnoses in (Lunze 2000). In (Lunze and Schiller 1999) fault detection for quantized systems is performed by examining the probabilities that a certain effect has occurred due to a certain cause. A fault detection method for hybrid systems that relies on the mixed logical dynamical modelling framework is discussed in (Bemporad *et al.* 1999). In (Cassandras and Lafortune 1999) a fault detection method for discrete-event systems is discussed which is based on the construction of an observer for the discrete-event system. From the measured sequence of symbols (outputs), the observer deduces the unmeasured transitions (the faults).

For other model based fault diagnoses methods, for example (Frank 1990, García and Frank 1997) give a survey of various linear and nonlinear observer-based fault diagnoses methods.

Chapter 7

Conclusions and recommendations

In this thesis, it is shown how to construct a discrete-event model of a continuous system described by a set of differential equations and based on a discretized state space. Moreover, the resulting model is used for control and fault detection purposes. In this respect, the following contributions have been made.

7.1 Contributions

7.1.1 Discrete-event models of continuous systems

Discrete-event models of continuous systems can simply serve as models of continuous systems that are observed by discrete sensors or that have to be controlled by a discrete-event controller, but can also be interpreted as coarse abstractions of continuous systems that can be used for designing high-level controllers in a hierarchical control scheme.

The definition of a discrete-event model of a continuous system, as it is used in this thesis, is given in a manner that is general enough to allow modelling techniques other than the one presented in this thesis. In particular, the *mappings from the discrete to the continuous domain* (*i.e.* the state and input space) are defined on (pieces of) trajectories rather than on points in the state and input space. This has proven to be necessary for constructing a mathematically correct model in the sense that otherwise the situation can occur where either no discrete state can be assigned to a point in continuous space or the resulting discrete-event model may be incorrect. The mapping used in this thesis has the natural interpretation that a boundary in state space separating and belonging to two hypercubes (that correspond with discrete

states) actually has to be crossed by the continuous-time trajectory before a new discrete state is observed. Moreover, this mapping has the advantage that more advanced constructions of discrete states are possible, which can be used for obtaining more precise discrete-event models (*i.e.* with less spurious solutions). However, this will most certainly be at the cost of a significantly larger number of discrete states and more elaborate computations for generating the discrete-event model.

The choice of the discrete states (which are based on hypercubes in the continuous space) for our models allows a computationally effective modelling method. This method is based on searching for positive (or negative) derivatives $f^r(x,u)$ for a coordinate r along the boundary plane separating two hypercubes. Due to Nagumo's theorem (Nagumo 1942) the existence of a positive (or negative) derivative is necessary and sufficient for a 'positive' (or 'negative') transition to be possible. It is shown that exploiting specific properties of the continuous systems such as linearity or *sparsity* can lead to a significant *reduction* of the number and complexity of the computations. For linear systems, the optimizations that are necessary for determining the possible transitions in the general case do not have to be performed. Linearity allows us to assign possible transitions to a whole range of states instead of on a one-by-one basis. Sparsity of the continuous system means that particular coordinates of the continuous state vector are not influenced by all other states or inputs. As a result, the original continuous system can be broken down into (smaller) subsystems for which the discrete-event models have to be computed. Finally, the overall discrete-event system can be easily constructed from these collections of discrete-event sub-models. This allows a modular approach for modelling large systems.

For *continuous systems with outputs* not equal to (a subset of coordinates of) the continuous state, it may be necessary to define additional discrete states other than those induced by the partitioning of the state space. The reason is that otherwise it might occur that for a given discrete state more than one discrete output is valid for the corresponding hypercube in the continuous state space. As this is impossible for our automaton, additional discrete states are needed to associate only one discrete output to each discrete state. However, it is shown that a continuous system for which the output is a linear combination of elements of the state vector can be transformed such that the standard modelling algorithm applies. This leads to a reduction of complexity and computation-time with regard to the general approach.

7.1.2 State reconstruction

For continuous systems with discrete measurements, the actual continuous state can be reconstructed on the basis of the exact knowledge of the time

and the value of the recorded component of the continuous state whenever a measurement is made. When the input is known, the differential equations exactly describe the continuous plant, and measurement noise and disturbances are absent, a multipoint boundary value problem has to be solved to reconstruct the continuous state. For linear, time-invariant systems, the necessary integration can be done analytically, for the linear time-varying case the transition matrix has to be determined by numerical integration. In both cases, the initial state is found by solving a set of linear equations. For the *nonlinear* case, this is no longer true and Newton's method is exploited for solving the multipoint boundary value problem. Questions related to observability are formalized and turn out to be difficult for discretely observed systems.

The discrete state can be reconstructed from measured discrete outputs or transitions. To do so, a set is constructed after each measurement, that contains all discrete states from which transitions are possible resulting in the measured output, and that could be reached in accordance with the previously recorded measurement. To allow transition measurement in our automaton-based framework, a non-standard output map is introduced that generates an output given a new and an old discrete state.

7.1.3 Control

The discrete-event models that result from the modelling algorithm can be used for controller synthesis. Some first steps are made for systematically designing control strategies for nonlinear systems on the basis of discretized information. For these strategies, besides the obtained discrete-event model, also the knowledge that the underlying system is continuous is used.

The control goals that are considered are the *reachability problem* and the *stabilization problem*. The former one involves controlling the continuous state from an initial discrete state (*i.e.* the corresponding hypercube) to a target set of discrete states. The second problem additionally requires to keep the continuous state in the target set once it has entered.

Two situations are distinguished: the case where it actually is measured that the continuous state trajectory *reaches* a boundary in state space, and the situation that only the *crossing* of such a boundary is observed. In the first case it is investigated whether a possible undesired transition can be prevented by choosing a suitable input. This requires instantaneous control action. For the second case, an occurred undesired transition is tried to be reversed (corrected) if possible. These ideas are formalized by the definition of '*preventing inputs*' and '*correcting inputs*', respectively. Together with the '*moving inputs*' which assure the movement of the state trajectory into a particular direction in the state space, these types of inputs form the building blocks of the proposed control strategies.

The first controller design method, the '*forceable state-transition*' control strategy, is based on the construction of '*forceable*' transitions. These are transitions that can be guaranteed to occur by choosing suitable inputs. Since it is difficult to determine whether a transition is forceable, a sufficient condition is given. The idea is to prevent (or correct) undesired transitions with inputs that are moving towards a particular desired transition. From all transitions that satisfy this sufficient condition (and are consequently forceable), a graph is constructed. For a given initial state and a desired target state, all suitable paths can be found from this graph by a constructive algorithm, thereby solving the reachability problem.

The principle of the second and third synthesis methods, the '*forceable set-transition*' and the '*invariant sets*' control strategies is based on the notion of a *discretely controlled invariant set*. For such a set it is certain that always inputs can be chosen that prevent the continuous trajectory from leaving this set. Related to this are the so-called Γ-*controlled invariant sets* for which the trajectory can only leave the set via Γ. The idea is now to compute a sequence of nested Γ-controlled invariant sets (for the reachability problem; for the stabilization problem controlled invariant sets are required) such that the first set contains the desired discrete states and the last one contains the initial state. This sequence of nested sets is constructed with the property that always inputs can be chosen that will move the continuous trajectory to a predecessor in the sequence, until eventually the target set is reached. The difference between the two synthesis methods is the construction of the controlled invariant sets. The 'forceable set-transition' method deletes discrete states that obstruct the Γ-controlled invariance, whereas the 'forceable set-transition' method includes discrete states to which the transition cannot be prevented. For the stabilization problem an algorithm is given to compute the largest controlled invariant set contained in the target set. If this set does not exist (if it is the empty set), then the stabilization problem is not solvable. The best we can do in such a situation, is to compute the smallest controlled invariant set containing the target set.

Two problems can be pointed out for the proposed strategies. First, even though for the prevention (or correction) of undesired transitions always inputs are chosen for which it is certain that the continuous state moves towards a desired set of discrete states (*i.e.* moving inputs) it can not be guaranteed that two subsequent inputs will move the state trajectory in the same direction. As a result cyclic behavior inside one of the invariant sets of the sequence may occur, which prevents improvement of the continuous state towards the desired objective. Second, the Assumption 3.2.1 made for the discrete-event modelling method might be violated by the designed controllers. Fortunately, by performing additional computations and cumbersome book-keeping, this last problem can be overcome. Verification or simulation has to performed for

checking the presence of the phenomena described in the first problem.

7.1.4 Fault detection and isolation

Discrete-event models of continuous systems can be exploited for the detection and isolation of faults. For these purposes, each fault that is to be expected results in separate discrete-event models. Next, by observing transitions of the real plant it is deduced by comparing the discrete-event models, if a measured transition can be caused by the nominal plant (*i.e.* without faults) or only can result from a fault that has occurred. Besides detecting a fault, it is also important to isolate the fault, which means that it has to be determined which fault actually happened. *Conditions* are given from which it can be seen if, for a given fault, there exist an input and a transition such that the particular fault can be detected at all (*detectability of a fault*) or even isolated by only observing one transition (*uniqueness of a fault detection*).

By *merging* two or more discrete states into one new state, the total number of discrete states can be reduced resulting in faster computations for on-line implementation of the fault detection scheme. However, it is only reasonable to merge discrete states if no information is lost. For example, transitions that are possible with the old set of states still should be possible with the new set, and conversely, transitions between new states should also be possible for the corresponding old discrete states. Explicit *expressions* are given that indicate that (*e.g.* transition or fault detection) information is *preserved* after merging states.

7.1.5 Explicit computations via Boolean vectors and matrices

Throughout this thesis, Boolean matrices and vectors are employed for computational and didactical reasons. One advantage of Boolean vectors is that a single vector can be used for representing a set. In this way, specific sets can be defined explicitly in terms of operations with Boolean matrices on discrete states or sets. All the controller design and fault detection schemes are explicitly stated in terms of Boolean vectors and matrices and can, in principle, be implemented directly. However, it still has to be investigated what the computationally most efficient way is for numerical implementation of the algorithms presented in the thesis, but it is shown that the Boolean representation is an effective method for analyzing, describing and proving a variety of properties.

7.2 Recommendations for further research

Roughly, I propose three directions that can be followed for further research. First, the obtained results based on the modelling method presented in this

thesis allow, of course, improvement and extension. Second, more practically, the methods proposed in this thesis can be tested and verified on real plants where noise and disturbances can no longer be neglected. It should be investigated which adjustments have to be made for implementation of the proposed concepts, preferably verified by experiments. Third, other discrete-event modelling methods can be investigated, possibly allowing more 'precise' models or the integration of additional information such as time.

Concerning the first research direction, new control strategies can be invented. A new control goal that can be relevant is to prevent the continuous state from reaching certain regions in state space (*e.g.* for safety reasons). The proposed strategies can be extended or adapted to satisfy this specification. Furthermore, transitions between hypercubes that are not adjacent can be included in the discrete-event models to overcome the problems related to points that belong to three or more hypercubes. The consequences of this inclusion deserve further attention. Also, the use of discrete-event models for hierarchical control is an important topic for future research.

With respect to the second issue, the presence of disturbances has to be incorporated in the modelling method. This can be achieved by adding variables to the continuous model (representing the disturbance). For each such variable the upper and lower bounds have to be determined. Doing so, the discrete-event model can be constructed by treating these variables as additional discretized 'inputs' that are always present. One might also think of pursuing a stochastic approach. Another possible subject of investigation is the dependence of the discrete-event model on the separating hyperplane between positive and negative derivatives of a state component indicating possible transitions. If these transitions are highly sensitive to the position of such equilibrium planes, small modelling errors lead to erroneous discrete-event models and consequently to possibly wrong control actions or detections of faults. Practical situations ask for an accurate treatment of this phenomenon.

Finally, for the third research direction other types of discrete states can be used as a basis for the discrete-event model. A couple of possibilities can be given.

- To overcome the problems with points that belong to three or more hypercubes, a partitioning of the state space can be used from which these points (*i.e.* the faces to which they belong) are excluded. The faces are then isolated and can be treated as separate discrete states.
- Or, as mentioned before, one might use ordered pairs of hypercubes as discrete states.
- Also, the boundary planes separating the hypercubes in our setting can serve as discrete states. For determining possible transitions for these

types of discrete states in an effective way, research can focus on restricted classes of systems, such as linear systems.

Time consuming integration can be avoided by using discrete-time models. Furthermore, time information such as minimal and maximal times for transitions to occur could be included in these discrete-event models. This information is useful for time-optimal control strategies and for further improving the fault detection scheme.

Many of the ideas and techniques presented in this thesis might be useful in these settings as well and contribute to the development of a mature theory and technology of using discrete-event models of continuous plants for control and fault detection in practice.

Bibliography

Alur, Rajeev, Henzinger, Thomas A. and Sontag, Eduardo D., Eds.) (1996). *Hybrid Systems III: Verification and Control*. Vol. 1066 of *Lecture Notes in Computer Science*. Springer-Verlag.

Angelis, Georgo and Patrick Philips (1999). Piecewise-constant control for systems with discrete measurements. In: *Proceedings of the 1999 American Control Conference (ACC'99)*. San Diego, USA. pp. 2965–2966.

Antsaklis, Panos, Kohn, Wolf, Lemmon, Michael, Nerode, Anil and Sastry, Shankar, Eds.) (1999). *Hybrid Systems V*. Vol. 1567 of *Lecture Notes in Computer Science*. Springer-Verlag.

Antsaklis, Panos, Kohn, Wolf, Nerode, Anil and Sastry, Shankar, Eds.) (1993). *Hybrid Systems II*. Vol. 999 of *Lecture Notes in Computer Science*. Springer-Verlag.

Antsaklis, Panos, Kohn, Wolf, Nerode, Anil and Sastry, Shankar, Eds.) (1997). *Hybrid Systems IV*. Vol. 1273 of *Lecture Notes in Computer Science*. Springer-Verlag.

Aubin, J.P. and A. Cellina (1984). *Differential Inclusions*. Grundlehren der Mathematischen Wissenschaften. 264 ed.. Springer-Verlag.

Bemporad, Alberto and Manfred Morari (1999). Control of systems integrating logic, dynamics, and constraints. *Automatica* **35**(3), 407–428.

Bemporad, Alberto, Domenico Mignone and Manfred Morari (1999). Moving horizon estimation for hybrid systems and fault detection. In: *Proceedings of the 1999 American Control Conference (ACC'99)*. San Diego, California. pp. 2471–2475.

Bett, Christopher J. and Michael D. Lemmon (1999). Bounded amplitude performance of switched LPV systems with applications to hybrid systems. *Automatica* **35**, 491–503.

Blanchini, F. (1999). Set invariance in control. *Automatica* **35**, 1747–1767.

Booth, Tylor L. (1967). *Sequential Machines and Automata Theory*. John Wiley and Sons, Inc.. New York.

Branicky, Michael S., Vivek S. Borkar and Sanjoy K. Mitter (1998). A unified framework for hybrid control: Model and optimal control theory. *IEEE Transactions on Automatic Control* **43**(1), 31–45.

Bruinsma, U.B.D.M.R. (1997). State-event discrete modelling of non-linear batch plants. Master's thesis. Eindhoven University of Technology. NR-1984.

Cassandras, Christos G. and Stéphane Lafortune (1999). *Introduction to Discrete Event Systems*. Kluwer Academic Publishers.

Cassandras, Christos G., Stéphane Lafortune and Geert Jan Olsder (1995). Introduction to the modelling, control and optimization of discrete event systems. Technical report. Faculty of Technical Mathematics and Informatics, Delft University of Technology. Delft, the Netherlands. Report 95-22.

Cury, José E.R., Bruce H. Kroch and Toshihiko Niinomi (1998). Synthesis of supervisory controllers for hybrid systems based on approximating automata. *IEEE Transactions on Automatic Control* **43**(4), 564–568.

David, R. and H. Alla (1994). Petri nets for modelling of dynamic systems. *Automatica* **30**(2), 175–202.

De Bruin, D. and P.P.J. van den Bosch (1998). Measurement of the lateral vehicle position with permanent magnets. In: *Proceedings of the IFAC Workshop on Intelligent Components for Vehicles (IVC'98)*. Seville, Spain. pp. 9–14.

Delchamps, David F. (1988). The 'stabilization' of linear systems with quantized feedback. In: *Proceedings of the 27th IEEE Conference on Decision and Control*. Austin, Texas. pp. 405–410.

Delchamps, David F. (1989). Extracting state information from a quantized output record. *Systems & Control Letters* **13**, 365–372.

Delchamps, David F. (1990). Stabilizing a linear system with quantized state feedback. *IEEE Transactions on Automatic Control* **35**(8), 916–924.

Doberkat, Ernst-Erich (1981). *Stochastic Automata: Stability, Nondeterminism, and Prediction*. Vol. 113 of *Lecture Notes in Computer Science*. Springer-Verlag.

Drakunov, Sergey, Murat Doğruel and Ümit Özgüner (1993). Sliding mode control in hybrid systems. In: *Proceedings of the 1993 International Symposium on Intelligent Control*. Chicago, Illinois, USA. pp. 186–189.

Förstner, Dirk and Jan Lunze (n.d.). Discrete-event models of quantised systems for diagnosis. *Accepted for publication in the International Journal of Control*.

Förstner, Dirk, Merten Jung and Jan Lunze (2000). Discrete-event abstraction of quantized systems with asynchronous input and state events. In: *Proceedings of the 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems* (S. Engell, S. Kowalewski and J. Zaytoon, Eds.). Dortmund, Germany. pp. 55–60.

Frank, P.M. (1990). Fault diagnosis in dynamic systems using analytical and knowledge based reduncancy - a survey and new methods. *Automatica* **26**(3), 459–474.

Freedman, David (1971). *Markov Chains*. Holden-Day Series in Probability and Statistics. Holden-Day. San Francisco.

García, E. Alcorta and P.M. Frank (1997). Deterministic nonlinear observer-based approaches to fault diagnoses: A survey. *Control Engineering Practice* **5**(5), 663–670.

Grossman, Robert L., Nerode, Anil, Ravn, Anders P. and Rischel, Hans, Eds.) (1993). *Hybrid Systems*. Vol. 736 of *Lecture Notes in Computer Science*. Springer-Verlag.

Habets, L.C.G.J.M. and J.H. van Schuppen (2000). A control problem for affine dynamical systems on a full-dimensional simplex. Technical report. CWI, PNA-R0017.

Heemels, W.P.M.H., J.M. Schumacher and S. Weiland (2000). Linear complementarity systems. *SIAM J. Appl. Math.* **60**(4), 1234–1269.

Heemels, W.P.M.H., R.J.A. Gorter, A. van Zijl, P.P.J. van den Bosch, S. Weiland, W.H.A. Hendrix and M.R. Vonder (1999). Asynchronous measurement and control: A case study on motor synchronization. *Control Engineering Practice* **7**, 1467–1482.

Henzinger, Thomas A. and Sastry, Shankar, Eds.) (1998). *Hybrid Systems: Computation and Control*. Vol. 1386 of *Lecture Notes in Computer Science*. Springer.

Heymann, Michael and Feng Lin (1998). Discrete-event control of nondeterministic systems. *IEEE Transactions on Automatic Control* **43**(1), 3–17.

Hopcroft, John E. and Jeffrey D. Ullman (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Series in Computer Science. Addison-Wesley.

Hsu, C.S. (1985). A discrete method of optimal control based upon the cell state space concept. *Journal of optimization theory and applications* **64**(4), 547–569.

Kalman, R.E., P.L. Falb and M.A. Arbib (1969). *Topics in Mathematical System Theory*. International Series in Pure and Applied Mathematics. McGraw-Hill.

Keller, Herbert B. (1968). *Numerical Methods for Two-Point Boundary-Value Problems*. Blaisdell. Waltham.

Kim, Ki Hang (1982). *Boolean Matrix Theory and Applications*. Marcel Dekker.

Kornoushenko, E.K. (1975). Finite-automaton approximation to the behaviour of continuous plants. *Automation and Remote Control* pp. 2068–2074.

Kowalewski, S., S. Engell, J. Preußig and O. Stursberg (1999). Verification of logic controllers for continuous plants using timed condition/event-system models. *Automatica* **35**, 505–518.

Lemmon, M.D. and P.J. Antsaklis (1993). Event identification in hybrid control systems. In: *Proceedings of the 32nd Conference on Decision and Control*. San Antonio, Texas, USA. pp. 2323–2328.

Lichtenberg, G. and J. Lunze (1997). Observation of qualitative states by means of a qualitative model. *International Journal of Control* **66**(6), 885–903.

Lin, F. and W.M. Wonham (1988). On observability of discrete event systems. *Information Sciences* **44**(3), 173–198.

Lunze, J. (1994). Qualitative modelling of linear dynamical systems with quantized state measurements. *Automatica* **30**(3), 417–431.

Lunze, J. (2000). Diagnosis of quantised systems by means of timed discrete-event representations. In: *Hybrid Systems: Computation and Control* (N. Lynch and B. Krogh, Eds.). Springer-Verlag. pp. 258–271.

Lunze, Jan (1995). Stabilization of nonlinear systems by qualitative feedback controllers. *International Journal of Control* **62**(1), 109–128.

Lunze, Jan (1996). The existence of a discrete-event representation of linear continuous-variable systems with quantised state measurements. In: *Proceedings of the 13th IFAC Triennial World Congress*. San Fransisco, USA. pp. 503–508.

Lunze, Jan (1999). A timed discrete-event abstraction of continuous-variable systems. *International Journal of Control* **72**(13), 1147–1164.

Lunze, Jan and Frank Schiller (1999). An example of fault diagnosis by means of probabilistic logic reasoning. *Control Engineering Practice* **7**, 271–278.

Lunze, Jan, Bernhard Nixdorf and Henrik Richter (1997). Hybrid modelling of continuous-variable systems with application to supervisory control. In: *Proceedings of the European Control Conference ECC' 97*. Brussels.

Lunze, Jan, Bernhard Nixdorf and Jochen Schröder (1999*a*). Deterministic discrete-event representations of linear continuous-variable systems. *Automatica* **35**, 395–406.

Lunze, Jan, Bernhard Nixdorf and Jochen Schröder (1999*b*). A unified approach to the representation of discrete-time and discrete-event quantised systems. In: *Proceedings of the European Control Conference ECC'99*. Karlsruhe, Germany.

Lygeros, John, Datta N. Godbole and Shankar Sastry (1998). Verified hybrid controllers for automated vehicles. *IEEE Transactions on Automatic Control* **43**(4), 522–539.

Lynch, Nancy and Krogh, Bruce, Eds.) (2000). *Hybrid Systems: Computation and Control*. Vol. 1790 of *Lecture Notes in Computer Science*. Springer.

Meyer, Gunter H. (1973). *Initial Value Methods for Boundary Value Problems*. Vol. 100 of *Mathematics in Science and Engineering*. Academic Press. New York.

Moor, T. and J. Raisch (1999). Supervisory control of hybrid systems within a behavioural framework. *System & Control Letters* **38**, 157–166.

Nagumo, Mitio (1942). über die lage der integralkurven gewöhnlicher differentialgleichungen. *Proceedings of the Physico-Mathematical Society of Japan* **24**, 551–559.

Niinomi, Toshihiko, Bruce H. Krogh and José E.R. Cury (1995). Synthesis of supervisory controllers for hybrid systems based on approximating automata. In: *Proceedings of the 34th Conference on Decision and Control*. New Orleans, LA. pp. 1461–1466.

Nijmeijer, H. and A.J. Van der Schaft (1990). *Nonlinear Dynamical Control Systems*. Springer.

Olsder, Gert Jan (1993). On structural properties of min-max systems. Technical Report 93-95. Delft University of Technology.

Özveren, Cüneyt M. and Alan S. Willsky (1990). Observability of discrete event dynamic systems. *IEEE Transactions on Automatic Control* **35**(7), 797–806.

Papa, Mauricio, Heng-Ming Tai and Sujeet Shenoi (1997). Cell mapping for controller design and evaluation. *IEEE Control Systems Magazine* **17**(2), 52–65.

Passino, Kevin M., Anthony N. Michel and Panos J. Antsaklis (1994). Lyapunov stability of a class of discrete event systems. *IEEE Transactions on Automatic Control* **39**(2), 269–279.

Perko, Lawrence (1991). *Differential Equations and Dynamical Systems*. Vol. 7 of *Texts in Applied Mathematics*. Springer-Verlag.

Philips, P., K.B. Ramkumar, K.W. Lim, H.A. Preisig and M. Weiss (1999*a*). Automaton-based fault detection and isolation. *Computers & Chemical Engineering* **23 Supplement**, S215–S218.

Philips, Patrick and Martin Weiss (2000). Hierarchical discrete-event models of continuous systems. In: *Proceedings of IMACS Symposium on Mathematical Modelling (MATHMOD 3)*. Vienna, Austria. pp. 437–440.

Philips, Patrick, Martin Weiss and Heinz A. Preisig (1999*b*). Control based on discrete-event models of continuous systems. In: *Proceedings of the European Control Conference (ECC'99)*. Karlsruhe, Germany.

Philips, Patrick, Martin Weiss and Heinz A. Preisig (1999*c*). A design strategy for discrete control of continuous systems. In: *Proceedings of the 1999 American Control Conference (ACC'99)*. San Diego, USA. pp. 2097–2101.

Philips, Patrick, Udo Bruinsma, Martin Weiss and Heinz A. Preisig (1997). A mathematical approach to discrete-event dynamic modelling of hybrid systems. In: *Proceedings IFAC Symposium on AI in Real-Time Control*. Kuala Lumpur, Malaysia.

Philips, P.P.H.H. and M. Weiss (1998). State reconstruction from discrete partial measurement data. In: *Proceedings 5-Th IFAC Symposium on Dynamics and Control of Process Systems*. Corfu, Greece. pp. 533–537.

Phillips, Anthony M. and Masayoshi Tomizuka (1995). Multirate extimation and control under time-varying data sampling with applications to information storage devices. In: *Proceedings of the 1995 American Control Conference*. Seatle, Washingthon. pp. 4151–4155.

Preisig, Heinz A. (1989). The application of finite automata theory to sequential control of chemical processes. In: *IFAC Dynamics and Control of Chemical Reactors (DYCORD+'89)*. Maastricht, the Netherlands.

Preisig, Heinz A. (1992). Discrete-event controlled systems in the chemical processing industry. In: *Proceedings of the IFAC Symposium on Dynamics and Control of Chemical Reactors, Distillation Columns and Batch Processes (DYCORD+92)*. College Park, Maryland, USA. pp. 277–282.

Preisig, Heinz A. (1993). First principle based event-discrete dynamic system models and EDD controller synthesis. In: *Proceedings of the IFAC 93 Worldcongress (Volume 4)*. Sydney, Australia. pp. 207–210.

Preisig, Heinz A. (1996*a*). Event-discrete modelling of manufactorings systems: Reduction of the state space. In: *Proceedings of the 2th International Conference on Computer Integrated Manufactoring in Process Industries (I-CIMPRO '96)*. Eindhoven, the Netherlands. pp. 434–443.

Preisig, Heinz A. (1996*b*). A mathematical approach to discrete-event dynamic modelling of hybrid systems. *Computers & Chemical Engeneering* **20**, S1301–S1306.

Preisig, Heinz A., Marc J.H. Pijpers and Martin Weiss (1997). A discrete modelling procedure for continuous processes based on state-discretisation. In: *Proceedings of the 2th IMACS Symposium on Mathematical Modelling*. Vienna. pp. 189–194.

Puri, Anuj, Vivek Borkar and Pravin Varaiya (1996). ϵ-aproximation of differential inclusions. In: *Hybrid Systems III: Verification and Control* (R. Alur, T. A. Henzinger and E. D. Sontag, Eds.). Springer-Verlag. pp. 362–376.

Raisch, J. and A. Itigin (2000). Synthesis of hierarchical process control systems based on sequential aggregation. In: *Proceedings of the IMACS Symposium on Mathematical Modelling (3rd MATHMOD)* (I. Troch and F. Breitenecker, Eds.). Vienna, Austria. pp. 385–390.

Raisch, Jörg (1993). Control of continuous plants by symbolic output feedback. In: *Hybrid Systems II* (Panos Antsaklis, Wolf Kohn, Anil Nerode and Shankar Sastry, Eds.). pp. 370–390.

Raisch, Jörg, Alexander Itigin and Thomas Moor (2000). Hierarchical control of hybrid systems. In: *Proceedings of the 4th International Conference on Automation of Mixed Processes: Hybrid Dynamical Systems* (S. Engell, S. Kowalewski and J. Zaytoon, Eds.). Dortmund, Germany. pp. 67–72.

Raisch, Jörg and Siu D. O'Young (1998). Discrete approximation and supervisory control of continuous systems. *IEEE Transactions on Automatic Control* **43**(4), 569–573.

Ramadge, Peter J. (1990). On the periodicity of symbolic observations of piecewise smooth discrete-time systems. *IEEE Transactions on Automatic Control* **35**(7), 807–813.

Ramadge, Peter J. G. and Murray W. Wonham (1987*a*). On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization* **25**(3), 637–659.

Ramadge, Peter J. G. and Murray W. Wonham (1987*b*). Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization* **25**(1), 206–230.

Ramadge, P.J. and W.M. Wonham (1982). Supervisory control of discrete event processes. In: *Feedback Control of Linear and Nonlinear Systems, Lecture Notes in Control and Information Sciences 39*. Springer-Verlag. New York. pp. 202–214.

Ramkumar, K.B., P. Philips, W.K. Ho, H.A. Preisig and K.W. Lim (1999). A real-time realization of fault-detection and diagnosis using finite-state automation. In: *Proceedings of the 14-Th IFAC World Congress*. Beijing, China.

Ramkumar, K.B., Patrick Philips, H.A. Preisig, W.K. Ho and K.W. Lim (1998). Structured fault-detection and diagnosis using finite-state automaton. In: *Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society*. Aachen, Germany. pp. 1667–1672.

Reisig, W. (1985). *Petri Nets: An Introduction*. Vol. 4 of *Monographs in Theoretical Computer Science*. Springer Verlag. New York.

Roberts, Sanford M. and Jerome S. Shipman (1972). *Two-Point Boundary Value Problems: Shooting Methods*. Vol. 31 of *Modern Analytic and Computational Methods in Science and Mathematics*. Elsevier. New York.

Robinson, D.F. and L.R. Foulds (1980). *Digraphs: Theory and Techniques*. Gordon and Breach science publishers.

Schnabel, Mark K. and Volker G. Krebs (2000). State reconstruction for a class of discrete-continuous dynamical systems based on discrete measurements. In: *Proceedings of the 4th International Conference on Automation of Mixed Processes: Hybrid Dynamical Systems* (S. Engell, S. Kowalewski and J. Zaytoon, Eds.). Dortmund, Germany. pp. 61–66.

Sontag, Eduardo D. (1990). *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Vol. 6 of *Texts in Applied Mathematics*. Springer-Verlag. New York.

Stiver, J.A. and Panos J. Antsaklis (1993). Extracting discrete event system models from hybrid control systems. In: *Proceedings of the 1993 International Symposium on Intelligent Control*. Chicago, Illinois, USA. pp. 298–301.

Stursberg, O., S. Kowalewski and S. Engell (1997). Generating timed discrete models of continuous systems. In: *Proceedings of the 2nd IMACS Symposium on Mathematical Modelling of Systems (MATHMOD)*. Vienna, Austria. pp. 203–210.

Tavernini, Lucio (1987). Differential automata and their discrete simulators. *Nonlinear Analysis, Theory, Methods and Applications* **11**(6), 665–683.

Thistle, J.G. and W.M. Wonham (1994). Supervision of infinite behavior of discrete-event systems. *SIAM Journal on Optimal Control and Optimization* **32**(4), 1098–1113.

Tittus, Michael and Bo Egardt (1998). Control design for integrator hybrid systems. *IEEE Transactions on Automatic Control* **43**(4), 491–500.

Tomlin, Claire, George J. Pappas and Sastry Shankar (1998). Conflict resolution for traffic management: A study in multiagent hybrid systems. *IEEE Transactions on Automatic Control* **43**(4), 509–521.

Vaandrager, Frits W. and van Schuppen, Jan, Eds.) (1999). *Hybrid Systems: Computation and Control*. Vol. 1569 of *Lecture Notes in Computer Science*. Springer.

Van der Schaft, A. J. and J. M. Schumacher (1996). The complementary-slackness class of hybrid systems. *Mathematics of Control, Signals and Systems* **9**, 266–301.

Van der Schaft, A.J. and J.M. Schumacher (2000). *An Introduction to Hybrid Dynamical Systems*. Vol. 251 of *Lecture Notes in Control and Information Sciences*. Springer. London.

Van Schuppen, Jan (1998). A sufficient condition for controllability of a class of hybrid systems. In: *Hybrid Systems: Computation and Control* (T.A. Henzinger and S. Sastry, Eds.). number 1386 In: *Lecture Notes in Computer Science*. Springer. Berlin. pp. 374–383.

Vinogradov, Yu. A. (1970). On finite model schemes having discrete functioning. *Systems Theory Research* **23**, 121–128.

Willems, J.C. (1991). Paradigms and puzzles in the theory of dynamic systems. *IEEE Transactions on Automatic Control* **36**(3), 258–294.

Zadeh, Lofti A. and Charles A. Desoer (1963). *Linear System Theory: The State Space Approach*. McGraw-Hill Series in System Science. McGraw-Hill.

Notation

Symbol	Description	Page
$\otimes$	Boolean 'and'	18
$\ominus$	Boolean 'difference'	18
$\oplus$	Boolean 'or'	18
$\#(\tilde{X})$	cardinality (number of elements) of a set $\tilde{X}$	30
$2^{\tilde{X}}$	collection of all subsets of $\tilde{X}$	15
$\tilde{X} \setminus \tilde{X}'$	set difference	6
$f \circ g(x)$	composite function $f(g(x))$	17
$\mathrm{Im}(A)$	image of A	43
$\ker(A)$	kernel of A	43
a^T	transposed of a	21
a^i	associated with the i-th coordinate of a vector	30
a_i	i-th element of a set	30
$A_{\tilde{u}}$, $A_{\tilde{U}}$	adjacency matrix for input $\tilde{u}$ or the input set $\tilde{U}$	21
$\mathbf{A}_{\tilde{U}}$	labeled adjacency matrix for the input set $\tilde{U}$	21
C^0	continuous functions	3.1
$\mathcal{C}^n$	continuous functions from closed intervals to $\mathbb{R}^n$	3.1
$\mathcal{C}^n_{x,int}$, $\mathcal{C}^l_{y,int}$	$\mathcal{C}^n$ functions starting in $int(H_x(\tilde{x}))$, $int(H_y(\tilde{y}))$	31
$\tilde{d}$	discrete fault	119
$D_{\tilde{u}}$	direction matrix for input $\tilde{u}$	90
$\tilde{D}$	set of discrete faults	119
$\mathcal{D}(\xi)$	domain of a function ξ	31
e_i	i-th unit vector	63
$F^{\tilde{d}}_{\tilde{u}}$	fault adjacency matrix for fault $\tilde{d}$ and input $\tilde{u}$	121
ϕ	transition function	15
h	output map	15
$H_x(\tilde{x})$, $H_u(\tilde{u})$, $H_y(\tilde{y})$	hypercube associated with $\tilde{x}$, $\tilde{u}$, or $\tilde{y}$	30
$int(H_x(\tilde{x}))$	(relative) interior of $H_x(\tilde{x})$	23
k	number of discrete inputs	32
$K_{\tilde{u}}$	matrix defining uncorrectable transitions for $\tilde{u}$	90

l	dimension of the output space	36
m	dimension of the input space	25
M	Boolean matrix used for merging discrete states	123
n	dimension of the state space	25
N	neighbor matrix	22
NC	number of computations	48
NO	number of optimizations	44
p	number of discrete states	16
PC^0	piecewise continuous functions	26
$\mathcal{PC}^m$	piecewise continuous functions from closed intervals to $\mathbb{R}^m$	26
$\mathcal{PC}^m_{u,int}$	$\mathcal{PC}^m$ function starting in the interior of $H_u(\tilde{u})$	33
Q	map from continuous to discrete states	31
r	number of discrete faults	119
R	map from continuous to discrete outputs	38
S	map from continuous to discrete inputs	33
$S_{\tilde{U}}$	matrix defining unpreventable transitions for the input set $\tilde{U}$	86
T	transformation matrix	16
u	continuous input	3
υ	input signal	26
$\upsilon_{[t_0,t_1]}$	input signal for a closed interval	26
$\tilde{U}$	set of discrete inputs	15
x	continuous state	3
$\tilde{x}, \bar{x}, \hat{x}$	discrete state (tuple, integer, Boolean vector representation)	16
ξ	state trajectory	26
$\xi_{[t_0,t_1]}$	state trajectory for a closed interval	26
$\tilde{X}, \bar{X}, \hat{X}$	set of discrete states	16
y	continous output	36
$\tilde{y}$	discrete output	36
θ	output signal	36
$\theta_{[t_0,t_1]}$	output signal for a closed interval	36
$\tilde{Y}$	set of discrete outputs	37

Index

Samenvatting

In dit proefschrift worden geregelde systemen bestudeerd, die bestaan uit een continu proces en een discrete-toestand regelaar. Voorbeelden hiervan zijn processen, die worden geobserveerd door discrete sensoren en die bestuurd worden d.m.v. discrete ingangen, en systemen waarbij de interactie met andere processen begeleid wordt door een computer programma. Voor deze systemen zijn modellen, regelstrategieën en een fout-detectie methode ontwikkeld.

De aanpak die in dit proefschrift wordt gevolgd om dergelijke hybride systemen te bestuderen is gebaseerd op het omzetten van de continue dynamica van het proces (beschreven door differentiaalvergelijkingen) naar een discrete-toestand systeem. Op deze manier kan de interactie tussen twee discrete-toestand systemen bestudeerd worden, hetgeen makelijker is dan het analyseren van het oorspronkelijke hybride systeem.

Om de continue dynamica te 'discretizeren' wordt de toestandsruimte gepartitioneerd in (meerdimensionale) rechthoeken waaraan discrete toestanden van een automaat worden toegekend. Een discrete-toestand model kan worden beschreven door de mogelijke transities tussen discrete toestanden te specificeren. Een overgang van een discrete toestand naar een andere is mogelijk wanneer er een trajectorie bestaat die aan de continue dynamica voldoet en van de overeenkomstige rechthoek naar de andere gaat. Om na te gaan of zo'n trajectorie bestaat is een algoritme ontwikkeld, dat het grensvlak tussen twee rechthoeken controleert op afgeleiden van de trajectoriën, die gericht zijn naar de naburige rechthoek en daardoor een transitie toestaan.

Voor systemen met discrete metingen, d.w.z. metingen waarbij alleen een signaal uitgegeven wordt als een bepaalde waarde is bereikt in plaats van wanneer een bepaalde tijd is verstreken, wordt getoond hoe de continue trajectorie gereconstrueerd kan worden voor zowel lineaire als niet-lineaire systemen. Wanneer de ingang bekend is, de differentiaalvergelijkingen het proces exact beschrijven, en meetruis en verstoringen afwezig zijn dient hiertoe een meerpunt randwaardeprobleem opgelost te worden. Op deze manier kan een conventionele continue regelaar worden aangewend, of de informatie kan gebruikt worden voor extra regelacties. In het geval dat een discrete-toestand regelaar gebruikt moet worden, wordt getoond hoe de discrete toestand gere-

construeerd kan worden uitgaande van discrete metingen.

De discrete-toestand modellen die resulteren van het modelleer algoritme kunnen gebruikt worden voor regelaarontwerp. Er worden drie regelaarontwerp strategieën voorgesteld, waarbij, behalve het verkregen discrete-toestand model, ook het feit wordt gebruikt dat het onderliggende systeem continu is.

De eerste regelaarontwerp methode is gebaseerd op de constructie van transities waarvan gegarandeerd kan worden dat ze daadwerkelijk plaatsvinden door de juiste ingangen te kiezen. Omdat het in het algemeen moeilijk is om te bepalen of dit mogelijk is, wordt een voldoende conditie hiervoor afgeleid. Het idee is om ongewenste transities te voorkomen (of the corrigeren) met ingangen die de continue toestand een gewenste richting laten bewegen. Met alle transities die aan de voldoende conditie voldoen wordt een graaf geconstrueerd. Geschikte paden van een gegeven begintoestand naar een gewenste eindtoestand kunnen worden gevonden d.m.v. een algoritme.

Het principe van de tweede en derde regelaarontwerp methode is gebaseerd op de constructie van (geregelde) invariante verzamelingen waarvoor het zeker is dat altijd ingangen gevonden kunnen worden die voorkomen dat de continue trajectorie de verzameling verlaat. Het idee is om een reeks van verzamelingen te berekenen zodanig dat de eerste verzameling de gewenste eindtoestand bevat, terwijl de laatste verzameling de begintoestand bevat. Deze reeks verzamelingen is zodanig geconstrueerd dat altijd een ingang gekozen kan worden die de continue toestand in de richting van een voorganger in de reeks verzamelingen beweegt, totdat uiteindelijk de gewenste toestand is bereikt. Het verschil tussen de twee regelaarontwerp methoden is de constructie van de invariante verzamelingen. Een methode verwijdert discrete toestanden die de invariantie onmogelijk maken, terwijl de andere methode die discrete toestanden toevoegt waarnaar transities niet voorkomen kunnen worden.

Discrete-toestand modellen van de continue systemen kunnen ook gebruikt worden voor het detecteren en isoleren van fouten in een proces. Hiertoe wordt voor iedere verwachte fout een apart discrete-toestand model gemaakt. Vervolgens wordt middels het vergelijken van de discrete-toestand modellen bepaald of een gemeten transitie veroorzaakt kan zijn door het nominale (foutvrije) proces of door een opgetreden fout. Door het samenvoegen van twee of meer discrete-toestanden tot een nieuwe toestand kan het totale aantal toestanden gereduceerd worden waardoor snellere berekeningen (voor *on-line* implementatie van de fout-detectie methode) mogelijk zijn. Het is echter alleen redelijk om toestanden samen te voegen wanneer er geen informatie verloren gaat. Expliciete uitdrukkingen worden gegeven die aangeven of dit het geval is.

Alle regelaarontwerp en fout-detectie methoden zijn expliciet uitgedrukt in termen van Booleaanse vectoren en matrices en kunnen, in principe, direct geïmplementeerd worden. Alle besproken concepten worden toegelicht door middel van voorbeelden, die de mogelijkheden van de methoden laten zien.

Dankwoord

In de tijd dat ik met mijn promotie bezig ben geweest zijn verschillende mensen van belang voor mij geweest. Graag wil ik deze mensen bedanken.

Als eerste wil ik mij promotor Heinz Preisig bedanken, die mij de kans en de ruimte gaf om iets van de opdracht te maken. Ook mijn tweede promotor, Paul van den Bosch ben ik dankbaar voor zijn begeleiding en adviezen. De commissieleden Jan van Schuppen en Jan Lunze ben ik erkentelijk voor hun waardevol commentaar. Grote dank en waardering ben ik Maurice Heemels verschuldigd, die als copromotor zijn best gedaan heeft om het beste uit mij te halen; een taak die hem de nodige tijd gekost heeft. Om dezelfde reden dank ik Martin Weiss, die in de eerste helft van mijn promotie van groot belang voor mij was.

Al mijn collega's bij de 'systems and control' groep van natuurkunde wil ik bedanken voor de prettige sfeer waarin ik met hen heb mogen werken. In het bijzonder wil ik mijn collega-AIO's bedanken (in volgorde van kennismaking): Georgo Angelis, Uwe Keineidam, Roel Lipsch, Annelies Balkema, Mathieu Westerweele en Gerwald Verdijck. Hartelijk dank voor al jullie adviezen, hulp, luisterende oren en, natuurlijk, voor alle lol.

Dankbaar ben ik ook de mensen bij de 'control systems' groep van elektrotechniek, die mij in de laatste fase van mijn promotie hartelijk in hun groep hebben opgenomen. Diederik, Dik, Vick, Bart, Leo, Patricia, Hardy, Andrei, en Mario: succes met jullie promotie of baan.

Tenslotte wil ik mijn vrienden, mijn ouders en mijn zus bedanken, die mij niet alleen de afgelopen periode gesteund hebben, maar altijd.

Patrick Philips
Eindhoven, Februari 2001.

Curriculum Vitae

9 mei 1971 Geboren te Roermond.

1983 - 1989 Atheneum B, scholengemeenschap Sint Ursula, Horn.

1989 - 1993 Studie wertuigbouwkunde, Hogeschool Eindhoven. Afstudeerrichting mechatronica.

1993 - 1996 Verkorte opleiding werktuigbouwkunde, Technische Universiteit Eindhoven. Met lof afgestudeerd bij de sectie Systeem en Regeltechniek op het onderwerp 'Selection of Actuators and Sensors Based on Robust Performance'. Afstudeerwerk beloond met de Unilever research award.

1996 - 2001 Promotieonderzoek, als assistent in opleiding (AIO-4) bij de 'systems and control group' van de faculteit Technische Natuurkunde, Technische Universiteit Eindhoven.