Control Synthesis using Modal Logic and Partial Bisimilarity

— A Treatise Supported by Computer Verified Proofs —

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op dinsdag 6 september 2016 om 16:00 uur

door

Allan Carolus van Hulst

geboren te Nijmegen

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. L.P.H. de Goey
1e promotor:	prof.dr. J.C.M. Baeten
2e promotor:	prof.dr. W.J. Fokkink
copromotor:	dr.ir. M.A. Reniers
leden:	prof.dr. M. Fabian (Chalmers University of Technology)
	prof.dr. S. Pinchinat (University of Rennes 1)
	prof.dr.ir. A.A. Basten

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

IPA Dissertation Series 2016-11
ISBN: 978-90-386-4141-6
A catalogue record is available from the Eindhoven University of Technology Library

Typeset with LaTeX (pdfTeX 3.14159265-2.6-1.40.16, BibTeX 0.99d)
Printed by: Gildeprint – Enschede

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The author was employed at the Eindhoven University of Technology.

Contents

Acknowledgments

I would hereby like to express my profound thanks to all members of the reading committee who gave many valuable comments and actively participated in helping to improve the contents of this thesis. The fruits of their labor even contained many re-created models and analyses. I do realize that not every PhD student is blessed with such an extensive effort by their reading committee and therefore I am grateful for this.

When my advisor Jos Baeten announced shortly after the start of my PhD that he would become the director of the CWI institute, I immediately realized that this presented an opportunity he could not let go. Compared to several years ago, I now better understand how science needs the type of person he is. Thank you Jos for choosing to stay my advisor until the very end.

My second advisor Wan Fokkink carries many admirable responsibilities in the scientific field. Besides being the head of the theoretical computer science group at the Vrije Universiteit Amsterdam, he supervises several PhD students at multiple universities and has taken on responsibilities in education beyond teaching efforts only. In my conversations with him, I experienced an openness for new ideas and a quite noticeable quickness of mind that I have not seen in many others. Thank you Wan for being as complete as you are.

Dear Michel, you deserve so much credit for helping me to come to this point and I admire you as a person for having the patience to be my daily supervisor, because I realize that I am — for lack of a better description — a difficult person to work with. Thank you Michel for continuing your efforts where others would have given up. I sincerely hope that the seven years of headaches you predicted will not materialize, although I do realize that seven seas of rye would not be a viable alternative, given your taste in music. I am convinced many people have noticed how you tried to help me and others through our daily struggles and I know this will some day reflect very positively upon you.

Chapter 1

Introduction

Control theory is a broad research topic which has been investigated in theoretical form for a considerable time. The main focus in this thesis is on discrete event supervisory control theory, as first coherently formulated by Ramadge and Wonham [76]. Although this thesis partially diverts from the approach in [76], many concepts are inherited. Supervisory control theory studies how a hardware device, manufacturing process or other kind of operational facility may be steered to operate as intended, thereby taking into account signals or measurements obtained from the device under control. In this sense, the word *control* requires disambiguation, since absolute control is often not achievable due to the fact that any realistic device or process is at least partially influenced by the input of uncontrollable sensor readings or environment conditions which may not be influenced. In this regard, the word *control* is semantically closer to how it is used in, for instance, air traffic control. An air traffic controller certainly does not control the operation of any individual aircraft in the airspace he oversees, but may guide the pilots in such a way that the operation of all aircraft around an airport functions flawlessly. Note how this example also takes into account how uncontrollable behavior may influence the controlled operation around an airfield. While the air traffic controller cannot prevent a sudden thunderstorm or the occurrence of strong crosswinds, he may guide pilots around such adverse weather effects. This is a first step towards an abstract understanding of supervisory control in a context where uncontrollable behavior appears: how to steer the controllable part of behavior such that overall desired behavior occurs. Figure 1.1 illustrates the most common application context for supervisory control: an (often larger) hardware device or manufacturing process is steered by con-

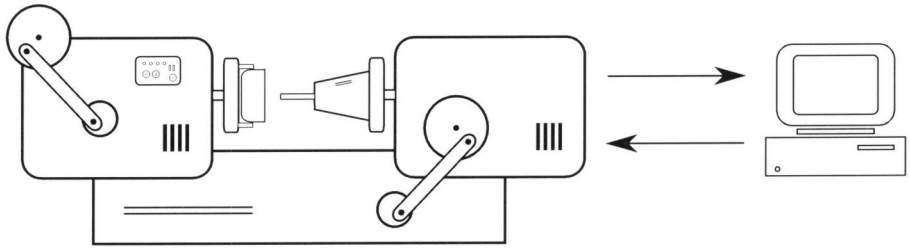

Figure 1.1: The application context for supervisory control. A large hardware device or manufacturing process is controlled by control software, which steers its operation in such a way that desired behavior occurs.

trol software to operate as desired. This thesis describes new well-founded methodologies for the automated generation of models of controlled behavior for such applications.

To further develop the treatment of supervisory control, abstractions and limitations to obtain a useful interpretation of control have to be made. In the sequel, the term *plant* is used to refer to the device or operational process under control, since this is the term most often used for this purpose in preceding works. Supervisory control is based on an abstract model of the behavior of the plant for the purpose of generalization and because in most situations the plant is far too complex to study directly. A further limitation at the core of this thesis is to limit plant models to discrete event models. A discrete event model consists of states and transitions between these states, which are labeled by events. It models the behavior of a system, or in most cases, an abstraction thereof. These models adhere to the often reasonable assumption that the system is at each point in time in a certain state and that any change between states may be described by an event taking place instantaneously. Systems which exhibit continuously evolving behavior do not fall within the realm of this description. From both a mathematical and illustrative point of view, a formalization of a discrete event model as a labeled graph is often helpful. An elaborate treatment of discrete event systems may be found in [21]. The examples in Figure 1.3 show such discrete event models. In many cases, a relatively complete system description is obtained by not only labeling transitions by events but also by adding more information to states by means of labels. Kripke-structures [20] with labeled transitions will be employed in this thesis to achieve this.

A discrete interpretation of behavior as described here is often a useful abstraction since it directly models which events should be controlled. It is also

a useful abstraction from a practical point of view since the digitization of sensors and actuators often directly leads to such a discrete interpretation. The study of supervisory control is further limited to those plant models which are assumed to be immutable. A strict distinction is made between controllable and uncontrollable events. A controllable event represents behavior which may be disabled. For instance, switching an electric motor off is a typical example of a controllable event. On the other hand, uncontrollable events represent behavior which may not be influenced. For example, the outcome of a temperature sensor is something that control cannot directly influence. In supervisory control, the controllability aspect of an event does not change. That is, an event does not suddenly change from being controllable to uncontrollable, or the other way around. In addition, an assumption is made in this thesis that all events are observable to the outside world, while control under partial observation is the subject of various other works (see, for instance, [5] and [16]).

Actual control in terms of affecting behavior still needs to be made more concrete. Control is limited to disabling (controllable) events for various reasons. First and foremost, since the plant model is assumed to be immutable, new behavior cannot be introduced. The objective that control should be achieved by means of disabling *events* is also justified by practical reasons. A state is an abstraction which represents a situation in the actual plant. For instance, the fact that a mechanical beam is oriented in a certain way may be represented by a state in a discrete event model. Effectuating control by removing *states* would therefore effectively remove or deny the existence of a practical situation inside the plant, which would contradict the earlier assumption that the plant model is immutable. On the other hand, suppose that the aforementioned mechanical beam is positioned by an electric motor, and suppose that the operation of this motor is steered by control software. It is then is far more realistic to prevent an undesired orientation by disabling the corresponding electric motor events. In other words, if an undesired state were to be prevented from being accessed, control in a discrete event context comes down to disabling events which provide access to this state.

The control loop in Figure 1.2 is a more concrete interpretation of discrete event control. First, the control loop as it is applied in traditional supervisory control theory [76], as depicted in Figure 1.2a, is considered. A distinction is made between two separate entities, the plant and the *supervisor*, which operate in conjunction with each other. The plant, being in a certain state, requires confirmation from the supervisor to execute each *controllable* event. In doing so, the supervisor ensures that the plant conforms to the desired operation of the system. Both plant and supervisor may have a different physical realiza-

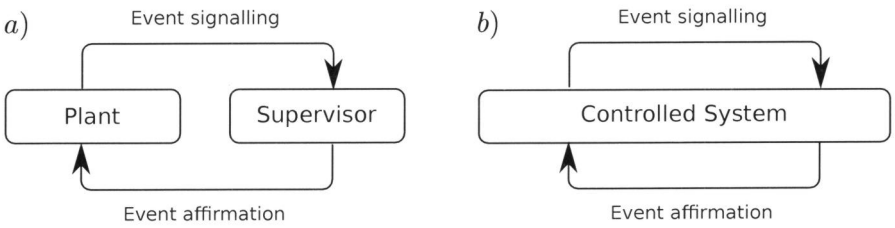

Figure 1.2: The traditional control loop for supervisory control is depicted in Figure 1.2a as a feedback model. Systems which do not allow a strictly separated supervisor are assumed to have an internal control loop, as shown in Figure 1.2b.

tion. For instance, a concrete machine may function as the plant, and an embedded software program may function as the supervisor. Furthermore, and also of importance with regard to this thesis, plant and supervisor are mainly considered in their conjunctive operation. It may therefore be required to interpret the entire system as having an internal control loop, as illustrated in Figure 1.2b. In order to realize such a controlled system, a given plant model is taken and the application of control synthesis results in a modified model which represents the behavior of the plant as if it were under control. Synthesis from a controlled systems perspective, as opposed to the construction of a strictly separate supervisor, results from the presence of non-determinism, as discussed in Section 1.2 and later on in this thesis.

The title of this thesis may now be explained by means of the preceding exhibition of control-theoretic notions. The formulation *control synthesis* in the title refers to the automatic derivation of the behavioral restriction of a plant model, based upon the specification of desired behavior in *modal logic*, which is the main topic of this thesis. The *partial bisimilarity* part refers to the approach taken in this thesis where the relationship between original plant model and controlled behavioral model is specified by means of the coinductive notion of partial bisimilarity. The subtitle, *a treatise supported by computer verified proofs*, refers to the thorough treatment which is applied to ascertain the correctness of the control synthesis methods described in this thesis. Most definitions and proofs have been formalized using the Coq proof assistant, which provides more certainty regarding the obtained results and turned out to be very helpful during the construction of the various theories for control synthesis in this thesis.

The remaining part of the setup for control is considered further in this introductory chapter. Section 1.1 provides a quick overview of the approach to control synthesis in this thesis by means of an example. Non-deterministic

models of plant behavior are allowed for the purpose of better abstraction in creating plant models. This has various implications upon the precise control framework which is applied, as explained in more detail in Section 1.2. Besides the aforementioned control objective of only limiting existing behavior, it is also required that this limitation of behavior is minimally restrictive. Section 1.3 treats this subject in more detail. In the preceding explanatory introduction to control theory, *desired behavior* was mentioned numerous times. Section 1.4 describes modal logic, which is applied as a formalism for the specification of desired behavior in this thesis. A comprehensive review of related works can be found in Section 1.5, which also compares several of these works to the main approach for controlled system synthesis in this thesis. Computer verified proofs are used to rigorously verify the correctness of the various theories involved. A short introduction into the application of such computer verified proofs is provided in Section 1.6. An overview of the remainder of this thesis, including an enumeration of the underlying scientific publications, can be found in Section 1.7.

1.1 Approach

This thesis concerns the controlled system synthesis on non-deterministic automata for specifications in modal logic, and thereby builds upon and extends earlier research in supervisory control theory [76]. This section concerns a short introduction to the specific type of controlled system synthesis in this thesis. The controlled systems perspective concerns a system under control — the *plant* — and a system component which restricts the plant behavior — the *supervisor* — which are interpreted as a single integrated entity, as shown in Figure 1.2b. This means that a given model of all possible plant behavior is taken, and a new model which is constrained according to a logical description of desired behavior — the *specification* — is constructed. This resulting model represents the controlled behavior of the plant, and is therefore referred to as the *controlled system*. The automated generation, or *synthesis*, of such a restricted behavioral model incorporates a number of standard concepts from supervisory control theory [76], which guarantees that the generated model is a proper controlled system with regard to the original plant model. This includes a strict partitioning of behaviors into controllable and uncontrollable events, such that synthesis does not disable accessible uncontrollable events, thereby achieving a property referred to as *controllability*. In addition, synthesis preserves all behavior which does not invalidate the specification, thereby inducing *maximal permissiveness*. The synthesis the-

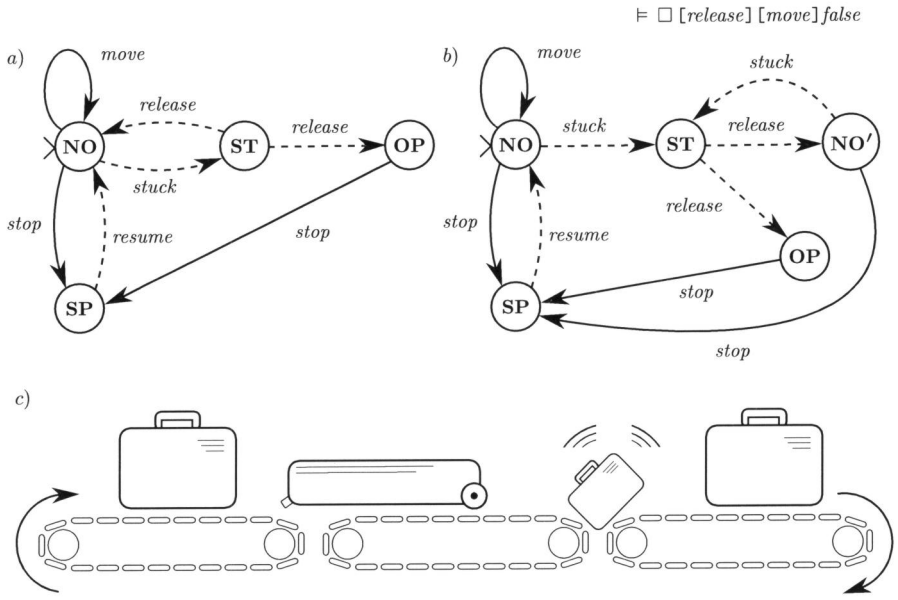

Figure 1.3: Control synthesis in a non-deterministic setting. A luggage conveyor belt, depicted in Figure 1.3c is modeled by the state-diagram in Figure 1.3a. Controlled operation such that a *release* event is not directly followed by a *move* event is shown in Figure 1.3b.

ory put forward in this thesis further allows the expression of marker state reachability and deadlock-freeness, which are often employed in supervisory control [76]. In a broad sense, this thesis describes research results into maximally permissive controlled system synthesis for non-deterministic behavioral models, thereby integrating existing notions from supervisory control theory. The controlled systems perspective is very similar to earlier approaches in supervisory control synthesis in the sense that the plant and supervisor in their combined operation are considered. However, due to non-determinism the approach in this thesis cannot be applied to obtain a strictly separated supervisory controller.

Controlled system synthesis on a non-deterministic model is further illustrated by the example in Figure 1.3. This example provides a first intuition into the type of models and specifications considered in this thesis, and sheds some light on the inherent problems that have been tackled in this research. The main purpose of this example is to transfer intuitions rather than be-

ing an actual example of a realistic system model. It consists of a system of conveyor belts for luggage handling at an airport, and is loosely based on research done at Vanderlande Industries [48, 50]. The state-diagram shown in Figure 1.3a models the uncontrolled operation of this system. If the system is in normal operation (state **NO**), it repeatedly executes a *move* event. However, as depicted in Figure 1.3c, a small suitcase might get stuck, halting the system (state **ST**). If the suitcase causing the obstruction is pulled loose by one of the travelers (event *release*), the conveyor belt resumes normal operation. Also, one of the operators may release the suitcase (state **OP**), *stop* the conveyor belt to make sure that everything is alright, and then *resume* its normal operation. Note that the occurrence of a *release* event may be caused by two different situations. First, the traveler who owns the suitcase may free it from its undesirable position, and subsequently leave the airport. Second, a different traveler, who does not own the suitcase, may pull it loose and — in good faith — put it back on the conveyor belt. Since in the second situation the suitcase still poses a threat to the desired operation of the system, the behavior of this system should be controlled in such a way that a *release* event cannot be followed immediately by a *move* event, thereby forcing the system to go through the **SP** state. This required behavior is formalized by the modal expression $\Box\,[\,release\,]\,[\,move\,]\,false$; intuitively described as: after every *release*, a *move* event should not be allowed. In Figure 1.3 dashed lines are used to indicate uncontrollable events, which may not be disallowed. Figure 1.3b models the controlled operation of this system, and thereby satisfies $\Box\,[\,release\,]\,[\,move\,]\,false$, while only behavior that invalidates this property has been disallowed. The adapted behavioral model incorporates a new state **NO'**, modeling the new behavior of the **NO** state, after a *release* event has happened. It thereby models the remaining behavior of the **NO** state, after a behavioral restriction has been applied. One of the main theoretical contributions of this thesis is a mathematically sound way to derive such new states.

1.2 Non-Determinism

In this thesis control synthesis is considered in conjunction with non-determinism. The demand for control synthesis in a non-deterministic context is clearly present in the research field, as witnessed by several research developments (see, for instance, [28] and [57]). Non-determinism allows for a higher level of abstraction in modeling plant components [22]. For instance, non-determinism may be used to model lack of observability or to intro-

duce other model-specific abstractions. Furthermore, as discussed in Section 1.5, there is at present no consistent theory regarding control synthesis for non-deterministic systems to which everyone agrees as being a definitive approach. Therefore, finding a (partial) solution to this problem is interesting in itself and will definitely contribute to the research field. The difficulties surrounding supervisory control and non-determinism are illustrated by a simple example. Suppose one creates a (very abstract) model of a printing system which is able to print in both color and black and white. This may be represented by two equal events *print* leaving a single state towards the respective sub-systems for printing in the requested color setting. If an uncontrollable sensor signal indicates that the printer is out of blue ink, a supervisor may disable the *print* event towards color printing. It is clear that strict event-based synchronization is not able to express that only one of these two *print* events should be disabled. As illustrated in Figure 1.2a, the supervisor cannot make a distinction between disabling one of the two *print* events, within this interpretation of control. Therefore, the research in this thesis is restricted to the controlled system perspective, where plant and supervisor are considered as a single integrated entity, and a new system is synthesized which represents the plant as if it were under control, as illustrated in Figure 1.2b. This forms a natural generalization in the sense that for deterministic models, the controlled system coincides with the supervisor [21]. However, for non-deterministic models it is in general not possible to derive a strictly separated supervisor.

Despite the interest in control synthesis among other researchers, one might wonder what the essential value is of such synthesis for non-deterministic models, for various reasons. Every non-deterministic model may be converted into a deterministic one, and may be subsequently subjected to deterministic synthesis methodologies. However, the conversion into a deterministic model does not preserve structural integrity and may therefore lead to essential properties of the model being lost [38]. For instance, the fact that a certain state has two outgoing transitions labeled by the same event might indicate that an abstraction was created by the modeler for two different underlying situations. If these two equal events are unified when the model is converted from non-deterministic to deterministic, part of the semantics of the model would have been lost. Preserving the semantics of a model in this way is particularly important if any post-synthesis step is to be applied. Control synthesis for non-deterministic models may also be rejected due to the idea that such models rely on some form of inherent independence in making a choice between identical events, whereas control would impose determinism in this regard. However, the main objective of control synthesis

is not to remove existing independence of the system but to achieve proper control flow. Nevertheless, the research results described in this thesis should mainly be interpreted as a proposed technique for control synthesis for non-deterministic systems, and not as an attempt to build a very strong case for the general usage of non-determinism in control synthesis.

1.3 Maximal Permissiveness

In the synthesis approach considered in this thesis it is required that all synthesis solutions are *maximally permissive*. That is, the resulting controlled system contains all original behavior which does not invalidate the specification. Maximal permissiveness, in this thesis often shortened to *maximality*, is key in achieving proper controlled behavior. A behavioral adaptation which removes too much behavior is simply not an adequate solution, since it may disable plant components which are unrelated to, or not subject to control. If the navigation system of a car is to be controlled in such a way that it avoids traffic jams, then disabling all of the cars functionality except for driving in and out of the garage would be, in a very strict sense, a solution which achieves this control objective. However, it is clear that as much behavior of the car as possible should be preserved, in order to achieve usable functionality under control. As a generalization, this comes down to generating the controlled system which retains the most possible original plant behavior. In the case that additional synthesis steps or further analysis is to be applied after control synthesis takes place, it is also of the utmost importance that as much original behavior as possible is preserved. Maximal permissiveness is a standard notion in supervisory control synthesis, and can be traced back to the first foundations [76].

Maximality is often expressed in terms of language inclusion. That is, every sequence of events in the behavior of the plant also occurs in the controlled system, provided that it is not to be disallowed. Since non-deterministic plant models are considered here, a different approach is required. *Partial bisimilarity* [77] defines the relationship between plant and controlled system via a coinductive preorder. It is required that the synthesis result is the greatest satisfying witness with regard to this preorder. That is, every satisfying partial bisimulant of the plant is also a partial bisimulant of the controlled system. This intuitively captures the notion that the synthesized controlled system is actually the *best* candidate in terms of both preserving behavior and effectuating control. More technical justification as to why partial bisimulation is employed for both these purposes follows Definition 2.6. Maximal

permissiveness influences many synthesis constructions, mostly due to side-effects induced by non-determinism and solution uniqueness. This leads to restrictions in the expressiveness of specifications, synthesis adaptations and duplication of behavior, but it also leads to proof difficulties, as will be shown in the next two chapters.

1.4 Modal Logic

In this thesis modal logic is applied to express which control objectives the controlled system should satisfy. In particular, the main synthesis methodology proposed in this thesis includes integral expressiveness for marker state reachability. This results in the ability to express non-blockingness [76] via the specification logic. Modal logic can be considered a standard formalism in the specification of properties which are tested in verification tasks, such as model checking. Earlier attempts have been made to define control synthesis in various variants for modal logics. Many of these works will be considered in Section 1.5 on related work. If modal logic is applied in a control synthesis setting, then most often the μ-calculus [52], or a subset thereof, is used. The logics applied in this thesis are strict subsets of a μ-calculus variant which conforms to the following grammar defined in terms of a set of events $\mathcal{E}$ and a set of basic properties $\mathcal{P}$:

$$\mathcal{F} ::= true \mid false \mid \mathcal{P} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid [\mathcal{E}]\mathcal{F} \mid <\mathcal{E}>\mathcal{F} \mid \mu X.\mathcal{F} \mid \nu X.\mathcal{F}$$

In this definition of $\mathcal{F}$, the test for a basic state-based property $\mathcal{P}$ forms an extension of the standard definition of μ-calculus [52]. The logic $\mathcal{F}$ further includes the universal $[e]$ and existential $<e>$ look ahead from Hennessy-Milner logic [34] and a minimal μ and maximal ν fixpoint operator. Since the μ-calculus is too expressive to allow unique maximally permissive controlled system synthesis on non-deterministic models, strict subsets of $\mathcal{F}$ need to be taken into consideration. The most significant control synthesis contribution in this thesis relies on the modal logic defined below, in terms of a subset $\mathcal{C} \subseteq \mathcal{E}$ of controllable events:

$$\mathcal{F} ::= \mathcal{B} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{B} \vee \mathcal{F} \mid [\mathcal{E}]\mathcal{F} \mid <\mathcal{C}>\mathcal{F} \mid \Box\mathcal{F} \mid \Diamond\mathcal{B} \mid <\mathcal{E}> \mid dlf$$

The logic $\mathcal{F}$ now includes a generalized set of basic formulas $\mathcal{B}$, a restricted disjunction operator, an existential look ahead which is limited to controllable events and the invariant $\Box$ and reachability $\Diamond$ modal operators from Gödel-Löb logic [78]. The justification for many of these restrictions can be found

in Chapter 2. In Chapter 4 the problem of generating all maximally permissive synthesis solutions for Hennessy-Milner logic is considered, instead of finding just a single unique solution. This variant of Hennessy-Milner logic conforms to the following grammar:

$$\mathcal{F} ::= true \mid false \mid \mathcal{P} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid [\mathcal{E}]\mathcal{F} \mid <\mathcal{E}>\mathcal{F}$$

In addition to these two logics, other logics have been studied and applied for control synthesis in the research which underlies this thesis. The inductive way in which these modal logics are defined allows for the application of induction principles in the correctness proofs. Furthermore, modal logic is directly related to coinductive relationships between behavioral models. It is well-known that the μ-calculus characterizes bisimulation [31]. From a practical point of view, modal logic allows for a more intuitive and model-independent way to express specifications, compared to the description of required behavior as a subset of existing behavior.

1.5 Related Work

Before more detailed research is considered, two important works are referred to as important general introductions into discrete event control [21, 54]. The foundations of supervisory control theory can be found in the original paper by Ramadge and Wonham [76]. These results were further extended in succeeding work [88]. Important basics were established in [76], many of which are inherited in this thesis. Notably: controllability, maximal permissiveness and marker-state reachability. The setup in [76] defines the necessary preconditions such that a supervisory controller can be automatically derived, given a deterministic model of the plant. Section 2.1 details the most important constructs of Ramadge-Wonham supervisory controller synthesis at a formal level, but also highlights the key differences between the setup in this thesis and the approach in [76]. The Ramadge-Wonham framework has been extensively studied from an applicability perspective, and various developments regarding efficient implementation have taken place. For instance, a BDD-based implementation is studied in [36], and an approach based on dynamic programming is covered in [86]. The CIF toolset [17] also provides an extensive coverage of supervisor synthesis tools and related conversion mechanisms. As a general conclusion based on the studies into the implementation of Ramadge-Wonham supervisory control synthesis, one can say that partial observability makes this problem harder, if not intractable

from a computational point of view [59, 71]. Detailed considerations regarding the implementation of Ramadge-Wonham control synthesis can be found in [25].

Despite the fact that Ramadge-Wonham supervisory control synthesis is already a well-established foundation for discrete event control, many researchers proposed improvements, to which this thesis should also be considered a contribution. Besides many peripheral research directions, these improvements can be classified as follows: 1) using different formalisms at the very heart of supervisory control, compared to the standard automata-based setup in [76], 2) allowing more expressiveness in plant models, notably extensions towards non-deterministic systems and timed automata, and 3) more expressive descriptions of desired behavior, such as various temporal or modal logics. Examples of the first modification include definitions of supervisory control synthesis in terms of Petri-nets [37, 79], Büchi-automata [75, 85], process-algebraic perspectives [35, 63], and specialized predicate-based models of behavior [68]. However, most instances of research into modified frameworks for discrete event control apply a formalism which is close to, or is in fact, automata theoretic [54]. In fact, much work has been done to align the techniques in [76] to related prevalent techniques in computer science [53]; an important example being the use of parallel composition operators to express the synchronization between plant and supervisor [55]. The second and third way of improvement, respectively summarized as allowance of non-determinism and specifications of desired behavior in modal logic, are considered extensively below, since these works are very relevant for the material in this thesis.

A significant amount of research is devoted to the adaptation of standard (deterministic) supervisory control synthesis [76] to a non-deterministic setting, while the control objective of a subset of marked traces in the plant stays the same. The work in [27] forms an important contribution in the step towards handling non-determinism in the sense that many techniques from [76] are projected onto a non-deterministic setting. This also applies to maximally permissive solutions, as shown in [28]. Several solutions have been proposed to handle non-determinism in such a control synthesis environment. For instance, the approach in [57] and [80] uses prioritized synchronizations in combination with a trajectory model. The work in [57] and [80] is a further development of the work on failure traces [35, 69]. Another attempt is by handling non-determinism via forced events [32, 29], or by only allowing non-determinism in the supervisor [56]. The latter mentioned works can be considered to be relatively distanced from the approach in this thesis.

A vast amount of work is devoted to the relationship between reactive

synthesis and supervisory control. Reactive synthesis originated from the desire to automatically adapt a system model such that it conforms to a logical specification [73, 1] without particular reference to discrete event control. Reactive synthesis was then modified to include supervisory control synthesis, which would allow the creation of a supervisor based on a logical description of required behavior; an objective also sought in this thesis. Section 1.4 provides more details regarding this subject. The relationship between control synthesis, reactive systems and various modal logics has been investigated in [61]. Recent work re-evaluates the methodology to close the gap between supervisory control and reactive synthesis in a deterministic context [26]. An example of a more specific control synthesis solution using reactive synthesis can be found in [5], where the supervisory control problem is converted into a μ-calculus satisfiability problem using an automata-quotient strategy. While [5] is restricted to deterministic plant models, this work is extended in [6] to non-deterministic plants, but the treatment is non-maximal.

A very similar approach is applied in [16, 15] where, again, a quotienting scheme guarantees a non-maximal solution of a control problem for a μ-calculus specification. The key contribution in [16, 15] is in terms of a semantic tableau method to resolve the relevant μ-calculus satisfiability problem. This work is closely related to other research into reactive synthesis [58]. However, in this instance branching time temporal logic (BTTL) is applied. Again, non-maximal solutions are found by solving a realizability problem. That is, a supervisor can be derived from an automaton which is generated from the given BTTL formula. A number of approaches for reactive synthesis are studied in [87], and subsequently generalized for Markov decision processes, non-deterministic plant models and specifications in linear temporal logic. The optimization applied in [87] relates to satisfiability testing for Rabin automata. Solutions are still not maximally permissive, but can be applied in a robotics setting [87].

The research into reactive synthesis for control-theoretic purposes is often approached from a game-theoretic point of view. This forms the core of a projection of Ramadge-Wonham supervisory control [76] onto real-time systems [62] and timed automata [72, 7]. A deviation from game-theoretic approaches for control synthesis and logical specifications of desired behavior can be found in [49]. In this work, a small model theorem is derived for CTL*, which allows a proof of the existence of a supervisor by means of an emptiness test of a Rabin automaton. However, the treatment in [49] is limited to deterministic plant models. A semantic tableau based methodology can be found in [23, 24] in terms of propositional linear temporal logic. A more general treatment of the analysis and synthesis for supervisors in a temporal

logic framework can be found in [60].

Fluent linear temporal logic is another logical formalism applied for supervisory control synthesis [45, 46]; a distinction is made between system goals and environment assumptions, and between controlled and monitored actions. Controller synthesis in [45, 46] relies upon a formalism referred to as the world/machine model [47].

Several researchers worked on generalized interpretations in terms of modal or temporal logics for supervisory control synthesis. An adaptation of μ-calculus, known as quantified μ-calculus, quantifies atomic propositions and is therefore able to express maximal permissiveness inside the logic [70]. Expressions for control objectives in temporal logic may be automatically derived from other means of describing desired behaviors [83, 84, 85].

Several approaches do not result in a supervisory controller, but instead generate a policy which determines how control should be enforced. An example can be found in [74], which uses a constraint-based approach for non-deterministic and partially observed domains, with another example in [19], which does not only apply to the synthesis of supervisors, but can also be applied to automatically derive planning strategies. Such planning-related methodologies have been connected to discrete event control via AI-based techniques [65].

Partial bisimilarity as a means to express controllability, as used extensively in this thesis, was introduced in [77]. It has been applied in related form in earlier research. The work in [81] uses an abstraction of non-deterministic automata to effectively attack the state explosion problem in supervisor synthesis. Only a single rule from the preorder in [81] differs from partial bisimilarity in Definition 2.6, and this difference is discussed in Chapter 2. An approximate simulation approach is used in [82] for real-time systems, combined with MTL logic. The coinductive nature of partial bisimilarity also relates to earlier research into control of discrete event systems and coalgebra [51]. Bisimulation equivalence in relation to control synthesis in a non-deterministic context appears in [89]. The work in [18] presents a unification of various control synthesis approaches. This generalization is also defined in terms of an inclusion-type preorder.

A similar approach from an algorithmic point of view, compared to the type of control synthesis in this thesis, appears in [14]. Indeed, a similar two-phase method is applied: 1) a forward satisfiability check, 2) a backtracking control enforcement procedure. The work in [14] is able to express safety and liveness properties, but is not maximally permissive. The restriction upon disjunctive formulas as applied in this thesis was observed in earlier research [4, 3].

1.6 Computer Verified Proofs

The majority of definitions and proofs in this thesis have been formalized and verified using the Coq proof assistant [13]. Such a proof assistant (or theorem prover) provides a number of features which are helpful in the development and verification of new theories. First and foremost, a computer verified proof ascertains that a given proof is absolutely correct, provided that the chosen formalization of the theory is right and under the assumption that the theory does not contain any intrinsic inconsistencies. Furthermore, it cannot be left unmentioned that it is — of course — also required that the proof assistant in itself is correct. Such an approach may be an important aid in writing an article which has a lot of mathematical content, not in the least because the author (and reviewers) are freed from having to worry about the correctness of the proofs. Such a very precise analysis of a theory requires more in-depth consideration of the theories at hand, and thereby may aid as a qualitative improvement. Usually, many errors are discovered when a formalized proof is being constructed, and this certainly applies to the work done in this thesis. Another important feature relates to the bookkeeping functionality a proof assistant provides. In particular, when an interactive proof assistant such as Coq is used, many proof elements are automatically stored or derived [13]. This applies to, for instance, the current state of the proof, the induction hypotheses and partial proof automation. If used wisely, this results in a focal shift towards exactly the necessary proof obligations, while the proof assistant takes care of other bookkeeping tasks. All this wizardry comes at a price: significantly more time may have to be spent figuring out all details of the theory and how to encode them in the proof assistant. Also, despite the fact that a proof assistant may help in shifting the focus to exactly the right proof obligations, these may still be too detailed. This may result in wrong results since resolving details may obfuscate the bigger picture and achieving overall correctness in encoding the right theories. Overall, the net result of studying control theory by means of the proof assistant is positive, as far as the material in this thesis is concerned. In particular, due to the fact that using the proof assistant revealed so many tiny errors during the research. In particular, details regarding unfolding, solution uniqueness, maximality as well as issues concerning general provability of the correctness of the synthesis method. An attempt has been made to make this material accessible to an audience which may lack the necessary background in formalized proofs, by providing details of the applied Coq-constructs and by considering a somewhat abstracted version of the proof. This results in a formalized proof that is quite close to the mathematical expressions from a syntactic as well as a semantic point of

view. Further details are provided in the latter parts of Chapter 3 and Chapter 4. Coq-proofs for the main synthesis methodology are available at the following location:

`https://github.com/ahulst/deds/`

1.7 Overview

This section provides an overview of the remainder of this thesis and the scientific publications upon which these results are based. Chapter 2 and Chapter 3 define a specific synthesis technique which addresses the main research question in this work:

> *How to define controlled system synthesis for a reasonably expressive modal logic and non-deterministic plant models such that synthesis results are unique and maximally permissive?*

A first attempt to define this specific type of controlled system synthesis using a modal logic beyond the expressiveness of Hennessy-Milner logic appeared in:

[42] A. van Hulst, M. Reniers, and W. Fokkink. Maximal Synthesis for Hennessy-Milner logic with the Box-Modality. In *Workshop on Discrete Event Systems* (WODES), pages 278–285, 2014.

The methodology for controlled system synthesis was then further refined and the modal logic was made more expressive. This resulted in two publications:

[43] A. van Hulst, M. Reniers, and W. Fokkink. Maximally Permissive Controlled System Synthesis for Modal Logic. In *International Conference on Current Trends in Theory and Practice of Computer Science* (SOFSEM), pages 230–241, 2015.

[44] A. van Hulst, M. Reniers, and W. Fokkink. Maximally Permissive Controlled System Synthesis for Non-Determinism and Modal Logic. *Discrete Event Dynamic Systems*, Submitted.

The work in [42] is succeeded by the research in [43] and [44]. Therefore, only the contents of the latter two works is considered in this thesis. Chapter 2 is mainly devoted to examples and formal definitions in order to build up the

main synthesis method. The main focus for Chapter 3 is on verification of the earlier defined control synthesis theory. Detailed proofs for the validity of the proposed theories are included in this chapter, combined with proofs created using the Coq proof assistant. Chapter 3 is concluded by two small case studies, in order to illustrate the applicability of the synthesis method.

Multiple synthesis solutions for unrestricted Hennessy-Milner logic are considered in Chapter 4, which is based upon a different synthesis construction compared to the main synthesis method. It addresses the following research question:

> *How to construct all maximally permissive control synthesis solutions for non-deterministic plants and control specifications in Hennessy-Milner logic?*

The two articles listed below propose a solution to this problem. The work in [41] is both a more in-depth analysis of the solution in [40] as well as an extension thereof. Therefore, Chapter 4 mainly considers the work in [41].

[40] A. van Hulst, M. Reniers, and W. Fokkink. Maximal Synthesis for Hennessy-Milner Logic, *Application of Concurrency to System Design* (ACSD), pages 1–10, 2013.

[41] A. van Hulst, M. Reniers, and W. Fokkink. Maximal Synthesis for Hennessy-Milner Logic, *ACM Transactions on Embedded Computing Systems* (TECS), pages 10:1–10:21, 2014.

In Chapter 5, a process theory is developed to express supervisory control theory [76] for non-deterministic plant models. It is based upon two publications and provides a solution to the following research question:

> *How to define supervisory control synthesis, including a treatment of non-determinism, in a process theoretic framework?*

The work in [9] is specified in more detail in [10]. A case study from Chapter 3 is again analyzed in the process theoretic framework in Chapter 5, which allows a comparison between the two techniques.

[9] J. Baeten, B. van Beek, A. van Hulst, and J. Markovski. A Process Algebra for Supervisory Coordination, *International Workshop on Process Algebra and Coordination* (PACO), pages 36–55, 2011.

[10] J. Baeten, B. van Beek, A. van Hulst, and J. Markovski. A Process Algebra for Supervisory Control, *SE Report 12-01*, Eindhoven University of Technology, 2012.

Chapter 2

Synthesis and Discrete Event Control

The overall purpose of this chapter is to formally define the main control synthesis methodology of this thesis. These formalisms are henceforth scrutinized and shown to lead to a valid synthesis framework in Chapter 3. The formal basics of Ramadge-Wonham supervisory control theory are introduced here, followed by a short overview of the key differences between [76] and the setup in this chapter. One of the key statements in this chapter is shown in Definition 2.10, which formally states the precise control synthesis problem solved in this thesis, in terms of the formal definitions provided in this chapter. A number of examples are given to aid in both the abstract and concrete understanding of the way reductions of modal formulas are assigned to states; a key feature of the synthesis method in this thesis. Control is enforced by means of transition removal based on a partial satisfiability test. This specific technique is illustrated by examples and afterwards specified in formal detail. Towards the end of this chapter, the formal definitions converge to a precise formulation of the entire synthesis construction. As a shortly phrased reading guide to this chapter, it should be mainly interpreted as the necessary precursor to the formal analysis and correctness proofs in Chapter 3. The connection to Ramadge-Wonham supervisory control is also made in the next chapter, as well as an applicability analysis in the form of case studies.

The succeeding sections in this chapter are organized as follows. Section 2.1 details the basics of Ramadge-Wonham supervisory control theory at a formal level. Basic definitions in Section 2.2 treat elementary formal building

blocks which allow a precise statement of the synthesis problem in Definition 2.10. The focus then shifts to an initial expansion step in Section 2.3, where the transition relation from the plant model is projected onto a new transition relation over the state-formula product space. This is done by means of a formal reduction relation on modal expressions, which is the subject of Section 2.4. Removing transitions in order to obtain a model which satisfies the control objective depends upon a partial satisfiability test for modal expressions, which is treated in more detail in Section 2.5, and subsequently formalized in Section 2.6. The entire formal definition of the synthesis construction is then considered in Section 2.7.

2.1 Supervisory Control Theory

Supervisory control theory [76] revolves around the inhibition of controllable behavior, while at the same time leaving accessible uncontrollable behavior unaffected. In this section some of the key features of traditional supervisory control theory are formalized using parallel composition, instead of the functional characterization of synchronization as applied in [76], since this treatment is more intuitive.

Ramadge-Wonham supervisory control theory [76] defines a broadly embraced methodology for supervisory control synthesis on deterministic plant models. It identifies a number of key characteristics in the relationship between plant and controlled system, such as controllability, marker state reachability, deadlock-freeness and maximal permissiveness, which are inherited by the synthesis theory in this paper. The limitation to deterministic plant models in [76] allows the derivation of a strictly separated unique and maximally permissive supervisor, but does not embrace the increased abstraction and flexibility offered by a non-deterministic plant model.

A behavioral description of the system under control — the plant — is assumed to be given. A separate entity — the supervisor — operates in conjunction with the plant and regulates its behavior, thereby complying to the illustration in Figure 1.2a. The automated generation of such a supervisor is known as supervisory control(ler) synthesis. This traditional setup for supervisory control differs from the approach in this thesis in the sense that it assumes a deterministic plant model, which allows the derivation of a strictly separated unique supervisor[1].

[1] In [76] the term *non-deterministic* is used to refer to multiple events leaving a single state. However in [76] it is specifically stated that all events leaving a single state are assumed to be distinct.

A straightforward formalization of these notions starts with the assumption of an event-set $\mathcal{E}$ where $\mathcal{E}$ is partitioned as $\mathcal{E} = \mathcal{U} \cup \mathcal{C}$. The $\mathcal{U}$-set of uncontrollable events may contain, for instance, sensor readings which occur in the plant model, while the $\mathcal{C}$-set of controllable events may consist of actuator signals. Plant and supervisor are then modeled by means of a labeled transition system, as given in Definition 2.1.

Definition 2.1. For state-space X, transition relation $\longrightarrow \subseteq X \times \mathcal{E} \times X$, initial state $x \in X$ and set $X_m \subseteq X$ of *marked* states, a labeled transition system (LTS) is defined as a four-tuple $(X, \longrightarrow, x, X_m)$. The set of all labeled transition systems is denoted by $\mathcal{G}$.

As usual, the notation $x \xrightarrow{e} x'$ is employed to indicate that $(x, e, x') \in \longrightarrow$. The transition relation $\longrightarrow$ is then naturally extended to its reflexive-transitive closure $\longrightarrow^* \subseteq X \times X$, which is combined with the notation $x \xrightarrow{s}{}^* x'$, for $s \in \mathcal{E}^*$. That is, it holds that $(x, x) \in \longrightarrow^*$ and if $e \in \mathcal{E}$, $y \in X$ and $s \in \mathcal{E}^*$ such that $x \xrightarrow{e} y$ and $y \xrightarrow{s}{}^* x'$ then $x \xrightarrow{es}{}^* x'$. The set $X_m \subseteq X$ of *marked* states in Definition 2.1 is used to model completed or finished tasks in the plant[2]. A number of standard definitions from language theory are reiterated here:

Definition 2.2. For $g = (X, \longrightarrow, x, X_m) \in \mathcal{G}$ the language $\mathcal{L}(g) \subseteq \mathcal{E}^*$ and the marked language $\mathcal{L}_m(g) \subseteq \mathcal{E}^*$ of g are defined in the following way:

$$\begin{aligned} \mathcal{L}(g) &= \{s \in \mathcal{E}^* \mid \exists x' \in X : x \xrightarrow{s}{}^* x'\} \\ \mathcal{L}_m(g) &= \{s \in \mathcal{E}^* \mid \exists x' \in X_m : x \xrightarrow{s}{}^* x'\} \end{aligned}$$

In addition, the language closure $\overline{L} \subseteq \mathcal{E}^*$ of a language $L \subseteq \mathcal{E}^*$ is defined as follows:

$$\overline{L} = \{s \in \mathcal{E}^* \mid \exists t \in \mathcal{E}^* : st \in L\}$$

Subsequently, the parallel composition operator $\|$ is defined in order to express the interaction between plant and supervisor in Definition 2.3. The adaptation of [76] in terms of this type of parallel composition first appeared in [55].

Definition 2.3. For $g = (X, \longrightarrow, x, X_m), g' = (X', \longrightarrow', x', X'_m) \in \mathcal{G}$ the parallel composition $g \parallel g'$ is defined in the following way:

$$g \parallel g' = (X \times X', \longrightarrow_\cap, (x, x'), X_m \times X'_m)$$

[2]Note the difference between marked states in supervisory control theory and final states in regular automata theory. While the first notion is used to indicate a completed task, the latter mainly serves the purpose of language acceptance.

where $(y, z) \xrightarrow{e}_{\cap} (y', z')$ if and only if $y \xrightarrow{e} y'$ and $z \xrightarrow{e} z'$. Clearly it holds that $\mathcal{L}(g \parallel g') \subseteq \mathcal{L}(g)$ and $\mathcal{L}(g \parallel g') \subseteq \mathcal{L}(g')$ (and similar for marked languages).

Marked states have the semantic purpose of indicating a completed task which is also present in the controlled behavior, as expressed using the parallel composition operator. Therefore, presence of a marked state in one of the components is inherited in the parallel construction, as can be seen in Definition 2.3.

In the sequel, it is assumed that every $g \in \mathcal{G}$ is limited such that each state is accessible by $\longrightarrow^*$ from the initial state. Furthermore, $g \in \mathcal{G}$ is defined to be *co-accessible* if each string in $\mathcal{L}(g)$ can be completed towards a marked state, that is: $\mathcal{L}(g) = \overline{\mathcal{L}_m(g)}$.

The formalization of control starts with the notion of *controllability* in Definition 2.4.

Definition 2.4. A language $K \subseteq \mathcal{E}^*$ is said to be *controllable* with regard to $L \subseteq \mathcal{E}^*$ if for each $s \in \overline{K}$ and $u \in \mathcal{U}$ such that $su \in \overline{L}$ it holds that $su \in \overline{K}$.

Intuitively, controllability indicates potential adaptability such that accessible uncontrollable behavior is preserved. In terms of the previous definitions, one of the main results in supervisory control theory may now be compactly stated:

Theorem 2.1. For plant model $p \in \mathcal{G}$ and language $K \subseteq \mathcal{L}_m(p)$ such that $K \neq \emptyset$ and K is controllable with regard to $\mathcal{L}(p)$ there exists a supervisor $s \in \mathcal{G}$ such that $\mathcal{L}(p \parallel s) \cap \mathcal{L}_m(p) = \mathcal{L}_m(p) \cap \overline{K}$ and the following two properties hold:

1. $\mathcal{L}(p \parallel s)$ is controllable with regard to $\mathcal{L}(p)$; and
2. $p \parallel s$ is *non-blocking*, that is: $\overline{\mathcal{L}(p \parallel s) \cap \mathcal{L}_m(p)} = \mathcal{L}(p \parallel s)$

For an essentially equivalent parallel construction between plant and supervisor, a proof for Theorem 2.1 can be found in [76]. In general, a different framework compared to [76] is applied in this thesis, which will be detailed in the remaining sections of this chapter. This adaptation includes a coinductive treatment of controllability, an integrated model of plant and supervisor in their conjunctive operation (the controlled system) which does not result in a strictly separated controller, a different attitude towards desired behavior by means of specifications in modal logic, and a fluent handling of marker states by means of reachability predicates in this logic.

2.2 Basic Definitions

Part of the standard setup of supervisory control theory, as introduced in Section 2.1, is inherited in the sense that the plant description is also assumed to be immutable. A global-event set is assumed which is partitioned into uncontrollable and controllable events. In this sense, there is no difference between the framework in this thesis and the setup in [76]. The standard notion that the controllability-aspect of an event does not change, is adhered to. That is, events cannot change between being controllable and uncontrollable during the operation of the plant.

A set $\mathcal{E}$ of events and a set $\mathcal{P}$ of state-assignable basic properties is assumed. The event-set $\mathcal{E}$ is partitioned into controllable events $\mathcal{C}$ and uncontrollable events $\mathcal{U}$, such that $\mathcal{C} \cup \mathcal{U} = \mathcal{E}$ and $\mathcal{C} \cap \mathcal{U} = \emptyset$. State-based properties are used to capture state-based information, and are assigned to states using a labeling function. Events are used to capture system dynamics, and represent actions occurring when the system switches between states. Controllable events may be used to model actuator actions in the plant, while an uncontrollable event may represent, for instance, a sensor reading or a user input.

Basic properties and events are used to model plant behavior in the form of a Kripke-structure [20] with labeled transitions, to be abbreviated as *Kripke-LTS*, as formalized in Definition 2.5. Such a model forms a useful abstraction in the sense that it allows a definition of synthesis of overseeable complexity which may be formally verified. On the other hand, it forms a useful and concrete interpretation in the sense that reasonably detailed plant models may be constructed. Definition 2.5 allows the expression of state-based information via state labels, and system dynamics via a transition relation. Kripke-models were used earlier in the context of supervisory control theory in [30]. It is essential for the well-definedness of the synthesis construction in this thesis that the transition relation in this Kripke-LTS is finite. This does not exclude loops or other kinds of infinite behavior; only finiteness of the transition relation is assumed for as far as its definition as a set of triples is concerned.

Definition 2.5. For state-space X, labeling function $L : X \mapsto 2^{\mathcal{P}}$, finite transition relation $\longrightarrow \subseteq X \times \mathcal{E} \times X$ and initial state $x \in X$, a Kripke-LTS is defined as a four-tuple $(X, L, \longrightarrow, x)$. The set of all Kripke-LTSes is denoted by $\mathcal{K}$.

As usual, the notation $x \xrightarrow{e} x'$ is used to denote that $(x, e, x') \in \longrightarrow$. The reflexive-transitive closure $\xrightarrow{s}{}^*$, for $s \in \mathcal{E}^*$, over transition relation $\longrightarrow$, is defined in the following way: For all $x \in X$ it holds that $(x, x) \in \xrightarrow{1}{}^*$, where 1 denotes the empty string; and if there exist $e \in \mathcal{E}$, $s \in \mathcal{E}^*$ and $y, x' \in X$ such

that $x \xrightarrow{e} y$ and $y \xrightarrow{s}{}^* x'$, then $x \xrightarrow{es}{}^* x'$. In most cases an abstraction of this reflexive-transitive closure is used, without reference to a particular $s \in \mathcal{E}^*$. That is, $x \longrightarrow^* x'$ if and only if there exists an $s \in \mathcal{E}^*$ such that $x \xrightarrow{s}{}^* x'$.

As discussed in Section 1.2 and as formally stated in Definition 2.5, a nondeterministic plant model is assumed. In general, this does not lead to a strictly separated supervisor which operates in conjunction with the plant under synchronization. Section 1.2 details a number of reasons why it is inherently problematic to strive towards strictly separated control in a nondeterministic context. The research effort should therefore focus on the synthesis of *controlled systems*. A $\mathcal{K}$-model of plant behavior is taken and subsequently a new $\mathcal{K}$-model is derived which behaves as the plant under control. Section 1.2 details why such a restriction is not too far-fetched.

A formal connection needs to be set up between the plant and the controlled system, in order to define that the latter is a proper behavioral restriction of the former. For this purpose, partial bisimilarity is employed. Partial bisimilarity [77] is an adaptation of bisimilarity such that controllable events are simulated, while uncontrollable events are bisimulated. For plant model $k \in \mathcal{K}$ and synthesis result $s \in \mathcal{K}$ it is required that s is related to k via partial bisimilarity. This signifies the fact that synthesis did not disallow any accessible uncontrollable event, which implies controllability in the context of supervisory control. Research in [64] details the nature of this partial bisimilarity relation. If all events are controllable, then partial bisimilarity coincides with strong similarity. On the other hand, if all events are uncontrollable, partial bisimilarity coincides with strong bisimilarity [31]. It is formalized in Definition 2.6.

Definition 2.6. A pair of Kripke-LTSes $k' = (X', L', \longrightarrow', x') \in \mathcal{K}$ and $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ are related via partial bisimulation (notation $k' \preceq k$) if there exists a relation $R \subseteq X' \times X$ such that $(x', x) \in R$ and for all $(y', y) \in R$ the following holds:

1. $L'(y') = L(y)$; and
2. if $y' \xrightarrow{e}{}' z'$, for $e \in \mathcal{E}$ and $z' \in X'$, then there exists a $z \in X$ such that $y \xrightarrow{e} z$ and $(z', z) \in R$; and
3. if $y \xrightarrow{e} z$, for $e \in \mathcal{U}$ and $z \in X$, then there exists a $z' \in X'$ such that $y' \xrightarrow{e}{}' z'$ and $(z', z) \in R$.

If the relation $R \subseteq X' \times X$ is of particular importance then the notation $k' \preceq_R k$ is used to indicate that k' and k are related via partial bisimulation as witnessed by R.

Both partial bisimulation as formalized in Definition 2.6 [77] as well as a variant which omits the requirement $(z', z) \in R$ in the 3rd clause in Definition 2.6 [81] have been described in the literature. In this thesis, an explicit choice is made to apply partial bisimulation as introduced in [77], due to the fact that it establishes a stronger control relation beyond uncontrollable events.

Partial bisimulation is employed in this thesis to both express controllability and maximal permissiveness in a coinductive way which resolves several problems related to non-determinism, as illustrated by a number of examples in this chapter. Controllability as defined in [76] follows from Definition 2.6, although partial bisimulation is indeed a stronger property. Due to the fact that maximal permissiveness expresses how a distinction should be made between multiple solutions each having the controllability property, and because of the fact that controllability is expressed here by means of partial bisimilarity, it follows that maximal permissiveness should be expressed via partial bisimilarity as well.

For the formalization of required behavior in terms of control objectives, this thesis applies modal logic. Instead of a general existence theorem regarding a subset of the language of the plant, as detailed in Section 2.1, a more precise and more expressive formal specification language is applied here. Modal logic is an obvious choice for this purpose, since it integrates well with any automata-like behavioral description and because it is a well-known logical formalism in verification tasks such as model checking, as detailed in Section 1.4. The work in this research is certainly not the first to connect supervisory control theory and modal logic, but it is believed this is the first attempt to do so for a non-deterministic, maximally permissive interpretation of discrete event control, as discussed in more detail in Section 1.5. The precise choice of the applied logical formalism $\mathcal{F}$, which is specified in Definition 2.8, depends upon the fact that non-determinism is allowed, and the other control synthesis objectives such as maximal permissiveness, partial bisimulation, and solution uniqueness. The precise justifications for taking such a restricted logic, compared to for instance the μ-calculus, will become more clear when the examples in the succeeding sections are studied.

Requirements are specified using the modal logic $\mathcal{F}$ given in Definition 2.8, which is built upon the set of state-based formulas $\mathcal{B}$, in Definition 2.7.

Definition 2.7. The set of state-based formulas $\mathcal{B}$ is defined by the grammar:

$$\mathcal{B} ::= true \mid false \mid \mathcal{P} \mid \neg\mathcal{B} \mid \mathcal{B} \wedge \mathcal{B} \mid \mathcal{B} \vee \mathcal{B}$$

As indicated in Definition 2.7, state-based formulas are constructed from a straightforward Boolean algebra which includes the basic expressions $true$

and $false$, as well as a state-based property test for $p \in \mathcal{P}$. Formulas in $\mathcal{B}$ are then combined using the standard Boolean operators $\neg$, $\wedge$ and $\vee$.

Definition 2.8. The specification logic $\mathcal{F}$ is defined by the grammar:

$$\mathcal{F} ::= \mathcal{B} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{B} \vee \mathcal{F} \mid [\mathcal{E}]\mathcal{F} \mid <\mathcal{C}>\mathcal{F} \mid \Box\mathcal{F} \mid \Diamond\mathcal{B} \mid <\mathcal{E}> \mid dlf$$

The elements of the specification logic $\mathcal{F}$ are briefly considered here. Basic expressions in $\mathcal{B}$ function as the building blocks in the modal logic $\mathcal{F}$. Conjunction is included in unrestricted form, while disjunctive formulas are restricted to those having a state-based formula from $\mathcal{B}$ in the left-hand disjunct. This restriction guarantees unique synthesis solutions, since it enables a local state-based test for retaining the appropriate transitions, as illustrated in Figure 2.4. The formula $[e]f$ can be used to test whether f holds after every e-step, while the formula $<e>f$ is used to assess whether there exists an e-step after which f holds. These two operators thereby follow their standard semantics from Hennessy-Milner logic [34]. The restriction for the operator $<e>$ to be limited to a controllable event $e \in \mathcal{C}$ relates to the specific synthesis for a formula $<e>f$ and is detailed in Figure 2.6. An invariant formula $\Box f$ tests whether f holds in every reachable state, while a reachability expression $\Diamond b$ may be used to check whether there exists a path such that the state-based formula b holds at some state on this path. The argument of a reachability expression is restricted to a state-based formula $b \in \mathcal{B}$. This is due to the fact that an unrestricted reachability formula may be used to express a formula of type $<e>f$ with $e \in \mathcal{U}$, which leads to a problem concerning controllability, as illustrated in Figure 2.6. The two operators $\Box$ and $\Diamond$ are borrowed from Gödel-Löb logic [2], and follow the same semantics. As an addition to the formulas $<e>f$, a universal existence test $<e>$ is provided, which only tests whether an e-step exists. The argument e for the operator $<e>$ may be any event $e \in \mathcal{E}$. The deadlock-freeness expression dlf tests whether there exists an outgoing step of the current state. Combined with the invariant operator, the formula $\Box\, dlf$ may be used to include absence of deadlock in the enforced controlled behavior. Deadlock-freeness is not defined as a state-based expression in $\mathcal{B}$ since it requires information about the existence of outgoing transitions, which may have been removed during synthesis. These notions of validity are formalized in Definition 2.9.

Definition 2.9. Validity of formulas in $\mathcal{B}$ with respect to $\mathcal{K}$ (notation: $k \Vdash b$) is defined by the derivation rules shown below. Assume a Kripke-LTS $k = (X, L, \longrightarrow, x) \in \mathcal{K}$, $p \in \mathcal{P}$ and $b, c \in \mathcal{B}$ in the following derivation rules:

$$\frac{}{k \Vdash true} \qquad \frac{p \in L(x)}{k \Vdash p} \qquad \frac{k \nVdash b}{k \Vdash \neg b} \qquad \frac{k \Vdash b \quad k \Vdash c}{k \Vdash b \wedge c} \qquad \frac{k \Vdash b}{k \Vdash b \vee c} \qquad \frac{k \Vdash c}{k \Vdash b \vee c}$$

Definition 2.9 (cont.) Validity of formulas in $\mathcal{F}$ with respect to $\mathcal{K}$ (notation: $k \vDash f$) is defined in the following way. Assume that $k = (X, L, \longrightarrow, x) \in \mathcal{K}$, $b \in \mathcal{B}$, $e \in \mathcal{E}$, $x' \in X$ and $f, g \in \mathcal{F}$ in the following derivation rules:

$$\frac{k \Vdash b}{k \vDash b} \qquad \frac{k \vDash f \quad k \vDash g}{k \vDash f \wedge g} \qquad \frac{k \vDash b}{k \vDash b \vee f} \qquad \frac{k \vDash f}{k \vDash b \vee f}$$

$$\frac{\forall x \xrightarrow{e} x' \quad (X, L, \longrightarrow, x') \vDash f}{(X, L, \longrightarrow, x) \vDash [e]f} \qquad \frac{x \xrightarrow{e} x' \quad (X, L, \longrightarrow, x') \vDash f}{(X, L, \longrightarrow, x) \vDash \langle e \rangle f}$$

$$\frac{\forall x \longrightarrow^* x' \quad (X, L, \longrightarrow, x') \vDash f}{(X, L, \longrightarrow, x) \vDash \Box f} \qquad \frac{x \longrightarrow^* x' \quad (X, L, \longrightarrow, x') \vDash b}{(X, L, \longrightarrow, x) \vDash \Diamond b}$$

$$\frac{x \xrightarrow{e} x'}{(X, L, \longrightarrow, x) \vDash \langle e \rangle} \qquad \frac{x \xrightarrow{e} x'}{(X, L, \longrightarrow, x) \vDash dlf}$$

The synthesis problem may now be concisely formulated by means of the previous definitions. This is the key problem for which a solution is proposed in this chapter, while the validity of this solution is acknowledged in the next chapter.

Definition 2.10. Given plant model $k \in \mathcal{K}$ and control objective $f \in \mathcal{F}$, find the controlled system $s \in \mathcal{K}$ such that the following properties hold: 1) $s \vDash f$, 2) $s \preceq k$, 3) For all $k' \preceq k$ and $k' \vDash f$ it holds that $k' \preceq s$; or determine that such an s does not exist.

The three properties in Definition 2.10 are interpreted in the context of supervisory control synthesis as follows. Property 1 (*validity*) states that the synthesis result satisfies the synthesized specification. Property 2 (*controllability*) ensures that no accessible uncontrollable behavior is disabled during synthesis. Property 3 (*maximality*) states that synthesis removes the least possible behavior, and thereby induces maximal permissiveness. That is, the behavior of every alternative synthesis option (with regard to validity) is included in the behavior of the synthesis result.

2.3 Initial Expansion

The purpose of this section is to elaborate upon a number of intuitive notions which relate to a preceding expansion step before actual control is enforced.

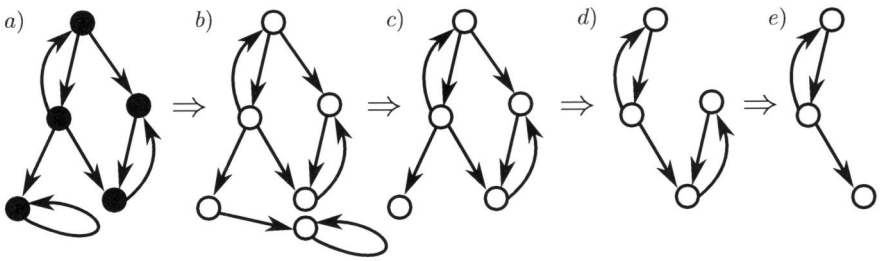

Figure 2.1: Abstraction of the synthesis process. The transition relation for the plant model in Figure 2.1a is augmented with reductions of the control specification in Figure 2.1b, which may induce an embedded unfolding. This new transition relation is then subjected to repeated transition removal in steps Figure 2.1c-2.1d, until a stable point has been reached in Figure 2.1e.

The approach developed during the research which underlies this thesis is to construct a projection of the transition relation of the plant onto the state-formula product space. The details of this projection are concerned in this section, and subsequently formalized in Section 2.4. This projection step is illustrated as the step 1a) $\rightarrow$ 1b) in Figure 2.1, and is followed by an iterative process of transition removal as shown in Figure 2.1 in steps 1c)-1e). Due to the fact that the precise construction of the projection is justified by the way transitions are removed later, the transition removal phase is intentionally left somewhat vague in some of the examples considered here. The precise definitions for transition removal are then considered in detail in later sections, which will lead to an integrated synthesis approach.

Given plant model $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ and control objective $f \in \mathcal{F}$, a new transition relation $\longrightarrow_0 \subseteq (X \times \mathcal{F}) \times \mathcal{E} \times (X \times \mathcal{F})$ is constructed such that $\longrightarrow_0$ is the transition relation of $S^0_{k,f} \in \mathcal{K}$, which has (x, f) as its initial state. The newly created $\mathcal{K}$-model $S^0_{k,f}$ will be the model from which transitions will be removed, until either a solution has been found, or it is determined that a solution does not exist. Therefore, the model $S^0_{k,f}$, as to be defined precisely in this chapter, will be referred to as the *synthesis starting point*. Intuitively, the transition relation $\longrightarrow_0$ will be set up in such a way that if $(x, f) \longrightarrow_0^* (x', f')$, then the model at location (x', f') will in the future have to be adapted such that f' is satisfied in location (x', f').

The way the new transition relation $\longrightarrow_0$ needs to be constructed is justified by observations regarding the formulas in $\mathcal{F}$. In fact, the validity of those formulas directly relates to the inductive build-up of $\mathcal{F}$ and to event labels. For instance, if a formula $[e]\, f$ is required to hold at state x, then af-

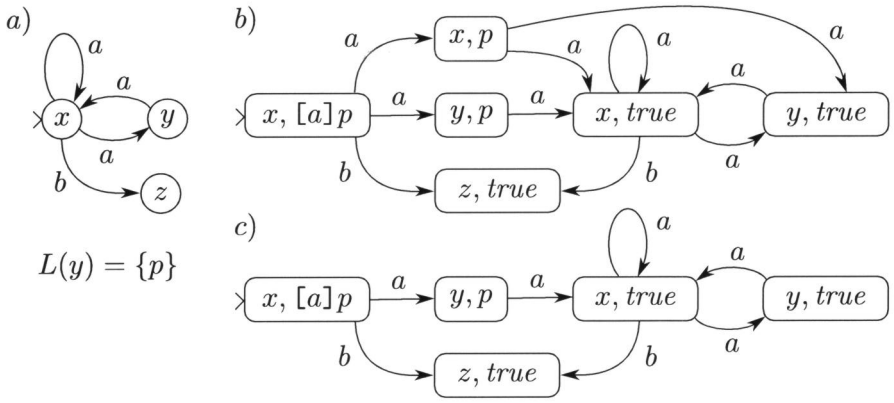

Figure 2.2: The model in Figure 2.2a is adapted in such a way that the control objective $[a]p$ becomes satisfied, as shown in Figure 2.2c. The intermediate expansion step in Figure 2.2b ensures maximally permissive synthesis.

ter each step $x \xrightarrow{e} x'$, f needs to hold at state x'. However, for each step $x \xrightarrow{e'} x'$ for $e \neq e'$, only $true$ needs to hold in x'. Such observations can be straightforwardly extended for operators such as conjunction and the invariant operator.

A first simple example is considered in Figure 2.2. Suppose that the model in Figure 2.2a needs to be adapted in such a way that the formula $[a]p$, for $p \in \mathcal{P}$, becomes satisfied. Note that this formula does not hold in the model in Figure 2.2a since an a-step to x exists and p is not assigned as a label to x. Observe that simply removing the a-loop at the initial state is not a maximally permissive solution. Instead, formula reductions are applied in Figure 2.2b to construct a new model where the original looping behavior is preserved at a later stage. The models in Figure 2.2a and Figure 2.2b are strictly bisimilar [31] and therefore satisfy the same formulas in $\mathcal{F}$. Figure 2.2b may now be adapted by removing all transition where p is assigned as a formula to the target state *and* where p does not hold at this state. The resulting model in Figure 2.2c now satisfies the control objective $[a]p$. This example, albeit very simple, shows precisely how the plant model in Figure 2.2a, using the expansion step in Figure 2.2b, may be modified to satisfy such a specification in modal logic.

The next operator to consider is $\langle e \rangle f$, which relates to non-determinism in a very particular way. Due to the fact that only existing behavior may be modified, no new e-step(s) may be added if those were not already present.

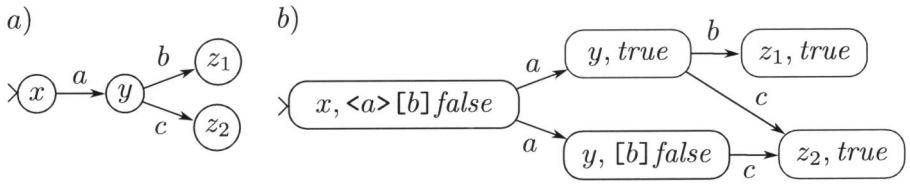

Figure 2.3: Synthesis for the control objective $<a>[b]\,false$ upon the model in Figure 2.3a, resulting in the model in Figure 2.3b, having duplicated behavior due to maximal permissiveness.

Furthermore, the only meaningful control synthesis task with regard to a formula of type $<e>f$ is to relay synthesis to the situation after each e-step, since no removal of a step makes such a formula *more* true. Another point worth mentioning is that if a formula $<e>f$ cannot be satisfied, due to the fact that in case synthesis after every e-step fails, the transition towards the state where $<e>f$ has been assigned may be removed. The latter situation is of particular importance to control synthesis. Also, maximal permissiveness needs to be taken into account. If $<e>f$ needs to be satisfied in x and a step $x \xrightarrow{e} x'$ exists where behavior needs to be limited in order to satisfy f in x', then maximality may be at stake if not all original behavior is preserved in x'. This situation is considered in more detail by the simple example in Figure 2.3. The model in Figure 2.3a is adapted in order to satisfy the control objective $<a>[b]\,false$, which results in the model in Figure 2.3b. In this example, simple removal of the b-step in Figure 2.3a is not an appropriate solution, since this would not lead to maximal permissiveness. An adequate formula reduction for formulas of type $<e>f$ therefore needs to take into account that f needs to be satisfied after an e-step, but also that original behavior needs to be preserved. This is reflected by a formula reduction of $<e>f$ to both f and $true$ after an e-step. This expands straightforwardly to a non-deterministic situation. Formula reductions ensure that after every e-step, f is attempted to be synthesized, but also that the behavior after each e-step is preserved. Formulas of type $<e>f$ are limited in such a way that $e \in \mathcal{C}$. This relates to various proof obligations concerning partial bisimulation, and is considered at a later stage.

The next operator to consider is the conjunction. A simple observation for the formula $[e]f \wedge [e]g$ indicates that $f \wedge g$ needs to be true after each e-step. An obvious generalization is that if f reduces to f' and if g reduces to g', then a new formula reduction needs to be defined where $f \wedge g$ reduces to $f' \wedge g'$. In fact, this is precisely the way formula reductions are defined for conjunction in this synthesis approach. This may, however, lead to problems which are

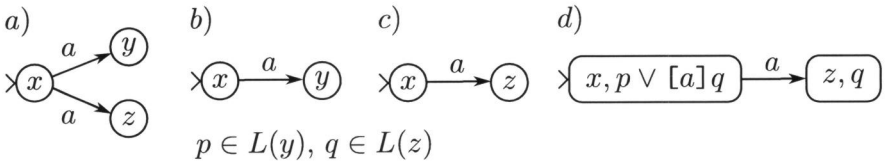

Figure 2.4: Synthesis for $[a]p \vee [a]q$ upon the model in Figure 2.4a would result in two different maximally permissive solutions, shown in Figures 2.4b and 2.4c. Instead, disjunctive formulas are restricted, as shown in Figure 2.4d.

not straightforwardly detectable. For instance, the formula $[e]f \wedge \langle e \rangle g$ may not have a solution if f cannot be satisfied in the state to which $f \wedge \mathit{true}$ is assigned, due to preserving original behavior after $\langle e \rangle g$. Whether such a solution exists cannot be detected at the expansion phase, and is therefore relayed to the transition removal phase. An example is considered in Figure 2.7. Formula reductions for conjunction are considered further in Figure 2.5, where invariant formulas are considered.

The focus now shifts to disjunction. The disjunction operator in Definition 2.8 is restricted such that only the right-hand disjunct may contain a formula of type $\mathcal{F}$, while the left-hand disjunct is limited to a $\mathcal{B}$-formula. The reason for this restriction is illustrated in Figure 2.4. There does not exist a unique maximally permissive adaptation of the model in Figure 2.4a, which satisfies $[a]p \vee [a]q$, for this non-deterministic case. Preserving original behavior, or any other solution similar to the one that was applied when defining reductions for formulas of type $\langle e \rangle f$, does not help. Unique and maximally permissive solutions are therefore only obtained via a restriction upon $\mathcal{F}$. The restricted left-hand formula $b \in \mathcal{B}$ may be readily validated on a state basis. If it holds then $b \vee f$ only reduces to true, thereby inducing maximal permissiveness. If it does not hold then the unrestricted right-hand disjunct $f \in \mathcal{F}$ is synthesized.

The next operator to consider is the invariant modality $\Box f$. A formula reduction is straightforwardly definable by observing that for $\Box f$ to be true at state x, it should hold that $\Box f$ is true at each $x' \in X$ such that $x \longrightarrow^* x'$. In addition, the formula reductions for f itself do apply, which leads to the following reduction principle: if f reduces to f' then $\Box f$ reduces to $\Box f \wedge f'$. Subsequent reductions under conjunction then lead to $(\Box f \wedge f') \wedge f''$, if f' reduces to f''. Such a reduction sequence may be infinitely expanding. For instance, the formula $\Box [e]p$ for $p \in \mathcal{P}$ first reduces to $\Box [e]p \wedge p$ after an e-step, and subsequently to $(\Box [e]) \wedge p \wedge \mathit{true}$ after two e-steps. Therefore,

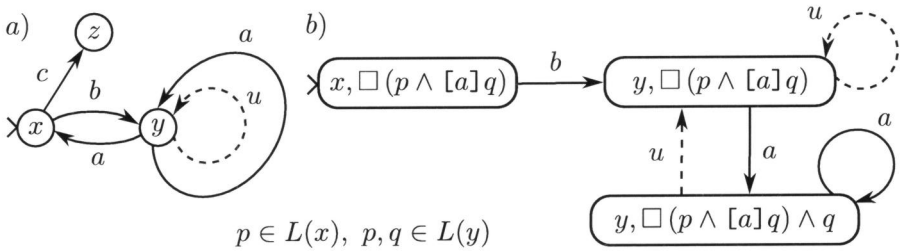

Figure 2.5: Synthesis for the control objective $\Box\,(p \wedge [a]\,q)$ upon the model in Figure 2.5a, resulting in the behavioral adaptation shown in Figure 2.5b. Note the application of normalization to inhibit indefinite expansion. Also, *true* conjuncts are omitted in this illustration for compactness.

reduction outcomes need to be normalized in such a way that double conjuncts are removed on a purely syntactic basis. Such a normalization step is straightforwardly computable and directly suppresses the aforementioned infinite expansion. An example for the synthesis of invariant formulas is considered in Figure 2.5.

A number of remaining formulas are now considered at the same time. These are the formulas $\Diamond\, b$, for $b \in \mathcal{B}$, and $<e>$, for $e \in \mathcal{E}$, and dlf. It is important to understand that no transition removal may aid in satisfying these formulas. Their real expressiveness lies in the fact that they may be used in other formulas. For instance, $\Box \Diamond\, marked$ to indicate that a marked state should always be reachable, or $\Box\, dlf$ to indicate that the entire synthesized controlled system should be deadlock-free. The value of formulas $<e>$ for $e \in \mathcal{E}$ lies in the fact that formulas $<e>f$ are restricted such that $e \in \mathcal{C}$, as mentioned earlier on, and as considered in more detail later on. Reductions for these formulas are defined in such a way that f reduces to *true* for $f \in \{\Diamond\, b \mid b \in \mathcal{B}\} \cup \{<e> \mid e \in \mathcal{E}\} \cup \{dlf\}$, and more of the intricate details are considered in Section 2.6.

2.4 Reduction Rules

A formal treatment of the formula reductions introduced in the previous section is provided in Definition 2.12. These reductions can not be defined in terms of modal expressions only; the original state of each transition also needs to be taken into account, as detailed in Figure 2.4. As discussed before, this is required to effectively implement synthesis for disjunctive formulas.

The left-hand side of a disjunction is restricted, as shown in Definition 2.8, to a basic formula $b \in \mathcal{B}$. Whether such a formula holds can be tested via a state-based evaluation only. This is used to determine which side of a disjunct will eventually be synthesized during the build-up of $\longrightarrow_0$. As a consequence, the initial transition relation $\longrightarrow_0 \subseteq (X \times \mathcal{F}) \times \mathcal{E} \times (X \times \mathcal{F})$ needs to be defined directly, and cannot be solely constructed as a composition between the original transition relation $\longrightarrow \subseteq X \times \mathcal{E} \times X$ and a possible reduction relation on modal expressions. As shown in Figure 2.5, it is necessary to inhibit infinite expansion of reductions of invariant expressions. For this purpose a formal definition of sub-formulas, as given in Definition 2.11, is applied.

Definition 2.11. For $k = (X, L, \longrightarrow, y)$ and $f \in \mathcal{F}$, the set of sub-formulas of f in state x (notation: $sub\,(x, f)$) is derived by the rules below. Assume that $x \in X$, $f, g, h \in \mathcal{F}$ and $b \in \mathcal{B}$ in the following definition:

$$\frac{}{f \in sub\,(x, f)} \qquad \frac{f \in sub\,(x, g)}{f \in sub\,(x, g \wedge h)} \qquad \frac{f \in sub\,(x, h)}{f \in sub\,(x, g \wedge h)}$$

$$\frac{f \in sub\,(x, g) \quad (X, L, \longrightarrow, x) \not\Vdash b}{f \in sub\,(x, b \vee g)} \qquad \frac{f \in sub\,(x, g)}{f \in sub\,(x, \Box\, g)}$$

Definition 2.12. Given $k = (X, L, \longrightarrow, y) \in \mathcal{K}$, $x, x' \in X$, $f, f' \in \mathcal{F}$ and $e \in \mathcal{E}$, an initial step $(x, f) \xrightarrow{e}_0 (x', f')$ is derived as shown below. Besides the previous instantiations, assume that $g, g' \in \mathcal{F}$, $b \in \mathcal{B}$ and $e' \in \mathcal{E}$ in the following derivation rules:

$$\frac{(x, f) \xrightarrow{e}_0 (x', f') \quad (x, g) \xrightarrow{e}_0 (x', g') \quad g' \in sub\,(x', f')}{(x, f \wedge g) \xrightarrow{e}_0 (x', f')}$$

$$\frac{(x, f) \xrightarrow{e}_0 (x', f') \quad (x, g) \xrightarrow{e}_0 (x', g') \quad g' \notin sub\,(x', f')}{(x, f \wedge g) \xrightarrow{e}_0 (x', f' \wedge g')}$$

$$\frac{x \xrightarrow{e} x'}{(x, b) \xrightarrow{e}_0 (x', true)} \qquad \frac{(X, L, \longrightarrow, x) \Vdash b \quad x \xrightarrow{e} x'}{(x, b \vee f) \xrightarrow{e}_0 (x', true)}$$

$$\frac{(X, L, \longrightarrow, x) \not\Vdash b \quad (x, f) \xrightarrow{e}_0 (x', f')}{(x, b \vee f) \xrightarrow{e}_0 (x', f')} \qquad \frac{x \xrightarrow{e} x'}{(x, [\,e\,]\, f) \xrightarrow{e}_0 (x', f)}$$

Definition 2.12 (cont.) The definition of $\longrightarrow_0$ is continued as follows:

$$\frac{x \xrightarrow{e} x' \quad e \neq e'}{(x, [e']f) \xrightarrow{e}_0 (x', true)} \qquad \frac{x \xrightarrow{e} x'}{(x, <e>f) \xrightarrow{e}_0 (x', f)}$$

$$\frac{x \xrightarrow{e} x'}{(x, <e'>f) \xrightarrow{e}_0 (x', true)} \qquad \frac{(x, f) \xrightarrow{e}_0 (x', f') \quad f' \in sub\,(x', f)}{(x, \Box f) \xrightarrow{e}_0 (x', \Box f)}$$

$$\frac{(x, f) \xrightarrow{e}_0 (x', f') \quad f' \notin sub\,(x', f)}{(x, \Box f) \xrightarrow{e}_0 (x', \Box f \wedge f')} \qquad \frac{x \xrightarrow{e} x'}{(x, <e'>) \xrightarrow{e}_0 (x', true)}$$

$$\frac{x \xrightarrow{e} x'}{(x, \Diamond b) \xrightarrow{e}_0 (x', true)} \qquad \frac{x \xrightarrow{e} x'}{(x, dlf) \xrightarrow{e}_0 (x', true)}$$

The derivation rules in Definition 2.12 are briefly considered here. A basic formula $b \in \mathcal{B}$ always reduces to $true$. Formula reductions are combined under conjunction, while the details surrounding the reductions of the left-hand disjunct have been considered in Figure 2.4. The right-hand reduction in a disjunctive formula is directly inherited, if the left-hand formula $b \in \mathcal{B}$ cannot be satisfied, as shown clearly in the fifth derivation rule in Definition 2.12. As detailed in Figure 2.2a, a formula $[e]f$ reduces to f after an e-step, while it reduces to $true$ after an e'-step with $e' \neq e$. The reduction for a formula $<e>f$ is somewhat more involved. After every e-step, an attempt is made to satisfy this formula, as signified by the reduction towards f. However, the original behavior after every e-step is also copied, which induces maximal permissiveness, as shown in Figure 2.3. This is the key difference between the synthesis for a formula $[e]f$ and a formula $<e>f$, which explains why the seventh and ninth rule are different. Synthesis for an invariant formula $\Box f$ has been considered in Figure 2.5. Both the invariant formula itself and its underlying reduct need to be present at the next state. The combination of reductions under conjunction then assures that appropriate modal expressions appear at later stages. The formulas $\Diamond b$, for $b \in \mathcal{B}$, $<e>$, for $e \in \mathcal{E}$, and dlf each reduce to a $true$ expression. Ensuring validity for these formulas relies upon the partial satisfiability test which is applied during synthesis.

Formulas of type $<e>f$ are synthesized in such a way that original behavior is left in place. This is illustrated in Figure 2.6. Synthesis for $<a>[b]\,false \wedge <a>[c]\,false$ upon the model in Figure 2.6a would not result in a maximally permissive solution if only the b and c steps were removed from the y-state.

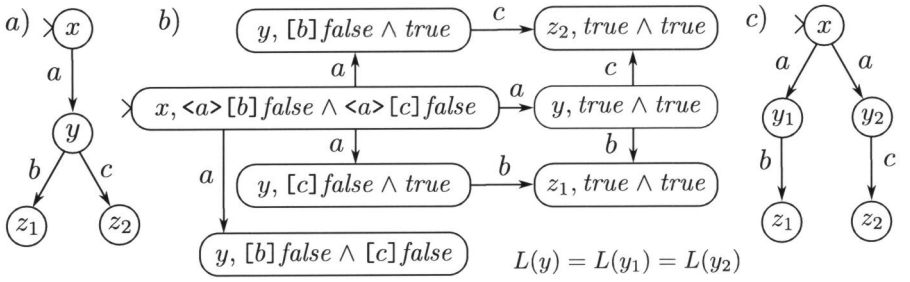

Figure 2.6: Synthesis for $\langle a \rangle [b] false \wedge \langle a \rangle [c] false$ upon the model in Figure 2.6a would not be maximally permissive if only the b-steps and c-steps were removed from the y-state. Instead, original behavior is copied, as shown in Figure 2.6b, which is the correct maximal synthesis result if $a \in \mathcal{C}$. If $a \in \mathcal{U}$, then Figure 2.6c is not a satisfying partial bisimulant of Figure 2.6b.

Instead, original behavior is left in place as shown in Figure 2.6b, where new non-deterministic a-steps are introduced by applying Definition 2.12, followed by transition removal. Note that this is only a viable solution if $a \in \mathcal{C}$. If $a \in \mathcal{U}$, then the model in Figure 2.6c is a satisfying partial bisimulant of the model in Figure 2.6a, but not of the model in Figure 2.6b. Hence, if $a \in \mathcal{U}$ then Figure 2.6b is not maximally permissive. Therefore, the restriction that $e \in \mathcal{C}$ is applied in Definition 2.8, for formulas of type $\langle e \rangle f$. The restriction that $b \in \mathcal{B}$ for formulas of type $\Diamond b$ is founded on the same basis. Assume that for each state $x \in X$, a state based property x is defined such that $x \in L(x)$. The formula $\langle a \rangle [b] false \wedge \langle a \rangle [c] false$ may then be expressed as $\Diamond (\neg x \wedge y \wedge \neg z_1 \wedge \neg z_2 \wedge [b] false) \wedge \Diamond (\neg x \wedge y \wedge \neg z_1 \wedge \neg z_2 \wedge [c] false)$ and has the same synthesis solution (modulo state names), as shown in Figure 2.6b. Consequently, the counterexample with regard to maximal permissiveness shown in Figure 2.6c applies.

2.5 Partial Satisfiability

In this section the test for transition removal is considered, which is illustrated by a number of examples, as part of the incremental process of transition removal. An expanded transition $(x, f) \xrightarrow{e} (x', f')$ is removed if it is not possible to satisfy f' in (x', f'), by means of transition removal, from that state onward. A number of cases for f' will now be considered, which leads to a formal definition of synthesizability in Section 2.6.

Consider the set of formulas $\mathcal{B}$. It is clear from Definition 2.7 and Definition 2.9 that validity of these formulas may be readily verified on a state-only basis. No transition removal from (x', b), for $b \in \mathcal{B}$, will aid in making (x', b) satisfy b. Therefore, $(x, f) \xrightarrow{e} (x', b)$, needs to be removed if $(x', b) \not\models b$.

Assume a formula $[e]f$, for $f \in \mathcal{F}$. A formula of type $[e]f$ can always be satisfied by transition removal by means of removing all outgoing e-steps. Clearly, this leads to the initial observation that each step $(x, g) \xrightarrow{e} (x', [e]f)$ should be retained, since it always is possible to satisfy $[e]f$. However, if $u \in \mathcal{U}$ and a step $(x', [u]f) \xrightarrow{u} (x'', f)$ exists such that f cannot be synthesized in (x'', f), then the step $(x, g) \xrightarrow{e} (x', [u]f)$ should be removed. The choice made in this synthesis setup is to evaluate partial satisfiability of $[e]f$ always as possible, but to introduce additional testing for reachable uncontrollable states during the process of transition removal.

The study of local partial satisfiability testing is continued by an analysis of formulas of type $<e>f$. Initially, it seems appropriate to check whether a step labeled e exists, followed by recursively evaluating whether the target state of this e-step can be made to satisfy f. However, a caveat appears on the path to a solution. Due to the fact that a copy of the original behavior via a formula reduction $<e>f \xrightarrow{e} true$ was introduced, it may be the case that the synthesizability evaluation for $<e>f$ returns $true$, due to the fact that a target $(x', true)$ can be made to satisfy f. However, if $<e>f$ is combined with $[e]g$ as in $<e>f \wedge [e]g$, it might not be possible to satisfy f in $(x', g \wedge true)$, due to additional transition removal induced by the synthesis of g. This essentially disables direct compositionality of the partial satisfiability test under conjunction. This possibility needs to be taken into account by requiring not only that an e-step exists where f can be satisfied, but indeed that an e-step exists towards an expanded state which corresponds to the $<e>f \xrightarrow{e} f$ formula reduction. A generalization of this notion of sub-formulas is formalized later Definition 2.11. An example for the synthesis of a combined $[e]/<e>$-formula is considered in Figure 2.7.

Due to the fact that synthesizability for f is evaluated in a combined state (x, f), dependency may be relieved in order to define synthesizability for conjunction. That is, formally it holds that $(x, g) \uparrow f_1 \wedge f_2$ if and only if $(x, g) \uparrow f_1$ and $(x, g) \uparrow f_2$, where $(x, g) \uparrow f$ is used to denote that f is synthesizable in state (x, g). The example in Figure 2.7 actually applies this. Synthesizability for a restricted disjunctive formula $b \vee f$ also needs to be defined in an appropriate way. Such a formula can always be satisfied if $(x, g) \models b$. However, if this is not the case then the definition should rely on the synthesizability for f in (x, g). That is, in the latter case it needs to be evaluated whether $(x, g) \uparrow f$.

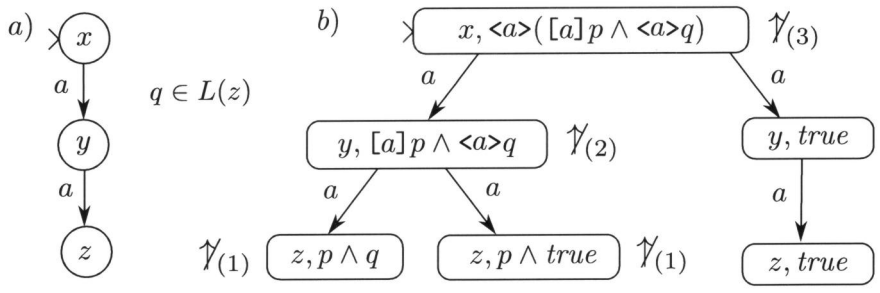

Figure 2.7: Synthesis for the control specification $<a>([a]p \wedge <a>q)$ upon the model shown in Figure 2.3a, resulting in Figure 2.3b. Synthesizability at each state for various iterations in the process of transition removal is indicated using $\uparrow_i$, for $i \in \{1, 2, 3\}$. This synthesis property necessitates the use of sub-formulas in the synthesizability for formulas of type $<e>f$, since the $(y, true)$-branch cannot be modified to satisfy $[a]p \wedge <a>q$.

Evaluating whether a formula $\Box f$ can be satisfied is significantly more difficult. Research revealed this depends upon the integrated synthesis approach where transitions are removed until synthesizability holds for the formula assigned to every reachable state. Within this context it is sufficient to define $(x, g) \uparrow \Box f$ solely as $(x, g) \uparrow f$. Intuitively, this corresponds to the notion that a formula $\Box f$ may be satisfied if the system can be modified in such a way that f may be satisfied at every reachable state.

The group of formulas $f \in \{\Diamond b \mid b \in \mathcal{B}\} \cup \{<e> \mid e \in \mathcal{E}\} \cup \{dlf\}$ allow a straightforward definition for synthesizability. This type of formula is either satisfiable or not; no transition removal may make such a formula satisfied if it is not satisfied already. However, transition removal in the synthesis for other formulas will possibly make such a formula unsatisfiable. Furthermore, if a formula of the aforementioned type does not hold at a particular state, then all in going transitions to that state need to be removed.

2.6 Transition Removal

This section formally defines the *synthesizability* condition $(x', f') \uparrow f'$, for the purpose of detecting whether a constructed step $(x, f) \xrightarrow{e} (x', f')$ needs to be removed or preserved, based on an evaluation of $(x', f') \uparrow f'$. A short reflection may be appropriate to verify that no simpler solution is available. An initial and provably sound observation is to retain a step $(x, f) \xrightarrow{e} (x', f')$

if a satisfying partial bisimulant exists at the target state. That is, a transition $(x, f) \xrightarrow{e} (x', f')$ should not be disabled if a $k' \in \mathcal{K}$ exists such that $k' \preceq (X, L, \longrightarrow, x')$ and $k' \vDash f'$. However, existence of such a satisfying partial bisimulant is not a feasible way from a computational perspective to express whether a constructed target state (x', f') should be retained after a step $(x, f) \xrightarrow{e} (x', f')$. Instead, an incremental approach is applied where iterative transition relations $\longrightarrow_0 \supseteq \longrightarrow_1 \supseteq \longrightarrow_2 \ldots$ are constructed until a stable point has been reached. The stability condition is then defined as the point where no constructed transitions are candidates for removal. A *synthesizability* test, as introduced in Section 2.5 (notation $(x', f') \uparrow_n f'$), will be used to assess whether a constructed step $(x, f) \xrightarrow{e} (x', f')$ should be retained in step n of the iterative synthesis process. Derivation rules for this test are listed in Definition 2.13, and are discussed in detail thereafter.

When studying Definition 2.13, it might be helpful to take a glance at Definition 2.14, where Definition 2.13 is applied to create succeeding iterations $S^n_{k,f}, S^{n+1}_{k,f}, S^{n+2}_{k,f}, \ldots$ of the synthesis result. Therefore, synthesizability needs to be defined in terms of a previously derived transition relation $\longrightarrow_n$. More precisely, the transition relation $\longrightarrow_n \subseteq (X \times \mathcal{F}) \times \mathcal{E} \times (X \times \mathcal{F})$ used in Definition 2.13 should be interpreted syntactically, without reference to a particular $n \in \mathbb{N}$. Definition 2.13 relies upon a syntactical notion of sub-formulas, as provided in Definition 2.11. This definition is used to resolve the issue addressed in Figure 2.7, which relates to copied original behavior, for the purpose of achieving maximal permissiveness.

Definition 2.13. For $k = (X, L, \longrightarrow, y) \in \mathcal{K}$, $b \in \mathcal{B}$, $x, x' \in X$, $f, f_1, f_2, g, g' \in \mathcal{F}$, $e \in \mathcal{E}$, and transition relation $\longrightarrow_n \subseteq (X \times \mathcal{F}) \times \mathcal{E} \times (X \times \mathcal{F})$, synthesizability can be derived as follows:

$$\frac{k \vDash b}{(x, g) \uparrow_n b} \qquad \frac{(x, g) \uparrow_n f_1 \quad (x, g) \uparrow_n f_2}{(x, g) \uparrow_n f_1 \wedge f_2} \qquad \frac{k \vDash b}{(x, g) \uparrow_n b \vee f} \qquad \frac{(x, g) \uparrow_n f}{(x, g) \uparrow_n b \vee f}$$

$$\frac{}{(x, g) \uparrow_n [e] f} \qquad \frac{(x, g) \xrightarrow{e}_n (x', g') \quad f \in sub\,(x', g') \quad (y, g') \uparrow_n f}{(x, g) \uparrow_n \langle e \rangle f}$$

$$\frac{(x, g) \uparrow_n f}{(x, g) \uparrow_n \Box f} \qquad \frac{(x, g) \longrightarrow^*_n (x', g') \quad (X, L, \longrightarrow, x') \vDash b}{(x, g) \uparrow_n \Diamond b}$$

$$\frac{(x, g) \xrightarrow{e}_n (x', g')}{(x, g) \uparrow_n \langle e \rangle} \qquad \frac{(x, g) \xrightarrow{e}_n (x', g')}{(x, g) \uparrow_n dlf}$$

The first derivation rule in Definition 2.13 expresses how synthesizability for a basic formula $b \in \mathcal{B}$ in (x, g) directly depends upon the validity of this formula at that particular state. If both conjuncts f_1 and f_2 can be synthesized, then the combination $f_1 \wedge f_2$ is synthesizable in (x, g). Synthesizability for disjunction is derived directly from its operands, as indicated by the third and fourth rule. A formula $[e]f$ is always synthesizable since every outgoing e-step may be removed. However, during the transition removal phase it needs to be taken into account that e may be uncontrollable. For $<e>f$ such that $e \in \mathcal{C}$, an example is considered in Figure 2.7. This example shows how multiple iterations of transition removal are required to determine that the model in Figure 2.7a cannot be adapted to satisfy the formula $<a>([a]p \wedge <a>q)$. Due to the introduction of a copy of the original behavior in the branch at the right-hand side in Figure 2.7b, it might seem that the formula $<a>([a]p \wedge <a>q)$ is still satisfiable, since an outgoing a-step exists. Therefore, synthesizability for a formula of type $<e>f$ requires that f is a sub-formula of the formula assigned to the relevant step, as shown in Definition 2.13.

Synthesizability for a formula $\Box f$ in a combined state (x, g) requires that f is synthesizable in (x, g). The remaining expressions $\Diamond b$, $<e>$ and dlf are only synthesizable if they can be directly satisfied. As justified by the intuition that no transition removal will make such an expression *more* true.

2.7 Synthesis Construction

Now, the succeeding iterations in the computational approximations $\longrightarrow_1$, $\longrightarrow_2, \ldots$, may be defined, for which $\longrightarrow_0$ has already been given in Definition 2.12. The corresponding synthesis results $S^1_{k,f}$, $S^2_{k,f}$, ... are also detailed in Definition 2.14.

Definition 2.14. For $k = (X, L, \longrightarrow, x)$, $f \in \mathcal{F}$ and $n \in \mathbb{N}$, the n-th iteration $S^n_{k,f}$ in the computational synthesis process is defined as follows:

$$S^n_{k,f} = (X \times \mathcal{F}, L_{\chi\mathrm{F}}, \longrightarrow_n, (x, f))$$

where $L_{\chi\mathrm{F}}(y, g) = L(y)$ for all $y \in X$ and $g \in \mathcal{F}$. The transition relation $\longrightarrow_n$ is defined for $y, y' \in X$, $g, g' \in \mathcal{F}$ and $e \in \mathcal{E}$ as follows:

$$\frac{(y, g) \xrightarrow{e}_n (y', g') \quad e \in \mathcal{U}}{(y, g) \xrightarrow{e}_{n+1} (y', g')}$$

$$\frac{(y, g) \xrightarrow{e}_n (y', g') \quad \forall\, v \in \mathcal{U}^* : \forall\, (y', g') \xrightarrow{v}{}^*{}_n (y'', g'') : (y'', g'') \uparrow_n g''}{(y, g) \xrightarrow{e}_{n+1} (y', g')}$$

The XF in L_{XF} refers to the fact that this new labeling function is defined upon the $X \times \mathcal{F}$ product space. The first rule in Definition 2.14 states that every uncontrollable step should be preserved. The second rule expresses the actual synthesis functionality, where a transition is only preserved if synthesizability holds at each state which is reachable by uncontrollable steps.

Transition removal in the succeeding iterations of the transition relation $\longrightarrow_0, \longrightarrow_1, \longrightarrow_2, \ldots$ proceeds until no more target states of individual transitions are considered candidates for removal. That is, if the synthesizability predicate holds at every reachable state. If a plant model $k \in \mathcal{K}$ has finitely many transitions, this process is terminating. This premise for completeness is formalized in Definition 2.15.

Definition 2.15. For $k = (X, L, \longrightarrow, x) \in \mathcal{K}$, $f \in \mathcal{F}$ and $n \in \mathbb{N}$ it holds that $S^n_{k,f}$ is *complete* if for all $(x, f) \longrightarrow^*_n (x', f')$ it holds that $(x', f') \uparrow_n f'$.

If the condition of completeness as stated in Definition 2.15 cannot be reached, a solution to the synthesis problem in Definition 2.10 does not exist.

2.8 Closing Remarks

The synthesis setup formally defined in this chapter is intended to be supported by the examples to such an extent that the main intuitions are clear. The projection of the original transition relation onto a new relation over the state-formula product space clearly establishes a correspondence that expresses which formulas should hold at which states. Formula reductions provide a neat way to define this projection and contribute to the soundness of synthesis due to the inductive way in which they are defined. The framework of transition removal by means of a synthesizability test at the target states of transitions allows for a computationally feasible way to achieve control in a way that is directly related to maximal permissiveness and synthesis correctness, as shown in the next chapter. This fixpoint approach clearly defines at which point a satisfying solution is obtained, and furthermore eliminates a post-synthesis test as to whether a successful solution has been achieved. The restrictions imposed upon $\mathcal{F}$, which directly relate to synthesis correctness, are assumed to be sufficiently clear to the reader now. Regarding further developmenst, it certainly needs to be taken into account that some of these restrictions may be lifted once new insights are obtained in future research.

Chapter 3

Correctness and Computation

Since formal definitions of the synthesis setup have all been provided in Chapter 2, the main objective now becomes the assessment of the validity of the proposed theory. In this chapter, this will be interpreted in a broad sense. First of all, it will be ensured that the technical formalities are satisfied. This means that synthesis solutions are shown to guarantee the control objective, given as an $\mathcal{F}$-expression. In addition, controllability and maximal permissiveness are shown to hold for synthesis outcomes. These aforementioned validity aspects may be summarized as the *soundness* of the synthesis theory. Furthermore, it is shown that if a solution exists, it will eventually be found, thereby covering the *completeness* aspect. An algorithmic representation of the synthesis method follows at a later stage in this chapter. This part bridges the gap between the mathematical formalization and an actual implementation. Subsequently, it is shown how Ramadge-Wonham supervisory control theory [76] can be expressed using this synthesis theory. The broad interpretation of correctness also justifies the inclusion of case studies in this chapter.

Considered in somewhat more detail, this chapter is set up as follows: mathematical proofs can be found in Section 3.1, while the computer verified proofs are included in Section 3.4. Besides these two proof-related sections, the remainder of this chapter is organized as follows. The synthesis construction is represented in algorithmic form, including an analysis of its computational complexity, in Section 3.2. The connection to Ramadge-Wonham supervisory control is made in Section 3.3. It is detailed how a traditional Ramadge-Wonham control synthesis problem may be expressed using the theories proposed in the previous chapter. Application of synthesis is considered in Section 3.5 and Section 3.6 by means of two case studies.

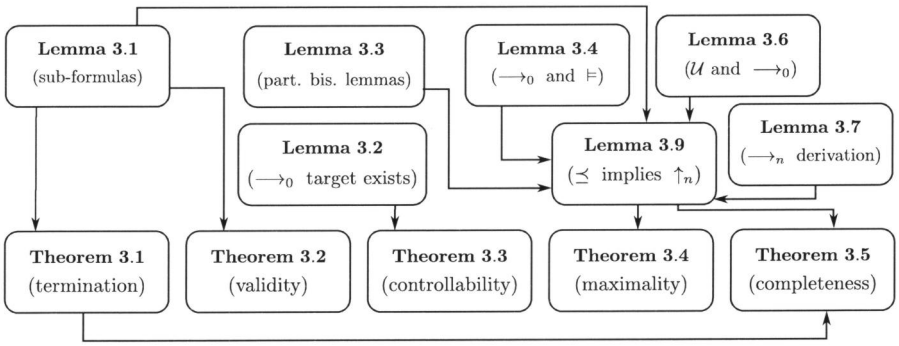

Figure 3.1: A graphical illustration of the dependencies between the most important lemmas and theorems within Section 3.1. In-going arrows represent a dependency upon the proof entity from which the arrow originates.

3.1 Correctness Proofs

This section mainly contains proofs, which are henceforth considered in terms of formal mathematics in in Section 3.4. Theorem 3.1 shows that the synthesis construction in Definition 2.14 is terminating; that is: it leads to the required synthesis result after the application of a finite number of derivation steps. The strategy applied in Theorem 3.1 is to define a finite overapproximation of the number of transitions in $\longrightarrow_0$. This leads to the conclusion that only finitely many transition removal steps will eventually take place. Subsequently it is shown that the synthesis construction satisfies the three main results from Definition 2.10: validity (Theorem 3.2), controllability (Theorem 3.3) and maximal permissiveness (Theorem 3.4). The final key result in this section is Theorem 3.5, where it is shown that if a solution exists, it will eventually be found. Figure 3.1 details the dependencies between several lemmas and theorems which can be found in this section. The reader who is familiar with computer verified proofs may wish to study the mathematical proofs and the formalized proofs in parallel. The latter proofs can be found at the following location[3] and are considered in more detail in Section 3.4.

`https://github.com/ahulst/deds`

[3]The author will strive to maintain this facility in the foreseeable future. If the Coq-proofs are not available anymore at this location, the reader is kindly requested to contact the author directly in order to obtain these.

A number of technical results for sub-formulas are developed first:

Lemma 3.1. For $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ the following results hold regarding sub-formulas:

(a) For $f \in sub\,(x, g)$ and $k \vDash g$ it holds that $k \vDash f$;

(b) For $f \wedge g \in sub\,(x, h)$ it holds that $f \in sub\,(x, h)$ and $g \in sub\,(x, h)$;

(c) For $[e]\,f \in sub\,(x, g)$ and $(x, g) \xrightarrow{e}_0 (x', g')$ it holds that $f \in sub\,(y, g')$, for all $x' \in X$;

(d) For $b \vee f \in sub\,(x, g)$ and $k \nVdash b$ it holds that $f \in sub\,(x, g)$;

(e) For $\Box f \in sub\,(x, g)$ it holds that $f \in sub\,(x, g)$;

(f) For $\Box f \in sub\,(x, g)$ and $(x, g) \xrightarrow{e}_0 (x', g')$ it holds that $\Box f \in sub\,(x', g')$, for all $x' \in X$; and

(g) For $(x, h) \uparrow_n g$ and $f \in sub\,(x, g)$ it holds that $(x, h) \uparrow_n f$, for all $n \in \mathbb{N}$; and

(h) For $f \in sub\,(x, g)$ and $g \in sub\,(x, h)$ it holds that $f \in sub\,(x, h)$.

Proof. These results can be obtained by induction towards the derivation depth in Definition 2.11. □

Theorem 3.1. The synthesis construction in Definition 2.14 is terminating.

Proof. The following result is shown: if $k = (X, L, \longrightarrow, x) \in \mathcal{K}$, for finite $\longrightarrow$, then $S^0_{k,f}$ has finitely many transitions. It therefore needs to be shown that the number of transitions in $\longrightarrow_0$ is finite. Every succeeding synthesis iteration removes steps until a stable point has been reached. Finiteness of $\longrightarrow_0$ is therefore sufficient to prove termination. Given that $\longrightarrow$ is finite, it needs to be shown that the following set is finite:

$$\{(x', f') \in X \times \mathcal{F} \mid (x, f) \longrightarrow_0^* (x', f')\}$$

This result is proven directly by induction towards the structure of f. For the cases where $f \equiv b$ or $f \equiv \Diamond b$, for $b \in \mathcal{B}$, or $f \equiv \langle e \rangle$ or $f \equiv dlf$, the set $\{(x, f)\} \cup (X \times \{true\})$ includes all newly constructed states reachable by $\longrightarrow_0^*$. Note that this is an overapproximation but still a finite set, if X is restricted to the states reachable by $\longrightarrow^*$.

If $f \equiv f_1 \wedge f_2$, then by induction the following sets are derived:

$$\begin{array}{lcl} C_1 & = & \{(x', f') \in X \times \mathcal{F} \mid (x, f_1) \longrightarrow_0^* (x', f')\} \\ C_2 & = & \{(x', f') \in X \times \mathcal{F} \mid (x, f_2) \longrightarrow_0^* (x', f')\} \end{array}$$

Subsequently, it needs to be shown that the set $C_1 \cup \{(y, g \wedge h) \mid (y, g) \in C_1, (y, h) \in C_2\}$ is finite. By induction towards the length of $(x, f_1 \wedge f_2) \longrightarrow_0^* (y, f')$, it can be shown that either $(y, f') \in C_1$, if the fourth rule in Definition 2.12 was applied, or $f' \equiv g \wedge h$ and $(y, g) \in C_1$ and $(y, h) \in C_2$. The next case to consider in the induction proof is when $f \equiv b \vee g$, for $b \in \mathcal{B}$. As an intermediate step, the following set is defined:

$$C = \{(x', f') \in X \times \mathcal{F} \mid (x, g) \longrightarrow_0^* (x', g')\}$$

which is finite by induction. Hence, the set $\{(x, b \vee g)\} \cup C \cup (X \times \{true\})$ is also finite. The inductive cases for $f \equiv [e] f'$ and $f \equiv \langle e \rangle f'$ are considered in parallel. By induction, for each step $(x, [e] f') \xrightarrow{e}_0 (y, f')$ and for each step $(x, \langle e \rangle f') \xrightarrow{e}_0 (y, f')$ a finite set C_y can be derived by induction. These sets C_y may then be combined under union and combined with $X \times \{true\}$ to finitely define the set of states reachable under $\longrightarrow_0$.

The final case for the inductive proof is where $f \equiv \Box f'$, which requires an additional helper function D defined below. Assume that $f \in \mathcal{F}$, $C \subseteq X \times \mathcal{F}$ and $n \in \mathbb{N}$ in the following inductive definition:

$$\begin{array}{lcl} D(f, C, 0) & = & X \times \{\Box f\} \\ D(f, C, n+1) & = & D(f, C, n) \cup \{(x, g \wedge h) \mid (x, g) \in D(f, C, n),\ h \in C\} \end{array}$$

If $C = \{(x', f') \in X \times \mathcal{F} \mid (x, f) \longrightarrow_0^* (x', f')\}$ then the finite overapproximation $D(f, C, |C|)$, where $|C|$ indicates the number of elements in C, contains all states reachable from $(x, \Box f)$ over $\longrightarrow_0^*$.

It is first shown that for all $(x, \Box f) \longrightarrow_0^* (x', g)$ there exists an $n \in \mathbb{N}$ such that $g \in D(f, C, n)$. If induction is applied to the structure of g, there are two relevant cases: 1) if $g \equiv \Box f$ then $(x, g) \in D(f, C, 0)$ and, 2) if $g \equiv g_1 \wedge g_2$ then there exists an $n \in \mathbb{N}$ such that $(x', g_1) \in D(f, C, n)$ and since $(x', g_2) \in C$, it holds that $(x', g_1 \wedge g_2) \in D(f, C, n+1)$.

Subsequently, the following is shown: if $(x, \Box f) \longrightarrow_0^* (x', g)$ then $(x', g) \in D(f, C, |C|)$. Clearly, as was just shown, there exists an $n \in \mathbb{N}$ such that $(x', g) \in D(f, C, n)$. However, if $(x', g) \in D(f, C, n)$ and $n > |C|$ then the derivation rules in Definition 2.12 show that $g \equiv g_1 \wedge g_2$ and $g_2 \notin sub\,(x', g_1)$, if $(x', g) \notin D(f, C, m)$ for all $m < n$. However, for all $(x, f) \longrightarrow_0^* (x', f')$ it holds that $(x', f') \in C$. If $(x', g) \notin D(f, C, m)$ for all $m < n$ then g has n different conjuncts and since $n > |C|$ it holds that $g_2 \in sub\,(x', g_1)$. $\Box$

If the synthesis result satisfies the completeness premise, and thus when the synthesizability predicate holds at every reachable state, then the synthesis result satisfies the specification for desired behavior, as shown in Theorem 3.2.

Theorem 3.2. If $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ and $f \in \mathcal{F}$ and $n \in \mathbb{N}$ such that $S^n_{k,f}$ is complete, then $S^n_{k,f} \vDash f$.

Proof. The following theorem will be proven: if $g \in \mathcal{F}$ such that $f \in sub\,(x, g)$ and $S^n_{k,g}$ is complete, then $S^n_{k,g} \vDash f$. This is sufficient, since $f \in sub\,(x, f)$. Induction is applied towards the structure of $f \in \mathcal{F}$, thereby generalizing over g and x. For each inductive case it holds that $(x, g) \uparrow_n g \Rightarrow (x, g) \uparrow_n f$, by Lemma 3.1(g).

If $f \equiv b$, for $b \in \mathcal{B}$, then $(x, g) \uparrow_n b$ and thus $k \vDash b$, which implies $S^n_{k,g} \vDash b$. If $f \equiv f_1 \wedge f_2$ then $f_1 \in sub\,(x, g)$ and $f_2 \in sub\,(x, g)$, which leads to $S^n_{k,g} \vDash f_1$ and $S^n_{k,g} \vDash f_1$ by induction. If $f \equiv b \vee f'$ then a distinction is made between two cases: 1) if $k \vDash b$ then $S^n_{k,g} \vDash b$, 2) if $k \nvDash b$ then by Lemma 3.1(d) it holds that $f' \in sub\,(x, g)$, which by induction leads to $S^n_{k,g} \vDash f'$.

If $f \equiv [e]\, f$, then assume there exists a step $(x, g) \xrightarrow{e}_n (x', g')$, and define $k' = (X, L, \longrightarrow, x')$. Since $(x, g) \xrightarrow{e}_0 (x', g')$, by Definition 2.14, it holds that $f \in sub\,(x', g')$, by Lemma 3.1(c). By the induction hypothesis for f', it can now be derived that $S^n_{k',g'} \vDash f'$. Note that the induction premise for completeness for $S^n_{k',g'}$ follows from the assumption $(x, g) \xrightarrow{e}_n (x', g')$ and Definition 2.15. If $f \equiv \langle e \rangle f'$, then by Definition 2.13 there exists a step $(x, g) \xrightarrow{e}_n (x', g')$ such that $f' \in sub\,(x', g')$ and $(X, L, \longrightarrow, x') \vDash g'$, using the abbreviation $k' = (X, L, \longrightarrow, x')$. Since $S^n_{k',g'}$ is complete, induction can be applied to derive $S^n_{k',g'} \vDash f'$.

If $f \equiv \Box f'$, then there exists a sequence of steps $(x, g) \longrightarrow^*_n (x', g')$ such that $f' \in sub\,(x', g')$ by Lemmas 3.1(e) and 3.1(f). Set $k' = (X, L, \longrightarrow, x')$. Due to the fact that $(x, g) \longrightarrow^*_n (x', g')$, $S^n_{k',g'}$ is complete, it follows that $S^n_{k',g'} \vDash f'$, by induction. For the cases $f \equiv \Diamond b$, for $b \in \mathcal{B}$, or $f \equiv \langle e \rangle$ or $f \equiv dlf$, the result $S^n_{k,g} \vDash f$ follows directly from $(x, g) \uparrow_n f$. □

Lemma 3.2. For each $f \in \mathcal{F}$, $e \in \mathcal{E}$ and $x \xrightarrow{e} x'$ there exists an $f' \in \mathcal{F}$ such that $(x, f) \xrightarrow{e}_0 (x', f')$.

Proof. By induction towards the structure of f. □

Controllability then follows directly from Lemma 3.2 and the construction in Definition 2.14, as shown in Theorem 3.3.

Theorem 3.3. For $k = (X, L, \longrightarrow, x) \in \mathcal{K}$, $f \in \mathcal{F}$ and $n \in \mathbb{N}$ it holds that $S^n_{k,f} \preceq k$.

Proof. It will be shown that $S^n_{k,f} \preceq_R k$ by defining R as follows:

$$R = \{((y, g), y) \mid (x, f) \longrightarrow^*_n (y, g)\}$$

Clearly $((x, f), x) \in R$. Assume that $((y, g), y) \in R$. If $(y, g) \xrightarrow{e}_n (z, g')$ then $y \xrightarrow{e} z$ by Definitions 2.14 and 2.12, such that $((z, g), z) \in R$.

If $y \xrightarrow{e} z$, for $e \in \mathcal{U}$, then by Lemma 3.2 there exists a formula-reduct $(y, g) \xrightarrow{e}_0 (z, g')$. Since $e \in \mathcal{U}$, it follows from the construction of $\longrightarrow_n$ in Definition 2.14 that $(y, g) \longrightarrow_n (y', g')$. □

Partial bisimilarity is related to Definition 2.12 and Definition 2.11, but also implies validity for formulas in $\mathcal{B}$. These results are listed in Lemma 3.3.

Lemma 3.3. For $k' = (X', L', \longrightarrow', x') \in \mathcal{K}$ and $k = (X, L, \longrightarrow, x) \in \mathcal{K}$, such that $k' \preceq k$, the following results hold:

(a) For all $b \in \mathcal{B}$ it holds that $k' \vDash b$ if and only if $k \vDash b$;

(b) For all $f, g \in \mathcal{F}$ it holds that $f \in sub(x', g)$ if and only if $f \in sub(x, g)$; and

(c) For all $f, f' \in \mathcal{F}$, $e \in \mathcal{E}$ and $x' \xrightarrow{e} y'$ and $x \xrightarrow{e} y$ and $(X', L', \longrightarrow', y') \preceq (X, L, \longrightarrow, y)$ it holds that $(x', f) \xrightarrow{e}{}'_0 (y', f')$ if and only if $(x, f) \xrightarrow{e}_0 (y, f')$.

Proof. Result (a) is obtained by induction towards the structure of $b \in \mathcal{B}$ in Definition 2.7, result (b) is derived by induction towards the derivation depth in Definition 2.11, and result (c) is shown by induction towards the derivation depth in Definition 2.12. □

Lemmas 3.4 and 3.5 detail how existence of a formula-reduct relates to validity. Lemma 3.5 can be considered a specific instance of Lemma 3.4, where the sub-formula property is required.

Lemma 3.4. For each $k = (X, L, \longrightarrow, x) \in \mathcal{K}$, $f \in \mathcal{F}$, $e \in \mathcal{E}$ and $x' \in X$ such that $k \vDash f$ and $x \xrightarrow{e} x'$, there exists an $f' \in \mathcal{F}$ such that $(x, f) \xrightarrow{e}_0 (x', f')$ and $(X, L, \longrightarrow, x') \vDash f'$.

Proof. By induction towards the structure of $f \in \mathcal{F}$. □

Lemma 3.5. For each $k = (X, L, \longrightarrow, x) \in \mathcal{K}$, $e \in \mathcal{E}$ and $f, g \in \mathcal{F}$ such that $<e>f \in sub\,(x, g)$ and $k \vDash g$, there exist $x' \in X$ and $g' \in \mathcal{F}$ such that $(x, g) \xrightarrow{e}_0 (x', g')$ and $f \in sub\,(x', g')$ and $(X, L, \longrightarrow, x') \vDash g'$.

Proof. By induction towards the derivation depth of $<e>f \in sub\,(x, g)$ in Definition 2.11, using Lemma 3.4 for both cases for conjunction to cover the opposite conjunct (i.e. the case not covered by induction). □

If $(x, f) \xrightarrow{e}_0 (x', g)$ and $(x, f) \xrightarrow{e}_0 (x', h)$ such that $e \in \mathcal{U}$, then $g \equiv h$. This determinism property gives rise to a specific result between formula-reducts and validity, as shown in Lemma 3.6.

Lemma 3.6. For $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ and $x \xrightarrow{e} x'$ for some $e \in \mathcal{U}$ and $x' \in X$, then for each $f, f' \in \mathcal{F}$ such that $(x, f) \xrightarrow{e}_0 (x', f')$ and $k \vDash f$, it holds that $(X, L, \longrightarrow, x') \vDash f'$.

Proof. By induction towards the derivation depth of $(x, f) \xrightarrow{e}_0 (x', f')$. □

Lemma 3.7. If $(x, f) \xrightarrow{e}_0 (x', f')$ and for any $n \in \mathbb{N}$ it holds that $(x, f) \xrightarrow{e}_n (x', f')$ if for all $m < n$ and for all $v \in \mathcal{U}^*$ and $(x', f') \xrightarrow{v}{}^*{}_m (x'', f'')$ it holds that $(x'', f'') \uparrow_m f''$.

Proof. This result follows from the construction in Definition 2.14 and by strong induction towards n. □

Lemma 3.8. If $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ and $k' = (X', L', \longrightarrow', x') \in \mathcal{K}$ such that $k' \preceq k$ and if $f, f' \in \mathcal{F}$ and $n \in \mathbb{N}$ and $v \in \mathcal{U}^*$ such that $(x, f) \xrightarrow{v}{}^*{}_n (y, f')$ and $(X', L', \longrightarrow', x') \vDash f$, then there exists an $y' \in X'$ such that $x' \{\longrightarrow'\}^* y'$ and $(X', L', \longrightarrow', y') \preceq (X, L, \longrightarrow, y)$ and $(X', L', \longrightarrow', y') \vDash f'$.

Proof. This result follows from Definition 2.6, Lemma 3.6 and induction towards the length of v. □

Lemma 3.9 details an important result between the semantic notion of synthesizability (i.e. existence of a satisfying partial bisimulant) and the syntactic notion as given in Definition 2.13.

Lemma 3.9. For $k = (X, L, \longrightarrow, x) \in \mathcal{K}$, $k' \in \mathcal{K}$ and for $f, g \in \mathcal{F}$ such that $f \in sub\,(x, g)$ and $k' \vDash g$, it holds that $(x, g) \uparrow_n f$.

Proof. The proof is somewhat involved. Strong induction is applied towards n, thereby generalizing over all other variables, and thereafter a nested induction towards the structure of f, thereby generalizing over g, x and k'. A number of cases for f can be resolved directly using the induction hypothesis for f, and do not depend upon the induction towards n. These are the cases for $f \equiv b \in \mathcal{B}$, $f \equiv f_1 \wedge f_2$, $f \equiv b \vee f'$, $f \equiv [e]\, f'$ and $f \equiv \Box f'$. Lemma 3.1 is required to resolve these cases.

Now the other inductive cases for f are considered, where the key differences for the cases under the inductions $n \equiv 0$ and $n + 1$ are highlighted. Assume that $k' = (X', L', \longrightarrow', x')$ and $R \subseteq X' \times X$ such that $k' \preceq_R k$.

If $f \equiv \langle e \rangle f'$, with $e \in \mathcal{C}$, then by Lemma 3.1(a) it holds that $k' \vDash \langle e \rangle f'$. By Lemma 3.5 there exists a step $x' \xrightarrow{e} y'$ and a formula-reduction $(x', g) \xrightarrow{e}_0 (y', g')$ such that $(X', L', \longrightarrow', y') \vDash g'$ and $f' \in sub\,(y', g')$. By partial bisimulation there exists a step $x \xrightarrow{e} y$ such that $(y', y) \in R$. Definition 2.12 then allows the construction of a step $(x, g) \xrightarrow{e}_0 (y, g')$. By the induction hypothesis for f', it holds that $(y, g') \uparrow_n f'$. For the case $n \equiv 0$, this is sufficient to derive that $(x, g) \uparrow_n \langle e \rangle f'$. For the inductive case for n, Lemma 3.7 needs to be applied to construct a step $(x, g) \xrightarrow{e}_n (y, g')$. This requires that for all $m < n$ and $v \in \mathcal{U}^*$ such that $(y, g') \xrightarrow{v}{}^*{}_m (z, g'')$ it holds that $(z, g'') \uparrow_m g''$. This can be resolved by Lemma 3.8 and the induction hypothesis for n.

The case for $f \equiv \Diamond b$, with $b \in \mathcal{B}$ is essentially a generalization for the case for $f \equiv \langle e \rangle f'$. If $k' \vDash \Diamond b$, then there exists a $x' \{\longrightarrow'\}^* y'$ such that $(X', L', \longrightarrow', y') \vDash b$, and by Lemma 3.4 there exists a $g' \in \mathcal{F}$ such that $(X', L', \longrightarrow', y') \vDash g'$. This allows the construction of $(x, g) \longrightarrow_0^* (y, g')$ by Definition 2.12, such that $(X, L, \longrightarrow, y) \vDash b$. For the inductive case for n it holds that $(x, g) \longrightarrow_n^* (y, g')$, which is sufficient to derive $(x, g) \uparrow_n \Diamond b$. The two remaining cases for $f \equiv \langle e \rangle$ and $f \equiv dlf$ are essentially instances for the case $f \equiv \langle e \rangle f'$. □

The main result required for the maximality theorem has been established in Lemma 3.9. Maximal permissiveness is then straightforwardly derivable.

Theorem 3.4. For $k', k \in \mathcal{K}$ such that $k' \preceq k$ and $k' \vDash f$ and for all $n \in \mathbb{N}$ it holds that $k' \preceq S^n_{k,f}$.

Proof. Assume $k = (X, L, \longrightarrow, x)$ and $k' = (X', L', \longrightarrow, x')$ and further assume that $R \subseteq X' \times X$ such that $k' \preceq_R k$. It will be shown that $k' \preceq_{R'} S^n_{k,f}$ where R' is defined as:

$$R' = \{(y', (y, g)) \mid (y', y) \in R \wedge (X', L', \longrightarrow', y') \vDash g\}$$

Clearly $(x', (x, f)) \in R'$ and for all $(y', (y, g)) \in R'$ it holds that $L'(y') = L_{\mathrm{XF}}(y, g)$. If $y' \xrightarrow{e}' z'$ then by Lemma 3.4 there exists a step $(y, g) \xrightarrow{e}_0 (z, g')$ such that $(X', L', \longrightarrow', z') \vDash g'$. By partial bisimulation, there exists a step $y \xrightarrow{e} z$ such that $(z', z) \in R$. Subsequently, Lemma 3.7 may be applied, to construct the appropriate step. This requires that for all $m < n$ and for all $v \in \mathcal{U}^*$ such that $(z, g') \xrightarrow{v}{}^*{}_m (w, g'')$ it holds that $(w, g'') \uparrow_m g''$. This follows by application of Lemma 3.8 and Lemma 3.9. It then follows that $(z', (z, g')) \in R'$.

For the right-to-left case, assume $(y, g) \xrightarrow{e}_n (z, g')$ for some $e \in \mathcal{U}$. Since $y \xrightarrow{e} z$, there exists a step $y' \xrightarrow{e}' z'$ such that $(z', z) \in R$. By Lemma 3.6, it holds that $(X', L', \longrightarrow', z') \vDash g'$, and therefore $(z', (z, g')) \in R'$. □

Theorem 3.5 shows that if a solution exists, it will eventually be found by the synthesis construction introduced before.

Theorem 3.5. If $k', k \in \mathcal{K}$ and $f \in \mathcal{F}$ such that $k' \preceq k$ and $k' \vDash f$, then there exists an $n \in \mathbb{N}$ such that $S^n_{k,f}$ is complete.

Proof. Assume that $k = (X, L, \longrightarrow, x)$ and $k' = (X', L', \longrightarrow', x')$. By Theorem 3.1 there exists an $n \in \mathbb{N}$ such that $S^n_{k,f}$ is stable. Due to the construction in Definition 2.14, for all $(x, f) \longrightarrow^*_n (y, f')$ at least one of the following two observations holds:

1. There exist $v, w \in \mathcal{U}^*$ and $s \in \mathcal{C}^*$ such that $(x, f) \xrightarrow{vsw}{}^*{}_n (y, f')$; or
2. There exists $u \in \mathcal{U}^*$ such that $(x, f) \xrightarrow{u}{}^* (y, f')$.

In the first case, by Definition 2.14 it holds that $(y, f') \uparrow_n f'$. For the second case, Lemma 3.8 can be applied to obtain an $y' \in X'$ such that $x' \{\longrightarrow'\}^* y'$ and $(X', L', \longrightarrow', y') \vDash f'$ and $(X', L', \longrightarrow', y') \preceq (X, L, \longrightarrow, y)$. By Lemma 3.9 it then holds that $(y, f') \uparrow_n f'$. It follows that $S^n_{k,f}$ is complete. □

3.2 Algorithm

The algorithm in Figure 3.2 is proposed as a direct implementation of the theoretical synthesis construction introduced in Chapter 2, for which termination and correctness have already been shown in Section 3.1. What remains to be analyzed is the computational complexity of the proposed algorithm, which is detailed in Theorem 3.6. The key parts of this algorithm will be analyzed first.

For the first part of this algorithm, shown in lines 1-11 in Figure 3.2, a somewhat different approach compared to the theoretical setup in Definition 2.12 is applied, since these derivation rules cannot be directly projected onto pseudo-code. Instead of a direct transformation of each original transition $x \xrightarrow{e} x'$ into a combined transition $(x, f) \xrightarrow{e}_0 (x', f')$, a recursive procedure $zero$ is used where finiteness of formula expansion, as shown in Theorem 3.1, is applied in order to obtain a finite set of connected transitions which form $\longrightarrow_0$. This means that once every outgoing transition of a certain state (x, f) has been computed, these transitions, and successive transitions thereof, do not have to be computed again. The test in line 5 in Figure 3.2, determines whether the state (x, f) has been inspected before.

The second part of the algorithm, shown in lines 13-34 in Figure 3.2, applies the synthesizability test repeatedly and removes the transitions to states for which this test fails. This is done until a stable point is reached, as can be observed in line 21 in Figure 3.2. The completeness test in line 30 in Figure 3.2 then determines whether synthesis has been successful.

What follows is a sketch of a proof for the computational complexity for the algorithm shown in Figure 3.2. This requires an appropriate metric for the formula size since the number of transitions in $S^0_{k,f}$ depends upon the size of the formula. Such a metric is given in Definition 3.1, followed by the algorithm and the actual computational complexity proof.

Definition 3.1. The function $size : \mathcal{F} \mapsto \mathbb{N}$ is defined as a metric for the size of the formulas in $\mathcal{F}$. Assume that $f, g \in \mathcal{F}$, $b \in \mathcal{B}$ and $e \in \mathcal{E}$ in the following definition:

$$
\begin{array}{lcl}
size\,(f) & = & 1 \text{ for } f \in \{b, \Diamond b, <e>, dlf\} \\
size\,(f \wedge g) & = & size\,(f) + size\,(g) \\
size\,(b \vee f) & = & 1 + size\,(f) \\
size\,([\,e\,]\,f) & = & 1 + size\,(f) \\
size\,(<e>f) & = & 1 + size\,(f) \\
size\,(\Box\, f) & = & 1 + size\,(f)
\end{array}
$$

```
1  procedure zero(⟶ ⊆ X × ℰ × X, (x, f) ∈ X × ℱ, H ⊆ X × ℱ)
2      returns ⟶₀ ⊆ (X × ℱ) × ℰ × (X × ℱ)
3  begin
4      set ⟶₀ := ∅
5      if (x, f) ∈ H
6          return ∅
7      for each (x, f) -e->₀ (x', f') as in Definition 2.12
8          set ⟶₀ := ⟶₀ ⋃ {((x, f), e, (x', f'))}
9          set ⟶₀ := ⟶₀ ⋃ zero(⟶, (x', f'), H ⋃ {(x, f)})
10     return ⟶₀
11 end
12
13 procedure synthesis(k ∈ 𝒦, f ∈ ℱ)
14     returns S^n_{k,f} ∈ 𝒦 or false
15 begin
16     let k = (X, L, ⟶, x)
17     set ⟶₀ := zero(⟶, (x, f), ∅)
18     for each n = −1 or n > 0
19         set ⟶_n := ∅
20     set n := 0
21     repeat until ⟶_{n−1} = ⟶_n
22         for each (y, g) -e->_n (y', g')
23             if e ∈ 𝒰
24                 set ⟶_{n+1} := ⟶_{n+1} ⋃ {(y, g) -e->_n (y', g')}
25             else if (y'', g'') ↑_n g'' for each v ∈ 𝒰* and
26                 (y', g') -v->*_n (y'', g'') as in Definition 2.13
27                 set ⟶_{n+1} := ⟶_{n+1} ⋃ {(y, g) -e->_n (y', g')}
28         set n := n + 1
29     set S^n_{k,f} := (X × ℱ, L_XF, ⟶_n, (x, f))
30     if (x', f') ↑_n f' for each (x, f) ⟶*_n (x', f')
31         return S^n_{k,f}
32     else
33         return false
34 end
```

Figure 3.2: Algorithm for synthesis of a formula $f \in \mathcal{F}$, applied to the Kripke-LTS $k \in \mathcal{K}$. Note the usage of a distinct procedure `zero` to construct the synthesis starting point $\longrightarrow_0$, since Definition 2.12 cannot be expressed directly in pseudo-code.

Theorem 3.6. The algorithmic generation of $S^n_{k,f}$ terminates in $\mathcal{O}(m^4)$ steps, if $m = size(f) \times l$ and k has l transitions.

Proof. Let $k = (X, L, \longrightarrow, x)$ and $f \in \mathcal{F}$. Assume that l is the number of transitions in $\longrightarrow$. The starting point of synthesis $\longrightarrow_0$, as generated by the procedure `zero`, has $\mathcal{O}(size(f) \times l)$ transitions, since it is expanded by a factor which depends upon the size of f. The procedure `zero` is invoked only once within the procedure `synthesis`, and returns the transition relation $\longrightarrow_0$, having $\mathcal{O}(m)$ transitions. Since the succeeding iteration in the `synthesis` procedure operates multiple times upon this transition relation, it can be safely assumed that this part of the procedure outgrows other parts in the computational complexity. A distinction is made between four nested operations which may each take $\mathcal{O}(m)$ steps:

1. The outer loop starting in line 21, which runs until a stable transition relation has been reached, which may involve as many iterations as there are transitions in $\longrightarrow_0$.

2. The inner loop starting in line 22, which considers for each transition whether it should be removed.

3. The applied synthesizability test in lines 25-26 can be subdivided into two nested parts:

   (a) Synthesizability is evaluated at every state reachable by uncontrollable events, which may involve a search over $\mathcal{O}(m)$ transitions.

   (b) The synthesizability test itself has complexity $\mathcal{O}(m)$ as shown in Definition 2.13, due to formulas of type $\Diamond b$.

This leads to the observation that the two nested loops in the `synthesis` procedure give rise to a computational complexity in the order of $\mathcal{O}(m^4)$. The succeeding invocation of the completeness test only computes the synthesizability for every remaining reachable state, and does not remove any more transitions. Its complexity is therefore superseded by the aforementioned two nested loops. □

The algorithm in Figure 3.2 is presented as a direct implementation of the synthesis construction as presented in this thesis and not as the most efficient or optimized implementation, since this would obscure the insight into such an algorithm. Nevertheless, the second case study in Section 3.6 analyzes the scalability of the algorithm presented in Figure 3.2.

3.3 Ramadge-Wonham Supervisory Control

In this section it is explained how a Ramadge-Wonham (RW) control synthesis problem [76] may be expressed using the theory proposed in this thesis. Among other adaptations, a relation between the language-based constructs in RW-synthesis and the behavioral preorder of partial bisimilarity needs to be established. RW-synthesis is limited to deterministic models of both plant and supervisor [76] and this restriction is applied in this section as well.

Recall from Section 2.1 that RW-synthesis explicitly stipulates a subset $X_m \subseteq X$ of states as being *marked*, modeling completed or notified tasks in the physical process the plant represents [76]. To cope with marked states, the plant model needs to be adapted as stated in Definition 3.2.

Definition 3.2. For plant model $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ and set $X_m \subseteq X$ of *marked* states a new label $marked$ will be added such that $marked \in L(y)$ for each $y \in X_m$, and $marked \notin L(y)$ for each $y \notin X_m$.

For the remainder of this section it is assumed that each $k \in \mathcal{K}$ is adapted as described in Definition 3.2. Two language-based notions are now introduced in Definition 3.3 and language-based controllability in Definition 3.4. These are definitions that are adapted from Section 2.1 for $\mathcal{K}$-structures.

Definition 3.3. The language $\mathcal{L}(k)$ and marked language $\mathcal{L}_m(k)$ of a Kripke-LTS $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ are defined as follows:

$$\begin{aligned} \mathcal{L}(k) &= \{s \in \mathcal{E}^* \mid \exists x' \in X : x \xrightarrow{s}{}^* x'\} \\ \mathcal{L}_m(k) &= \{s \in \mathcal{E}^* \mid \exists x' \in X : x \xrightarrow{s}{}^* x' \wedge marked \in L(x')\} \end{aligned}$$

In addition the language closure $\overline{L}$ of $L \subseteq \mathcal{E}^*$ is defined as:

$$\overline{L} = \{s \in L \mid \exists t \in \mathcal{E}^* : st \in L\}.$$

Definition 3.4. For languages $L \subseteq \mathcal{E}^*$ and $K \subseteq \mathcal{E}^*$ it is defined that K is *controllable* with regard to L if for each $s \in \overline{K}$ and $su \in \overline{L}$, for $u \in \mathcal{U}$, it holds that $su \in \overline{K}$.

Parallel composition is given in Definition 3.5. This type of parallel composition is borrowed from process theory [8] and may be used to define the construction between plant and supervisor in supervisory control. Definition 3.5 is essentially an adaptation of Definition 2.3 which accommodates state labels and handles marked states via these aforementioned labels.

Definition 3.5. For $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ and $k' = (X', L', \longrightarrow', x') \in \mathcal{K}$, the parallel composition $k' \parallel k$ is defined as follows:

$$k' \parallel k = (X' \times X, L'', \longrightarrow'', (x', x))$$

where $L''(y', y) = L'(y') \cap L(y)$, for each $y' \in X'$ and $y \in X$, and $(y', y) \xrightarrow{e} {}''(z', z)$ if and only if $y' \xrightarrow{e} {}'z'$ and $y \xrightarrow{e} z$. Clearly $\mathcal{L}(k' \parallel k) \subseteq \mathcal{L}(k)$ and $\mathcal{L}(k' \parallel k) \subseteq \mathcal{L}(k')$.

To express an RW-synthesis problem, a deterministic plant model $p \in \mathcal{K}$ is assumed. Subsequently, a supervisor $s \in \mathcal{K}$ needs to be constructed such that the following properties hold: 1) $\mathcal{L}(p \parallel s)$ is controllable with regard to $\mathcal{L}(p)$; and 2) $p \parallel s$ is non-blocking; that is: $\overline{\mathcal{L}(p \parallel s) \cap \mathcal{L}_m(p)} = \mathcal{L}(p \parallel s)$. In Theorem 3.7 it is shown that if $S^1_{p,\Box \Diamond marked}$ is chosen for s these two conditions are satisfied. Observe that also $S^1_{p,\Box \Diamond marked}$ is deterministic, since every step $x \xrightarrow{e} x'$ gives rise to at most one step which is of the form $(x, \Box \Diamond marked) \xrightarrow{e}_1 (x', \Box \Diamond marked)$, due to normalization, Definition 2.12 and Definition 2.14.

Lemma 3.10. If $k', k \in \mathcal{K}$ such that $k' \preceq k$, then $\mathcal{L}(k')$ is controllable with regard to $\mathcal{L}(k)$.

Proof. Let $k' = (X', L', \longrightarrow', x')$ and $k = (X, L, \longrightarrow, x)$ and assume $R \subseteq X' \times X$ exists such that $k' \preceq_R k$. Definition 3.4 may now be followed. Assume that $s \in \overline{\mathcal{L}(k')}$ and $su \in \overline{\mathcal{L}(k)}$, for some $u \in \mathcal{U}$. Then clearly there exists $y' \in X'$ such that $x' \{\xrightarrow{s}'\}^* y'$ and $y \in X$ such that $x \xrightarrow{s}{}^* y$ and $(y', y) \in R$, by Definition 2.6. Since $y \xrightarrow{u} z$ exists, again by Definition 2.6 there exists a step $y' \xrightarrow{u}{}' z'$ and thus $su \in \overline{\mathcal{L}(k')}$. This result is a direct consequence of the assumption that both k' and k are deterministic. □

For the remainder of this section, only those plant models $p \in \mathcal{K}$ for which a non-empty solution exists are considered.

Lemma 3.11. For each $k = (X, L, \longrightarrow, x) \in \mathcal{K}$ it holds that $S^1_{k,\Box \Diamond marked}$ is complete.

Proof. For each $(x, \Box \Diamond marked) \longrightarrow_0^* (y, \Box \Diamond marked)$ and $(y, \Box \Diamond marked) \xrightarrow{e}_0 (z, \Box \Diamond marked)$ it holds that $(y, \Box \Diamond marked) \xrightarrow{e}_1 (z, \Box \Diamond marked)$ if and only if $(z', \Box \Diamond marked) \uparrow_0 \Diamond marked$, for all $u \in \mathcal{U}^*$ and $(z, \Box \Diamond marked) \xrightarrow{u}_0^* (z', \Box \Diamond marked)$, as follows from Definition 2.14. Due to the assumption that for each $u \in \mathcal{U}^*$ and $(x, \Box \Diamond marked) \xrightarrow{u}_0^* (y, \Box \Diamond marked)$ it holds that $(y, \Box \Diamond marked) \uparrow_0 \Diamond marked$, it may be concluded that $S^1_{k,\Box \Diamond marked}$ is complete. □

Theorem 3.7. For each $p \in \mathcal{K}$ it holds that $\mathcal{L}\,(p \parallel S^1_{p,\Box\,\Diamond\,marked})$ is controllable with regard to $\mathcal{L}\,(p)$, and in addition $p \parallel S^1_{p,\Box\,\Diamond\,marked}$ is non-blocking.

Proof. By Lemma 3.11, $S^1_{p,\Box\,\Diamond\,marked}$ is complete. Then by Lemma 3.10 and Theorem 3.3 it holds that $\mathcal{L}\,(p \parallel S^1_{p,\Box\,\Diamond\,marked})$ is controllable with regard to $\mathcal{L}\,(p)$. Now non-blockingness needs to be considered. Observe that:

$$\overline{\mathcal{L}\,(p \parallel S^1_{p,\Box\,\Diamond\,marked}) \cap \mathcal{L}_m(p)} \subseteq \mathcal{L}\,(p \parallel S^1_{p,\Box\,\Diamond\,marked})$$

already holds, so therefore:

$$\mathcal{L}\,(p \parallel S^1_{p,\Box\,\Diamond\,marked}) \subseteq \overline{\mathcal{L}\,(p \parallel S^1_{p,\Box\,\Diamond\,marked}) \cap \mathcal{L}_m(p)}$$

is what remains to be shown. Assume that $p = (X, L, \longrightarrow, x)$ and $t \in \mathcal{L}\,(p \parallel S^1_{p,\Box\,\Diamond\,marked})$, then $(x, \Box\,\Diamond\,marked) \xrightarrow{t}{}^*_1 (y, \Box\,\Diamond\,marked)$. Since $(y, \Box\,\Diamond\,marked) \uparrow_0 \Diamond\,marked$ there exists $v \in \mathcal{E}^*$ such that $(y, \Box\,\Diamond\,marked) \xrightarrow{v}{}^*_1 (z, \Box\,\Diamond\,marked)$ and $marked \in \mathcal{L}\,(z)$. Then by Definition 2.12 it holds that $x \xrightarrow{tv}{}^* z$, and therefore $tv \in \mathcal{L}_m(p)$. □

3.4 Computer Verified Proofs

In this section computer verified proofs are detailed which have been used to support the main synthesis construction as introduced in the previous chapters. The complete file containing all computer verified proofs is available online at the following location:

`https://github.com/ahulst/deds`

In conjunction with explaining the computer verified proofs in detail, information about the specific Coq constructions which are applied will be provided. The reader is only expected to have initial knowledge regarding computer verified proofs or the syntax of specific Coq constructions. Therefore, it is the intention to provide detailed explanations in this chapter. Useful resources for more in-depth knowledge regarding proof assistants and Coq may be found in [13] and [39]. The quick hands-on guide Coq in a Hurry[4] written by Yves Bertot may also be very useful in acquiring or updating knowledge regarding the Coq proof assistant.

An important requirement for each computer verified proof (and for each formal model, for that matter) is that it should be as close as possible to the

[4] Available at: `https://cel.archives-ouvertes.fr/inria-00001173v5/document`

actual subject it is intended to model. In the case of formal proofs, this comes down to encoding the applied constructions in such a way that they closely mimic their counterparts in the mathematical content of the proof. For parts of the correctness construction in the previous chapter, this was found to be not possible. It would certainly have been possible to encode proofs of the entire correctness construction in Coq. However, such an encoding would involve different formalizations and close resemblance between the mathematical proofs, and the formal encoding would be lost. Therefore, a particular formalization was chosen which stays closer to the mathematical content in the previous chapter but does not allow the encoding of the proofs for termination and solution existence. This is explained in more detail when the relevant parts of the proof are considered.

An important note relevant for this proof regards the specifics for decidability as materialized in Coq. The first line of the proof specifies the following: `Axiom classic : forall (P : Prop), P \/ ~P.` This indicates that for each proposition a distinction may be made between the case where it is true and the case where it is false; the usual interpretation of classical logic. Since Coq is based on constructive logic, this axiom needs to be specified explicitly. It would have been possible to write the formalized proof in a constructive way in its entirety, since classical logic is only applied when considering labels and state-based properties. However, then each labeling function would have to be explicitly specified as being computable and each Kripke-LTS would have to be extended in such a way that equality on states is decidable. This would further obfuscate the proof and for reasons mentioned earlier, regarding resemblance between formalized and original proof, this solution was not chosen.

Starting point of the actual proof consists of the enumeration of three parameters to the theory by means of the Coq code below. The first line specifies that the theory is based on the assumption of the existence of two sets `E` and `P` for events and propositions respectively. In addition, the characteristic function for the subset of uncontrollable events is specified as another parameter of the theory.

```
Parameter E P : Set.
Parameter U : E -> Prop.
```

The next step is to specify the Kripke-LTS by means of an inductive structure. Several options exist to encode this construct in Coq. The most straightforward option would be to stay close to the mathematical definition and use a four-tuple $(X, L, \longrightarrow, x)$. However, the type of the three latter parameters L, $\longrightarrow$ and x depends upon the type X, and hence it is not possible to encode

$\mathcal{K}$ directly as a four-tuple. Therefore, $\mathcal{K}$ is defined as an inductive type, as shown in the Coq listing below, where `klts` is a *constructor* which is used to construct a new instance of the specified type.

```
Inductive K : Type := klts : forall (X : Set),
  (X -> P -> Prop) -> (X -> E -> X -> Prop) -> X -> K.
```

This definition is followed directly by another inductive definition where the reflexive-transitive closure of a transition relation is defined. Note that the keyword `Inductive` is applied here to define an inductive predicate, having the two parameters X and $\longrightarrow$. Since such inductive predicates will be applied later on in the proofs as well, they need to be considered in somewhat more detail. First note that Coq makes a distinction between recursive definitions and inductive definitions, as illustrated as follows. In Coq, a recursive function is a total function parameterized by an inductive type which may only invoke itself if applied using a smaller case in the construction of this parameter. For example, by following the standard constructive definition of natural numbers using zero and successor, a recursive function f which is called using parameter $n+1$ is only well-defined in Coq if f is applied to n or 0. On the contrary, an inductive predicate specifies only the cases for which the predicate is true, and thus the definition of an inductive predicate in Coq need not be total. From a mathematical point of view, inductive predicates are therefore better suited to capture derivation rules, compared to recursive functions, since derivation rules are often only applied to define the positive cases of a predicate. The restrictions applied to inductive predicates in Coq are more strict: a function is not allowed to be applied to an invocation of the predicate itself during its definition. This restriction is generally referred to as *strict positivity*. Derivation rules, for instance when they are applied to define an operational semantics, offer a less strict regime and are often considered to be well-defined under some form of stratification.[5] Now, the reflexive-transitive closure of a transition relation by means of two derivation rules is considered first, which may then be readily compared to the Coq-based definition below.

$$\frac{}{x \longrightarrow^* x} \qquad \frac{x \xrightarrow{e} y \quad y \longrightarrow^* x'}{x \longrightarrow^* x'}$$

Now compare these two derivation rules to their two respective counterparts within the inductive predicate defined below. This definition relies upon two

[5] At present there exists no Coq library which provides a generalized implementation of stratification from a derivation rule perspective. This would be a very interesting future project.

constructors which each correspond to one of the derivation rules. These two constructors define precisely under which conditions the predicate which is built by these derivation rules holds. Due to the fact that only the cases for which this predicate holds are stated, this definition is a typical example of a non-total definition:

```
Inductive trans (X : Set) (step : X -> E -> X -> Prop) :
    X -> X -> Prop :=
  | trans_refl : forall (x : X), trans X step x x
  | trans_closed : forall (x y x' : X) (e : E),
    step x e y -> trans X step y x' -> trans X step x x'.
```

Partial bisimilarity can now be defined in Coq, as shown in the listing below. The relation R is defined as a two-parameter characteristic function, while a more intuitive definition would have been to use a characteristic function on pairs. However, the latter option results in less clarity in proofs, since these pairs need to be broken down in their parts all the time if one part of a pair is considered in a proof. The definition of partial bisimilarity in Coq, as shown below, is further considered to be self-explanatory.

```
Definition pbis (k' k : K) : Prop :=
  match (k', k) with
  | (klts X' L' step' x', klts X L step x) =>
      exists R : X' -> X -> Prop, R x' x /\
        forall (y' : X') (y : X), R y' y ->
          (forall (p : P), L' y' p <-> L y p) /\
          (forall (e : E) (z' : X'), step' y' e z' ->
            exists z : X, step y e z /\ R z' z) /\
          (forall (e : E) (z : X), U e -> step y e z ->
            exists z' : X', step' y' e z' /\ R z' z)
  end.
```

The treatment of the initial definitions is continued by specifying the two sets of formulas $\mathcal{B}$ and $\mathcal{F}$ as inductive types. The translation into Coq code of Definition 2.8 is shown below. Such inductive definitions of sets quite closely mimic their corresponding definitions as grammars in BNF form. However, Definition 2.8 only defines formulas $<e>f$ for $e \in \mathcal{C}$. Since controllability is defined by a characteristic function on events, and due to the fact that propositional tests are not allowed in the construction of inductive types, an additional well-formedness test for formulas in $\mathcal{F}$ needs to be specified. This is done by defining the recursive predicate `wf` as shown in the Coq listing below. This predicate is only used as a premise in lemmas and theorems when

the restriction that $e \in \mathcal{C}$ in a formula $<e>f$ is actually relevant. When encoding a fixpoint definition in Coq syntax, the placeholder notation (_) is applied to match all cases not covered previously.

```
Inductive F : Set :=
  | basic : B -> F
  | conj : F -> F -> F
  | disj : B -> F -> F
  | all : E -> F -> F
  | ex : E -> F -> F
  | box : F -> F
  | diam : B -> F
  | can : E -> F
  | dlf : F.

Fixpoint wf (f : F) : Prop :=
  match f with
  | conj f g => wf f /\ wf g
  | disj b f => wf f
  | all e f => wf f
  | ex e f => ~U e /\ wf f
  | box f => wf f
  | _ => True
  end.
```

The next step is to define validity of formulas in $\mathcal{B}$ and $\mathcal{F}$ with regard to a Kripke-LTS in Coq. An implementation of $\mathcal{F}$-validity in Coq is shown below:

```
Fixpoint val (k : K) (f : F) : Prop :=
  match (k, f) with
  | (klts X L step x, basic b) => bval X L x b
  | (_, conj f g) => val k f /\ val k g
  | (klts X L step x, disj b f) => bval X L x b \/ val k f
  | (klts X L step x, all e f) => forall (x' : X),
    step x e x' -> val (klts X L step x') f
  | (klts X L step x, ex e f) => exists x' : X,
    step x e x' /\ val (klts X L step x') f
  | (klts X L step x, box f) => forall (x' : X),
    trans X step x x' -> val (klts X L step x') f
```

⋮

(some lines of Coq code omitted)

⋮

The next step is to define the reduction relation on modal expressions in Coq. For reasons mentioned earlier, an inductive predicate is applied to encode the corresponding derivation rules. This definition directly follows the derivation rules in Definition 2.12.

```
Inductive red (X : Set) (L : X -> P -> Prop) (x : X) :
    F -> E -> F -> Prop :=
  | red_basic : forall (b : B) (e : E),
    red X L x (basic b) e (basic true)
  | red_and : forall (f g f' g' : F) (e : E),
    red X L x f e f' -> red X L x g e g' ->
    red X L x (conj f g) e (conj f' g')
  | red_or_left : forall (f : F) (b : B) (e : E),
    bval X L x b -> red X L x (disj b f) e (basic true)
  | red_or_right : forall (f f' : F) (b : B) (e : E),
    ~ bval X L x b -> red X L x f e f' ->
    red X L x (disj b f) e f'
  | red_all_pos : forall (f : F) (e : E),
    red X L x (all e f) e f
  | red_all_neg : forall (f : F) (e e' : E), e <> e' ->
    red X L x (all e f) e' (basic true)
  | red_ex_pos : forall (f : F) (e : E),
    red X L x (ex e f) e f
  | red_ex_neg : forall (f : F) (e e' : E),
    red X L x (ex e f) e' (basic true)
  | red_box : forall (f f' : F) (e : E),
    red X L x f e f' ->
    red X L x (box f) e (conj (box f) f')
  | red_diam : forall (b : B) (e : E),
    red X L x (diam b) e (basic true)
  | red_can : forall (e e' : E),
    red X L x (can e) e' (basic true)
  | red_dlf : forall (e : E),
    red X L x dlf e (basic true).
```

Next comes the definition of sub-formulas and the definition of the synthesizability predicate in Coq. Sub-formulas as shown in Definition 2.11 are an example of a recursive predicate which may be encoded recursively as well as inductively in Coq (the latter option was used due to similarity with its definition via derivation rules). For instance, the reflexive-transitive closure of a transition relation is far simpler to encode inductively. Since a recursive definition depends upon the presence of a parameter of an inductive type, an extra parameter would have to be added. For instance, the application depth

as a natural number. On the other hand, a function such as `bval` is, as mentioned earlier, impossible to encode inductively, since it negates the result of a recursive invocation. Since negation is a function in Coq, this would not be allowed under the strict positivity restriction. Sub-formulas are encoded in Coq as follows:

```
Inductive sub (X : Set) (L : X -> P -> Prop) (x : X) :
    F -> F -> Prop :=
  | sub_refl : forall (f : F), sub X L x f f
  | sub_and_left : forall (f g h : F), sub X L x f g ->
    sub X L x f (conj g h)
  | sub_and_right : forall (f g h : F), sub X L x f h ->
    sub X L x f (conj g h)
  | sub_or : forall (f g : F) (b : B), ~ bval X L x b ->
    sub X L x f g -> sub X L x f (disj b g)
  | sub_box : forall (f g : F), sub X L x f g ->
    sub X L x f (box g).
```

The next definition to be detailed is that of synthesizability, which is shown in the Coq listing below. This formalized definition is somewhat complicated and therefore requires some explanation. Again, an inductive predicate was applied to achieve close resemblance to the derivation rules in Definition 2.13. This has as an additional advantage that no existential variables need to be explicitly specified using existential quantifiers, which may lead to more complicated proofs. Due to the fact that the transformation into Coq code of Definition 2.13 results in a quite lengthy code fragment, the listing below is split into two parts:

```
Inductive syn (X : Set) (L : X -> P -> Prop)
    (step : X * F -> E -> X * F -> Prop) :
    X * F -> F -> Prop :=
  | syn_basic : forall (x : X) (b : B) (g : F),
    bval X L x b -> syn X L step (x, g) (basic b)
  | syn_and : forall (x : X) (f1 f2 g : F),
    syn X L step (x, g) f1 -> syn X L step (x, g) f2 ->
    syn X L step (x, g) (conj f1 f2)
  | syn_or_left : forall (x : X) (b : B) (f g : F),
    bval X L x b -> syn X L step (x, g) (disj b f)
  | syn_or_right : forall (x : X) (b : B) (f g : F),
    syn X L step (x, g) f ->
    syn X L step (x, g) (disj b f)
  | syn_all : forall (x : X) (f g : F) (e : E),
    syn X L step (x, g) (all e f)
```

The definition of synthesizability in Coq continues below. Note the cases for $\Box f$ which is recursively defined in a relatively simple way, and the case for $\Diamond b$, for $b \in \mathcal{B}$, which tests for the existence of a reachable state-formula pair where b holds.

```
  | syn_ex : forall (x x' : X) (f g g' : F) (e : E),
    step (x, g) e (x', g') -> sub X L x' f g' ->
    syn X L step (x', g') f ->
    syn X L step (x, g) (ex e f)
  | syn_box : forall (x : X) (f g : F),
    syn X L step (x, g) f ->
    syn X L step (x, g) (box f)
  | syn_diam : forall (x x' : X) (b : B) (g g' : F),
    trans (X * F) step (x, g) (x', g') ->
    bval X L x' b -> syn X L step (x, g) (diam b)
  | syn_can : forall (x x' : X) (g g' : F) (e : E),
    step (x, g) e (x', g') -> syn X L step (x, g) (can e)
  | syn_dlf : forall (x x' : X) (g g' : F) (e : E),
    step (x, g) e (x', g') -> syn X L step (x, g) dlf.
```

The treatment of the synthesis construction is continued by detailing how the main parts are set up, as detailed in the Coq listing below. A helper predicate `unctrl` is defined first, which may be used to restrict a transition relation to uncontrollable steps. The succeeding steps in the process of transition removal are then formalized using the fixpoint `rel`. An ambiguity in this formalization needs to be clarified. The expression `S n` is used to refer to the successor of `n` and not to the actual synthesis construction, which is defined thereafter.

```
Definition unctrl (X : Set) (step : X -> E -> X -> Prop)
  (x : X) (e : E) (x' : X) : Prop := step x e x' /\ U e.

Fixpoint rel (n : nat) (X : Set) (L : X -> P -> Prop)
  (step : X -> E -> X -> Prop) (xf : X * F) (e : E)
  (xf' : X * F) : Prop :=
  match n with
  | 0 => step (fst xf) e (fst xf') /\
         red X L (fst xf) (snd xf) e (snd xf')
  | S n => rel n X L step xf e xf' /\ (U e \/
    forall (xf'' : X * F), trans (X * F)
      (unctrl (X * F) (rel n X L step)) xf' xf'' ->
    syn X L (rel n X L step) xf'' (snd xf''))
  end.
```

The synthesis construction itself is then defined in terms of the projection of the labeling function onto the state-formula product space and the fixpoint `rel` defined one step earlier. This definition thereby overloads the build-in Coq function of successor.

```
Definition S (k : K) (f : F) (n : nat) : K :=
  match k with
  | klts X L step x =>
    klts (X * F) (fun xf => L (fst xf))
      (rel n X L step) (x, f)
  end.
```

One remaining definition in Coq is that of completeness of the synthesis construction, which signifies that no more transition removal is required. This may be straightforwardly encoded in Coq using the following definition:

```
Definition complete (n : nat) (X : Set)
  (L : X -> P -> Prop) (step : X -> E -> X -> Prop)
    (x : X) (f : F) : Prop :=
      forall (x' : X) (f' : F), trans (X * F)
        (rel n X L step) (x, f) (x', f') ->
          syn X L (rel n X L step) (x', f') f'.
```

Since definitions are now encoded in Coq, the next phase of formalization is the encoding of the actual proofs. A number of smaller lemmas are first required in order to prove the major theorems. A relatively simple lemma is discussed in somewhat more detail to consider the basic foundations of how Coq proofs are built, by illustrating how the proof for $k \vDash b \iff S^n_{k,f} \vDash b$ is set up. This is detailed in the Coq listing below. The function `bval` does not depend upon the transition relation, as mentioned earlier, and is therefore only defined in terms of the parameters of the state space, labeling functions and initial state. When studying this proof, one should note that the application of a single tactic (a step in the proof) may result in multiple other goals having to be proven. One particularly important remark is that if a tactic is closed by a semi-colon, all consequential proof statements are subjected to the tactics after the semi-colon. The proof steps in `bval_expand` are now considered in somewhat more detail. First, the **`intros`** tactic is applied to assume a number of variables. Then the **`induction`** tactic is applied to signify induction towards the structure of a variable $b \in \mathcal{B}$. This tactic is then followed by the application of **`split`**, which splits a conjunction into a proof obligation for its two conjuncts. Application of this tactic is required here since Coq encodes an if-and-only-if statement `P <-> Q` for propositions `P`

and Q as (P -> Q) /\ (Q -> P). This also explains the **intro** H tactic after the application of **split**. The next step is to apply simplification in the proof obligation and all hypotheses by **simpl in** *, which applies β and ι reductions [13]. All proof obligations are then resolved by application of the **tauto** tactic. This tactic resolves first-order propositional expressions, which are clearly decidable.

```
Lemma bval_expand : forall (X : Set) (L : X -> P -> Prop)
  (x : X) (g : F) (b : B), bval X L x b <->
  bval (X * F) (fun xf => L (fst xf)) (x, g) b.
Proof.
  intros X L x g b ; induction b ; split ; intro H ;
    simpl in * ; tauto.
Qed.
```

Before the Coq encoding of the major proofs is considered, one more lemma has to be considered in somewhat more detail, since it applies a variant of the induction applied in many other proofs. Therefore, the following lemma will be discussed: if $f \in sub\,(x, g)$ and $(X, L, \longrightarrow, x) \vDash g$, then $(X, L, \longrightarrow, x) \vDash f$. One option might be to apply induction towards the structure of f. However, there exists a more compact strategy to resolve this proof. Therefore, sub X L x f g will be introduced as the premise Hsub, followed by applying the tactic **induction** Hsub. This results in Coq applying induction towards the construction of sub X L x f g. In essence, this is equivalent to induction towards the derivation depth of $f \in sub\,(x, g)$ and thereafter a case distinction between the various cases for $f \in sub\,(x, g)$. Application of the induction hypothesis is performed automatically in most cases by means of the **auto** tactic, except for the case for sub-formulas of invariant formulas.

```
Lemma sub_val : forall (f g : F) (X : Set)
  (L : X -> P -> Prop) (step : X -> E -> X -> Prop)
  (x : X), sub X L x f g -> val (klts X L step x) g ->
  val (klts X L step x) f.
Proof.
  intros f g X L step x Hsub Hval ; induction Hsub ; auto.
  simpl in Hval ; destruct Hval ; auto.
  simpl in Hval ; destruct Hval ; auto.
  simpl in Hval ; destruct Hval ; contradiction || auto.
  apply IHHsub ; apply Hval ; apply trans_refl.
Qed.
```

Now the first theorem is considered where it needs to be shown that the synthesis result satisfies the synthesized formula, as shown in the Coq list-

ing below. The proof of this theorem is not contained in its entirety due to its vast length. However, at the start of this proof a number of interesting steps are applied which require some more explanation. After introduction of the generalized variables X to n, it can be easily proven that $f \in sub\,(x, f)$, which holds trivially. Then, this fact is again introduced as a premise for the proof obligation using the **revert** tactic, followed by application of the same **revert** tactic for the initial state x. Then f is renamed into g in the proof obligation, but only at four specific instances. This allows induction towards the structure of f, while at the same time generalizing over the variable g, at the appropriate positions. The included code shows the validity-proof up until the inductive case for the $[e]f$ operator.

```
Theorem validity : forall (X : Set) (L : X -> P -> Prop)
  (step : X -> E -> X -> Prop) (x : X) (f : F) (n : nat),
  complete n X L step x f ->
  val (S (klts X L step x) f n) f.
Proof.
  intros X L step x f n ;
  assert (sub X L x f f) as Hsub by apply sub_refl.
  revert Hsub ; revert x ; generalize f at 2 3 4 as g.
  induction f ; intros g x Hsub Hcomplete.
  apply complete_impl_syn in Hcomplete.
  apply sub_syn with (f := basic b) in Hcomplete ; auto.
  inversion Hcomplete ; simpl ; apply bval_expand ; auto.
  apply sub_and in Hsub ; destruct Hsub ; split ; auto.
  destruct (classic (bval X L x b)) as [ H | H ].
  simpl ; left ; apply bval_expand ; auto.
  simpl ; right ; apply sub_not_or in Hsub ;
    apply IHf || auto ; auto.
  simpl ; intros xg' Hstep ; destruct xg' as [ x' g' ] ;
    apply IHf ; auto.
  apply rel_impl_step_red in Hstep ;
    destruct Hstep as [ _ Hred ].
  apply sub_all with (x' := x') (g' := g') (f := f) in Hred ;
    auto.
          ⋮
  (some lines of Coq code omitted)
          ⋮
Qed.
```

Two lemmas are then required for the controllability proof, which itself follows straightforwardly from these lemmas. The witness relation for the partial bisimulation needs to be specified as a predicate to express which states are contained in the set R, by means of the statement `fun yg y => fst yg = y`. Note that the `fst` function extracts the first element of a pair.

```
Theorem controllability : forall (X : Set)
  (L : X -> P -> Prop) (step : X -> E -> X -> Prop)
  (x : X) (f : F) (n : nat),
  pbis (S (klts X L step x) f n) (klts X L step x).
Proof.
  intros X L step x f n.
  exists (fun yg y => fst yg = y) ; split ; auto.
  intros yg y' H ; destruct yg as [ y g ] ; simpl in H.
  rewrite <- H in * ; split ;
    [ simpl ; tauto | split ] ; clear H y'.
  intros e yg' Hrel ; destruct yg' as [ y' g' ].
  apply rel_impl_step_red in Hrel ;
    destruct Hrel as [ Hstep _ ].
  exists y' ; repeat split ; simpl ; auto.
  intros e y' HU Hstep ;
    destruct (red_ex X L y g e) as [ g' Hred ].
  exists (y', g') ; repeat split ; auto.
  apply unctrl_ex ; auto.
Qed.
```

The next step is to prove a number of lemmas which are required to complete the maximality proofs. It is assumed that the reader is able to understand the applied Coq constructs of these lemmas, given the previous explanations. The key lemma which is required for the maximality proof is listed in the Coq code shown below. Nested induction is applied as well as generalization over the other variables. Another key element is the application of the `solve` tactic, which is combined with the proof-theoretic operand `||`. Once a tactic on the left-hand side does not succeed, the tactic on the right-hand side is applied. Also, note the important difference between the proof-theoretic constructs `||` and `|`. While the first one is applied to try several tactics until one succeeds, the second one is used to apply different tactics to different proof obligations. Another important element of the Coq listing below also needs to be taken into consideration. First, the natural number `n` is assumed via the `intro n` tactic. Thereafter, induction is applied towards `n`. Since the introduction of other variables is postponed to after the `intro` tactic, induction automatically generalizes over all other variables. The same strategy is applied for the vari-

able f. Induction towards the structure of f does not generalize over n, but does in fact generalize over all subsequently introduced variables.

```
Lemma pbis_impl_syn : forall (n : nat) (f g : F)
  (X : Set) (L : X -> P -> Prop)
  (step : X -> E -> X -> Prop) (x : X)
  (k' : K), sub X L x f g -> wf g ->
  pbis k' (klts X L step x) -> val k' g ->
  syn X L (rel n X L step) (x, g) f.
Proof.
  intro n ; induction n using strong_ind ; intro f ;
    induction f ;
    intros g X L step x k' Hsub Hwf Hpbis Hval ;
    solve [ apply syn_all ] ||
    solve [ apply sub_and in Hsub ;
    destruct Hsub as [ Hf1 Hf2 ] ;
    apply syn_and ; [ apply IHf1 with k' |
    apply IHf2 with k' ] ; auto ] || auto.
                    ⋮
  (some lines of Coq code omitted)
                    ⋮
Qed.
```

The remaining theorem concerns the maximality proof, which requires the wf premise. The previous lemma is the most complex one, and forms essentially the major part of the construction used to establish maximal permissiveness. The contents of the maximality theorem follows a similar construction as the controllability theorem, where the witness relation is specified using a characteristic function.

```
Theorem maximality : forall (k' k : K) (f : F) (n : nat),
  wf f -> pbis k' k -> val k' f -> pbis k' (S k f n).
Proof.
  intros k' k f n Hwf Hpbis Hval.
                    ⋮
  (some lines of Coq code omitted)
                    ⋮
Qed.
```

3.5 Case Study

Controlled system synthesis as achieved by the proposed theories in this thesis can be employed in an actual application setting, as detailed in Chapter 3 and Chapter 5, where the coordination of maintenance procedures in the printing process of a high-end industrial printer is modeled [63]. Since the same case study is used in two different instances in this thesis, this case is introduced in general terms in this section. An implementation of the synthesis algorithm was constructed using the C programming language. This implementation may be found at the following location:

`https://github.com/ahulst/deds`

Several distributed independent components make up the printing process of an industrial printer, as shown in Figure 3.3. The main task of the printing process is to apply the toner image onto the toner transfuse belt, followed by the actual printing task where it is fused onto the paper sheet. Preserving printing quality over numerous print jobs involves several maintenance operations. For instance, the toner transfuse belt is jittered periodically to ensure an even spread of the wear induced by sharp paper edges, occasional printing of completely black pages takes place to remove paper dust, and various techniques are applied to remove coarse toner particles. Maintenance scheduling is mainly based upon the number of prints which have taken place, and maintenance scheduling is therefore related to various strict and postponable thresholds. If a maintenance operation has to be performed, the printing process has to switch its power mode from **Run** to **Standby**. However, such a power mode switch may actually trigger the activation of other queued maintenance operations, which may lead to users of the printer having to wait unnecessarily long before the printer becomes usable again.

In Figure 3.3g, the occurrence of a non-delayable maintenance operation **A** suspends the current print task, followed by a power mode transition to **Standby**. However, the power mode change triggers the execution of another maintenance operation **B**, having a longer duration than operation **A**. A realistic example of a practical situation where this occurs is when a black image is printed (**A**), taking the exact time required to print a single page, while the significantly longer transfuse belt jittering (**B**) is initiated due to the power mode switch. This combined behavior gives rise to a prolonged user wait time between print jobs, as shown in Figure 3.3g.

In this case, the control objective is to enforce this uncontrolled system to behave in such a way that undesired emergent behaviors does not occur. In

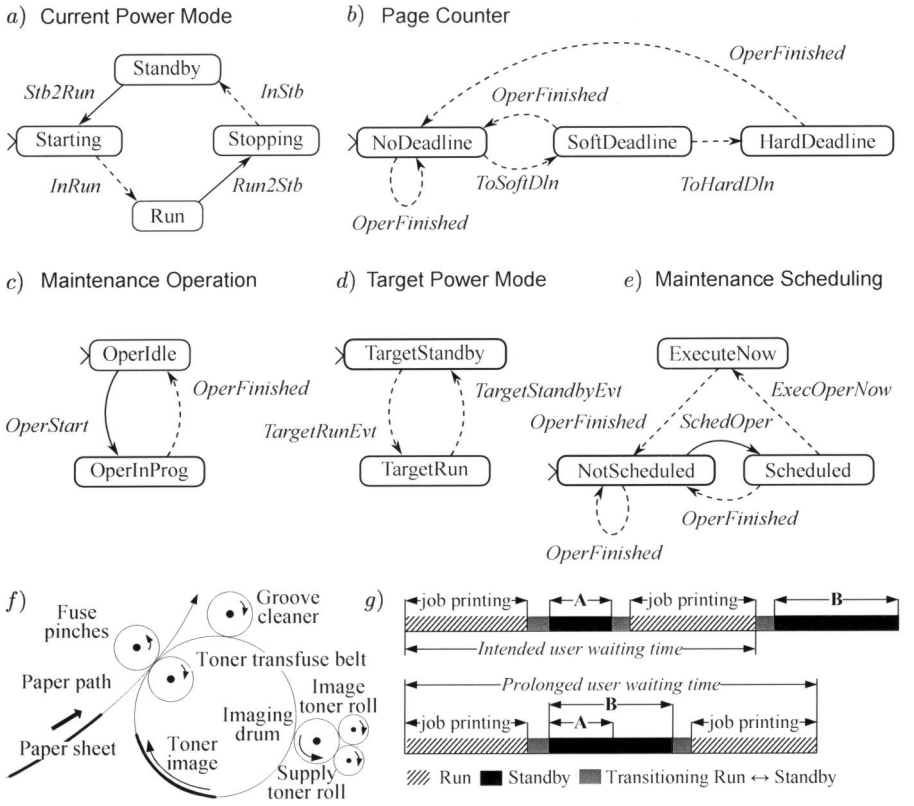

Figure 3.3: Five automata shown in Figure 3.3a-3.3e constitute the main components of maintenance coordination inside an industrial printer. An abstraction of the printing setup is shown in Figure 3.3f, while an example of coordination procedures is illustrated in Figure 3.3g. Figures 3.3f and 3.3g were previously used in [63].

addition, all other behavior of the otherwise correctly functioning distributed components which make up the printing process should be left in place. That is, the controlled system affected by the imposed restrictions due to more stringent maintenance coordination should be maximally permissive. The various distributed components which model the uncontrolled part of the printing process are shown in Figure 3.3a-3.3e.

The functionality of the various components depicted in Figure 3.3 is now briefly discussed at an informal level. Dashed lines are again used in Figure 3.3 to indicate uncontrollable events. These system models comprise

a natural abstraction in the sense that the scheduling of only one maintenance operation is considered and timing issues are not taken into account. The components Target Power Mode and Maintenance Scheduling execute scheduling tasks. The Current Power Mode, Maintenance Operation and Page Counter components are responsible for handling maintenance tasks and actuating underlying hardware control. The Current Power Mode sets the power mode to **Run** or **Standby**, in reaction to the enabling signals *Stb2Run* and *Run2Stb* respectively. A confirmation is replied by means of *InStb* and *InRun*. The Maintenance Operation component switches between carrying out maintenance, triggered by *OperStart*, or being idle, as confirmed by the *OperFinished* signal. The component Page Counter is responsible for counting the number of printed pages since maintenance has taken place. It sends the signals *ToSoftDln* and *ToHardDln* when soft and hard deadlines are reached. Once the maintenance has finished, the page counter module is reset by receiving the *OperFinished* signal from the Maintenance Operation component. The controlling unit Target Power Mode defines which mode is requested by the manager by sending the control signals *TargetStandbyEvt* and *TargetRunEvt* respectively. The Maintenance Scheduling receives a request for maintenance via the signal *SchedOper*, which is forwarded to the manager. Confirmation is sent by the manager using the *ExecOperNow* event. In addition, it receives feedback from Maintenance Operation to confirm that maintenance has finished in order to reset the scheduling.

The case study previously introduced in general terms is now subjected to for a number of control objectives. As the starting point for synthesis, the parallel composition of the five components in Figure 3.3 is constructed, where it is assumed that all components share the same event alphabet. Upon this intermediate result, synthesis for six separate control specifications, which are partly based on earlier research done in [63], will be synthesized. Informal and formal interpretations for these control specifications are listed below. The expression $p \Rightarrow q$ is used as an abbreviation for $\neg p \vee q$ and bold face is used in the formal specifications to indicate state name propositions. For specifications 2-4, $\neg\langle e \rangle$ is expressed as $[e]\,false$.

1. Maintenance operations can be performed only when the printing process is in standby, formalized as:

$$\Box\,(\mathbf{OperInProg} \Rightarrow \mathbf{Standby})$$

2. Maintenance operations can be scheduled only when a soft deadline has been reached and there are no print jobs in progress, or when a hard deadline has passed, formalized as:

$$\Box\,(<SchedOper> \Rightarrow \\ ((\mathbf{SoftDeadline} \wedge \neg\mathbf{TargetRun}) \vee \mathbf{HardDeadline}))$$

3. Maintenance operations can be started only after being scheduled, formalized as:

$$\Box\,(<OperStart> \Rightarrow \mathbf{ExecuteNow})$$

4. The power mode of the printing process must follow the power mode dictated by the managers, unless overridden by any pending maintenance operation, formalized using two specifications as:

$$\Box\,(<Stb2Run> \Rightarrow (\mathbf{TargetRun} \wedge \neg\mathbf{ExecuteNow})) \\ \Box\,(<Run2Stb> \Rightarrow (\mathbf{TargetStandby} \vee \mathbf{ExecuteNow}))$$

5. After a maintenance operation has finished, the system should be in the TargetStandby state, formalized as:

$$\Box\,[\,OperFinished\,]\,\mathbf{TargetStandby}$$

6. Once a maintenance operation has started, it should be completed immediately, such that no new maintenance operation is scheduled and the system is not in the Standby state, formalized as:

$$\Box\,[\,OperStart\,]\,[\,OperFinished\,]\,(\mathbf{NotScheduled} \wedge \neg\mathbf{Standby})$$

A controlled system conforming to control specifications 1-4 may be synthesized using the traditional event-based supervisory control framework [76] (including synthesis for marker state reachability), but also using the theories proposed in this thesis. Both synthesis techniques result in the exact same model consisting of 60 states and 172 transitions. Requirements 5-6 are used to illustrate the extended expressibility for control specifications the theories in this thesis provide. Synthesis for specifications 1-6 results in a further restricted controlled system consisting of 56 states and 158 transitions. This case study, albeit small, reflects that the synthesis theory put forward in this thesis is able to solve synthesis problems which were considered in earlier research.

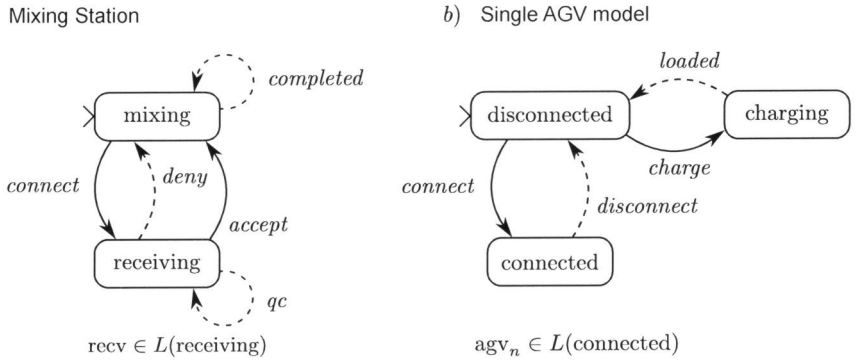

Figure 3.4: Components in a chemical production plant for which a guiding control strategy is derived. A single mixing station (Figure 3.4a) is combined with multiple instances of the AGV model (Figure 3.4b). Models are combined under parallel composition, such that a *connect* event between the mixing station and precisely one AGV is the only synchronizing event.

3.6 Scalability Analysis

The purpose of this section is to analyze the scalability of the algorithm introduced in Section 3.2 by means of an extensible case study, which is loosely based on the work in [67]. A control coordination strategy for a variable number of automated guided vehicles (AGVs) and a mixing station within a chemical production plant will be computed. Since this model can be instantiated at variable plant sizes and due to the fact that each new AGV introduces separate control objectives, this case is well-suited to analyze the scalability of the type of synthesis presented in this thesis.

The case study at hand, as illustrated in Figure 3.4, will now be introduced. A single mixing station (Figure 3.4a) in a chemical production plant is combined with several AGVs. A single instance of the AGV model is shown in Figure 3.4b. As shown in Figure 3.4b, for each $1 \leq n \leq N$, the label agv_n is added to the connected state in AGV n, if there are N AGVs in the entire model. Synchronization takes place under parallel composition in such a way that the mixing station synchronizes with precisely one AGV over the *connect* event. This is the only point of synchronization in the constructed parallel model. The *connect* event represents the AGV connecting to the station where it delivers a chemical component. The AGV may then disconnect after which it can be ordered to charge its batteries. If the mixing station

has received a chemical component from the AGV, it performs quality checks after which the chemical may be either accepted or denied for mixing. The *completed* event represents the situation where the station has completed its task of mixing, after which its mixing tank is supposed to be immediately cleaned. For this purpose, a cleaning agent is again delivered by an AGV.

It is intended to synthesize a controlled system for the behavior of both the AGVs and the mixing station, which needs to adhere to several control specifications, as listed below:

1. If there are N AGVs, then after N direct subsequential quality checks at least one AGV should have been disconnected. This specification ensures that the system does not solely perform quality checks while not sending the AGVs back. This control objective may be formalized as follows:

$$\Box \left(([\, qc \,])^N \left(\bigvee_{1 \leq n \leq N} \neg agv_n \right) \right)$$

2. If the mixing process is finished, as signalled by the *completed* event, cleaning should happen immediately. In this case, AGV_1 should either be already connected or immediately available for connecting, since this is the only AGV which is allowed to deliver the cleaning agent:

$$\Box \, [\, completed \,] \; (agv_1 \vee <connect> agv_1)$$

3. If no AGV is connected and if the mixing station is not receiving, then at least one AGV should be immediately available for connecting. This may be expressed in terms of the agv_n labels:

$$\Box \left(\left(recv \vee \bigvee_{1 \leq n \leq N} agv_n \right) \vee <connect> \right)$$

The table below shows the computation results for increasing numbers of AGVs in terms of original plant size (as a single $\mathcal{K}$-model, integrated under the aforementioned form of parallel composition), the size of $S^0_{k,f}$ and the size of the final resulting models. All data for model sizes is expressed as S/T, where S refers to the number of states and T refers to the number of transitions. In addition, maximal memory consumption and the number of iterations required to achieve the final synthesis result are mentioned. Memory consumption here refers to the maximal number of bytes allocated at some point during the execution of the program.

#AGVs	plant (k) size	$S^0_{k,f}$ size	$S^n_{k,f}$ size	memory	iterations
2	18/78	51/236	23/93	2Mb	3
3	54/297	182/1060	105/558	17Mb	3
4	162/1080	633/4406	487/2956	87Mb	3

The obtained results are briefly characterized here in terms of related research. In [87], the synthesis for a comparable formula consisting of a conjunction of invariants of reachability expressions is considered. An interesting similarity to the research described in this thesis occurs when synthesis for variable-size plant models in the orders of tens of states results in large (up to several gigabytes) state models which remain computable in minutes. Work in [26] also shows exponential growth of the controller in terms of the size of the synthesized formula. Many comparable works indicate exponential growth of both the resulting model and running time once either the plant model or the synthesized formula expands linearly [5, 58, 28]. Different results were obtained as well. [45] and [74] do not observe such strong increases in memory consumption, although in the first case this might be partly due to the specific approach applied in [45]. An offset in results for synthesis for formulas in temporal logic may be noticed based on work in [81], where abstractions due to non-determinism result in effective plant models having significantly smaller state spaces. Based on these observations, the hypothesis arises that if a state space reduction via abstraction through the introduction of non-determinism precedes a memory demanding computation, the net result may be more efficient compared to observations of the synthesis problem in itself.

3.7 Closing Remarks

The final stage of two chapters has now been reached which present a novel approach to controlled system synthesis for modal logic on non-deterministic plant models. The behavior of a Kripke-structure with labeled transitions is adapted such that it satisfies the synthesized controlled behavior, expressed as a formula in modal logic. The relationship between the synthesis result and the original plant model adheres to important notions in control synthesis: controllability and maximal permissiveness. The controlled behavior specification logic also comprises deadlock-freeness and marker state reachability. The synthesis approach, via a reduction on modal expressions combined with an iteratively applied synthesizability test for formulas assigned to target states of transitions, results in an effective synthesis procedure which

may be implemented in a straightforward fashion. The synthesis theory presented in this thesis allows the full expressibility of Ramadge-Wonham control synthesis on deterministic plant models. An implementation of the synthesis technique has been developed in the C programming language. This will allow an assessment of various parameters regarding tractability and efficiency of this algorithm in further case studies. In particular, the synthesizability of reachability formulas may be modified such that recomputations can be avoided using dynamic programming or similar techniques. To enable a clear focus on the theoretical results and proofs presented in this chapter, an embedding of such optimizations was not included. Partial bisimilarity as a means to express controllability, as well as other behavioral preorders for this purpose, will also have to be studied further.

Chapter 4

Control Synthesis and Multiple Solutions

This chapter applies a different synthesis approach compared to the previous two chapters. Whereas in the main synthesis methodology defined in Chapter 2 a unique solution is derived, research in this chapter involves multiple solutions. Since this research aim is more challenging due to obtaining more than one solution, control objectives are limited to Hennessy-Milner logic [34]. Furthermore, part of the research framework applied in the previous chapters is inherited here. Again, plant models are allowed to be nondeterministic and synthesis results are required to be maximally permissive. An important difference between this chapter and the previously described work is due to an abstraction with regard to controllability of individual events: all events are assumed to be controllable in this chapter. Figure 4.1 depicts a global overview of the applied synthesis method in this chapter. More formally, synthesis is defined in such a way that a *set* of results is obtained, since multiple solutions are induced by having an unrestricted disjunction operator. Each result in the synthesized set satisfies the given HML formula, and is related to the original structure via simulation. This result set contains at least one outcome which is maximal with regard to the simulation preorder.

The remainder of this chapter is organized as follows. Section 4.1 contains a number of preliminary definitions. Note that a number of definitions in Section 4.1 are adaptations of earlier definitions in slightly modified form. Section 4.2 discusses the specific synthesis approach considered in this chap-

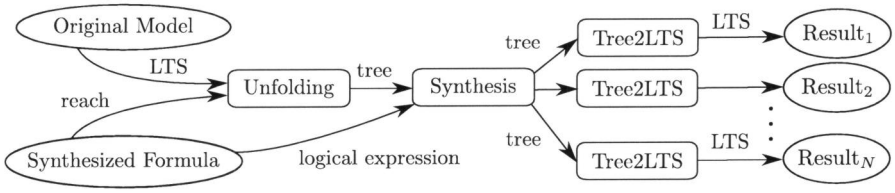

Figure 4.1: Illustrating key parts of the synthesis process. The first step consists of unfolding the LTS up to the depth of synthesized formula in the form of a partial tree representation of behavior. After the actual synthesis itself is applied, each obtained result is again converted back into an LTS

ter. Section 4.3 then details a partial tree-representation of the plant model which is used as a preceding step for this specific type of synthesis. Section 4.4 contains an operational definition of the synthesis construction, which is subsequently shown to be correct in Section 4.6 and Section 4.7. An algorithmic representation of this specific synthesis construction is the subject of Section 4.5. Computer verified proofs for the theories introduced in this chapter are contained in Section 4.8.

4.1 Definitions

Just as in Section 2.2, existence of a set of events $\mathcal{E}$ and a set of atomic propositions $\mathcal{P}$ is assumed. A difference with regard to the previous chapters is that the set of basic states $\mathcal{X}$ is assumed at a global level. The atomic propositions are interpreted with regard to a labeling function $L : \mathcal{X} \mapsto 2^{\mathcal{P}}$ relating states to properties. These are used to capture the state-based information of a system. An atomic proposition $p \in \mathcal{P}$ holds in state $x \in \mathcal{X}$ if and only if $p \in L(x)$. Assuming these definitions at a global level brings clarity to the definition of synthesis later on, while not limiting the scope of this definition. Since labels are not added or removed from states during synthesis, the global definition of $\mathcal{P}$ does not affect the synthesis semantics. In this chapter, two types of structures are used to express structural behavior. The first is the standard labeled transition system (LTS). The second is a partial tree representation upon which synthesis is defined. This will be the topic of Section 4.3. Both behavioral models capture process dynamics via labeled transitions between states. It is assumed that they are finitely branching, which means that each state has finitely many outgoing transitions. The definition of an LTS differs from Definition 2.1, since state-labeling is defined at a global level.

Definition 4.1. Given a state space $X \subseteq \mathcal{X}$, transition relation $\longrightarrow \subseteq X \times \mathcal{E} \times X$, and initial state x, an LTS is defined as the tuple $(X, \longrightarrow, x)$.

The notation $\mathcal{G}$ will be used to denote the universe of LTSes. For transition relation $\longrightarrow \subseteq X \times \mathcal{E} \times X$, the notation $x \xrightarrow{e} x'$ is used to indicate that $(x, e, x') \in \longrightarrow$. The standard simulation preorder and bisimulation equivalence [31] are used to relate elements in $\mathcal{G}$, according to the definitions shown below. Note that unlabeled transition systems are used in [31], but results can be adapted and applied straightforwardly.

Definition 4.2. The LTSes $g' = (X', \longrightarrow', x')$ and $g = (X, \longrightarrow, x) \in \mathcal{G}$ are related via *simulation* (notation: $g' \preceq g$) if there exists a relation $R \subseteq X' \times X$ such that $(x', x) \in R$ and for all $(y', y) \in R$ it holds that:

1. $L'(y') = L(y)$; and
2. For all $e \in \mathcal{E}$ and $z' \in X'$ such that $y' \xrightarrow{e}' z'$, there exists a $z \in X$ such that $y \xrightarrow{e} z$ and $(z', z) \in R$.

The notation $g' \preceq_R g$ will be used to indicate that $g' \preceq g$ as witnessed by R.

The first clause in Definition 4.2 requires equality of the sets of satisfied basic properties. This reflects the synthesis semantics where validity is enforced by removal of transitions, while state-based properties are not adjusted.

Definition 4.3. If $g' \preceq_R g$ and $g' \preceq_{R^{-1}} g$ according to Definition 4.2, then g' and g are related via *bisimulation* (notation: $g' \leftrightarrows_R g$).

Simulation is reflexive as witnessed by the identity relation on $\mathcal{G}$ and transitive by composition of the two underlying witness relations. Bisimilarity is reflexive and transitive for the very same reasons, but it is symmetric by the inverted witness relation as well, and therefore an equivalence. These are standard results [31].

The standard definition of formulas in Hennessy-Milner Logic (HML) [34] is extended with a test for basic properties $p \in \mathcal{P}$, and its negation $\neg p$.

Definition 4.4. The set $\mathcal{F}$ of HML-formulas is defined as follows:

$$\mathcal{F} \Rightarrow \mathit{true} \mid \mathit{false} \mid \mathcal{P} \mid \neg\mathcal{P} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid [\mathcal{E}]\mathcal{F} \mid \langle\mathcal{E}\rangle\mathcal{F}$$

The formulas *true* and *false* indicate truth and falsehood respectively, while the formula p, for $p \in \mathcal{P}$, can be used to test whether atomic proposition p holds in a specific state. Negation $\neg p$ is defined for state-based properties only. The meaning of the operators for conjunction $\wedge$ and disjunction $\vee$ is

as expected. The universal modality $[e]f$ tests whether f holds after every e-step, while $<e>f$ tests for the existence of an e-step after which f holds. Negation at the level of atomic propositions is sufficient to extend the operator $\neg$ to the full set $\mathcal{F}$, as shown in [34]. Besides being less expressive, the main differences between Definition 4.4 and Definition 2.8 are the inclusion of unrestricted disjunction and the fact that the operator $<e>f$ is no longer restricted to $e \in \mathcal{C}$. The following measure is defined on formulas to express the modal depth of a formula:

Definition 4.5. The function $depth : \mathcal{F} \mapsto \mathbb{N}$ is defined in the following way, for $p \in \mathcal{P}$, $e \in \mathcal{E}$ and $f, f_1, f_2 \in \mathcal{F}$:

$$\begin{array}{lcl} depth\,(true) & = & 0 \\ depth\,(false) & = & 0 \\ depth\,(p) & = & 0 \\ depth\,(\neg p) & = & 0 \\ depth\,(f_1 \wedge f_2) & = & max\,(depth\,(f_1), depth\,(f_2)) \\ depth\,(f_1 \vee f_2) & = & max\,(depth\,(f_1), depth\,(f_2)) \\ depth\,([e]\,f) & = & 1 + depth\,(f) \\ depth\,(<e>f) & = & 1 + depth\,(f) \end{array}$$

The validity of formulas in $\mathcal{F}$ is expressed with respect to the labeled transitions systems $\mathcal{G}$ using the *valuation* function $\models$, defined in the following way:

Definition 4.6. The predicate $\models$ over $\mathcal{G} \times \mathcal{F}$ is defined for $g = (X, \longrightarrow, x) \in \mathcal{G}$, $f, f_1, f_2 \in \mathcal{F}$, $e \in \mathcal{E}$, $x \in \mathcal{X}$ and $p \in \mathcal{P}$ by the following derivation rules:

$$\frac{}{g \models true} \qquad \frac{p \in L(x)}{(X, \longrightarrow, x) \models p} \qquad \frac{p \notin L(x)}{(X, \longrightarrow, x) \models \neg p} \qquad \frac{g \models f_1 \quad g \models f_2}{g \models f_1 \wedge f_2} \qquad \frac{g \models f_1}{g \models f_1 \vee f_2}$$

$$\frac{g \models f_2}{g \models f_1 \vee f_2} \qquad \frac{\forall\, x \xrightarrow{e} x' \quad (X, \longrightarrow, x') \models f}{(X, \longrightarrow, x) \models [e]\,f} \qquad \frac{x \xrightarrow{e} x' \quad (X, \longrightarrow, x') \models f}{(X, \longrightarrow, x) \models <e>f}$$

If $g \models f$ then g *satisfies* the formula f. Note that validity for HML-formulas is preserved under bisimulation [31].

4.2 Approach

This section concerns the specific synthesis approach applied in this chapter, including the differences as compared to the synthesis setup in Chapter

2. Several of the constructions involved will be illustrated, and the caveats that were encountered during the research are noted. A formal definition of synthesis will then follow in the later sections in terms of a partial tree representation of the LTS.

First, a closer look is taken at synthesis for the various elements in $\mathcal{F}$. It is clear that synthesis for *true* should be neutral as no modification of the LTS is required to satisfy this formula. On the other hand, synthesis of the formula *false* should not yield any result because no possible modification to the original structure exists which achieves validity for this formula. This is a subtle difference with the approach in Chapter 2. Synthesis for *false* in Chapter 2 results in failure to find a solution. However, in this chapter synthesis results in a set of solutions. Synthesis for *false* then results in the empty set. The formulas p, for $p \in \mathcal{P}$, are always evaluated and synthesized with respect to a single state x in the state space $\mathcal{X}$. If $p \in L(x)$, then synthesis should be the same as if the formula were *true*, where no modification of the LTS is required. On the other hand, if $p \notin L(x)$, then the formula should be treated as if it were *false* and the empty set of synthesized results should be returned. Note that assigning the basic property p to x if $p \notin L(x)$ is not desired, as this would add information to the LTS, thereby invalidating the basic principle of synthesis of not introducing additional new behavior or properties. The inverted procedure is followed for the negation of basic properties $\neg p$. If $p \notin L(x)$, then no modification needs to be applied to satisfy the formula $\neg p$. However, if $p \in L(x)$ then the formula $\neg p$ cannot be satisfied for x and therefore synthesis should not result in any satisfying model.

Now the other elements of $\mathcal{F}$ are considered, thereby first taking a look at the formulas $[e]f$ and $<e>f$, since any non-trivial example regarding the operator $\wedge$ and $\vee$ uses $[e]f$ or $<e>f$. For the formulas of type $[e]f$, essentially the same principle is applied as shown in Figure 2.2. The precise details regarding unfolding are considered at a later stage. Assume a formula $[e]f$. Then, synthesis is applied recursively for f after each e-step. If such synthesis does not result in any solution, for instance if $f \equiv \mathit{false}$, then the corresponding e-step is removed. On the other hand, if synthesis is successful, the e-step is retained and the LTS is modified recursively after the e-step in order to satisfy f. In comparison to the synthesis approach in Chapter 2, this important difference needs to be highlighted here, since recursive application of synthesis may result in multiple solutions. At this point it is important to stress that deleting a disallowed behavior does not contradict the maximality requirement, since maximal permissiveness is defined with respect to all *satisfying* simulants.

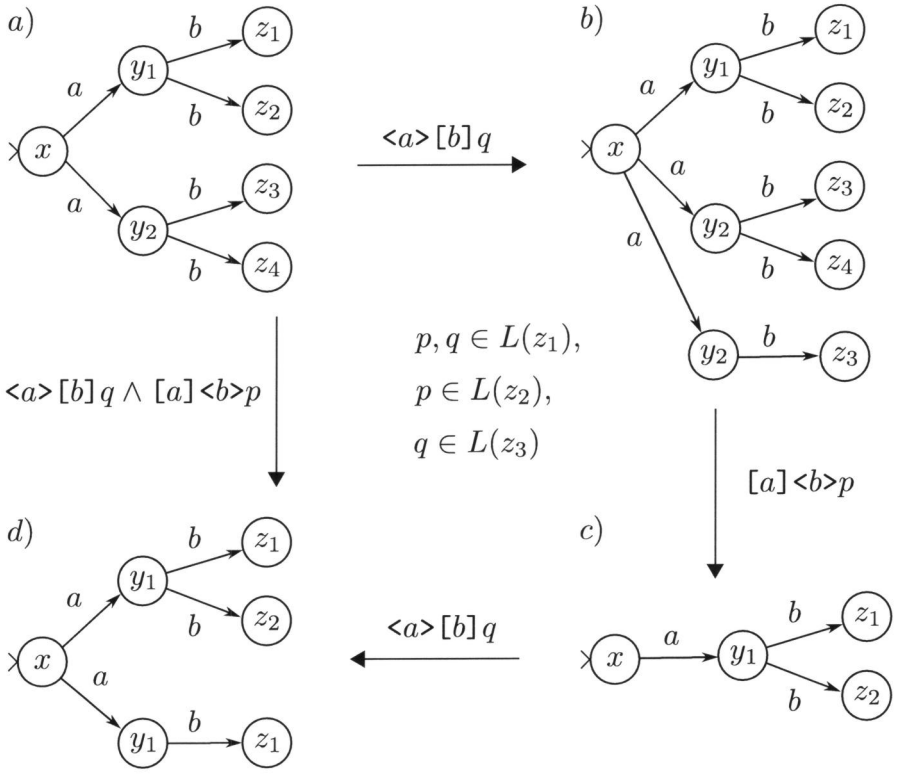

Figure 4.2: Synthesis for the operator $\wedge$ is realized via alternating applications of synthesis for both conjuncts. Figure 4.2a shows the input-LTS and Figure 4.2d the final result after synthesis for $\langle a \rangle [b] q \wedge [a] \langle b \rangle p$, via intermediate steps shown in Figure 4.2b and Figure 4.2c.

For the formulas of type $\langle e \rangle f$, an attempt is made to synthesize f after each e-step. If none of these steps is successful, synthesis for the formula $\langle e \rangle f$ does not result in a valid outcome. Otherwise, synthesis proceeds recursively after an e-step while all other transitions are left in place unmodified. In addition, maximal permissiveness needs to be taken into account. Therefore, in order to give an appropriate definition for synthesis of formulas $\langle e \rangle f$, the unmodified transitions after the e-step need to be preserved as well. Note that, analogous to the synthesis for $[e] f$, the synthesis for $\langle e \rangle f$ might result in multiple solutions if f can be synthesized in multiple ways after the e-step. Also, the presence of multiple e-steps might result in multiple solutions if

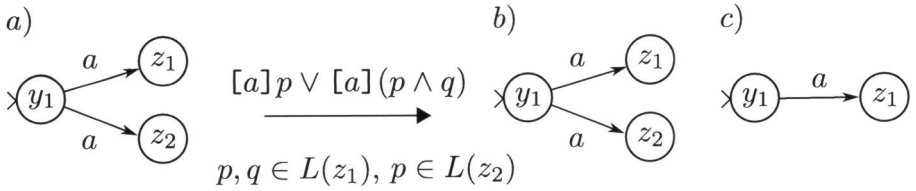

Figure 4.3: Synthesis of the formula $[a]p \vee [a](p \wedge q)$ upon the model in Figure 4.3a results in a maximal solution shown in Figure 4.3b as well as a non-maximal solution in Figure 4.3c, due to the nature of synthesis for disjunction, where each operand is considered separately and results are combined into a set of synthesis products.

each of these e-steps allows the synthesis of the $<e>f$ formula.

The next two operators to consider are conjunction and disjunction. With regard to disjunction, the example in Figure 2.4 is still relevant. Since the synthesis method in this chapter results in multiple solutions, the possibility arises to include multiple outcomes in the set of results. This leads to compositionality of this synthesis method for disjunction. Synthesis for a formula $f \vee g$ therefore results in the union of the solutions for f and g respectively.

The operator $\wedge$ introduces additional complications and therefore requires an entirely different approach compared to the setup in Chapter 2. As shown in Figure 4.2, multiple applications of the synthesis for each conjunct might be required to obtain a synthesis result which satisfies both formulas. The input-LTS, as shown in Figure 4.2a, is modified in order to satisfy the formula $<a>[b]q \wedge [a]<b>p$. The end result, as shown in Figure 4.2d, is obtained via the intermediate steps 4.2b and 4.2c. Synthesis for $<a>[b]q$ is applied to the original in Figure 4.2a, resulting in the model in Figure 4.2b. Consequently, synthesis for $[a]<b>p$ is applied in Figure 4.2b, resulting in Figure 4.2c. In the last step, synthesis for $<a>[b]q$ is again applied, resulting in the final outcome in Figure 4.2d, which also satisfies the second conjunct. Observe that now both conjuncts are satisfied and no more applications of synthesis are required.

This definition for synthesis for conjunction is generalized later on, where synthesis for conjunction will be defined as a fixpoint construction that alternatingly applies synthesis for both conjuncts. Note that two possible intermediate results exist after the first synthesis step in Figure 4.2a, but only one is shown for clarity.

Two important general aspects of synthesis need to be taken into account, before a formal definition for this type of synthesis may be treated in detail: *unfolding* and *maximality*, or maximal permissiveness. As stated before, syn-

thesis outcomes are required to be maximally permissive in the sense that the least amount of modification is applied in order to satisfy the given formula. Maximality is reflected in two ways in the synthesis process:

1. Synthesis for a formula $[e]f$ should only remove an e-step if f can not be satisfied in the state reached after the e-step.

2. The set of synthesis products should contain a maximal solution, with respect to all simulants of the original model which satisfy the synthesized formula.

The first synthesis requirement is illustrated in Figure 2.2. Regarding the second property, it should be noted that non-maximal solutions can not always be avoided. As shown in Figure 4.3, the set of synthesis results for the formula $[a]p \vee [a](p \wedge q)$, contains a non-maximal solution that can not be avoided due to the nature of the synthesis for disjunction, where each operand is considered separately. In this regard it is again important to interpret maximality with respect to *all* satisfying simulants. That is, if a satisfying simulant of the input-LTS exists, then this simulant is related via simulation to one of the synthesis outcomes.

4.3 Tree Representation

Synthesis will be defined in such a way that it only manipulates elements of the transition system within reach of the synthesized formula. Additionally, unfolding is required in order to obtain a maximal solution. Therefore, a partial tree representation of the transition structure is introduced which allows a clear and coherent definition of synthesis, and also contains an embedded unfolding.

One might wonder why a new formalism is required, while it would also seem plausible to simply rely upon the LTS formalism and unfold up to a given depth. However, as the examples in the previous chapters have shown, a single state may play multiple roles at various stages of synthesis. Therefore, it is not possible to specify synthesis as defined in this chapter via *in situ* changes to the transition relation. Also, the partial tree representation allows a definition of synthesis in terms of a distinction on formula type based on a clear definition in terms of operational rules. Compared to the approach in Chapter 2, this partial tree representation is better able to handle multiple solutions.

The intuitive idea behind this construction can be stated in the following way: For as far as the depth of the synthesized formula, the transitions between the respective states are represented in tree form. Beyond the formula reach, behavior is modeled via the usual transition relation. This allows for the same state being considered at different depths, if this state plays a different role at various stages of synthesis, for instance if it is contained in a cycle. Furthermore, the partial tree representation allows for a clear and coherent operational definition of synthesis, since points of transition removal and recursive application of synthesis are directly clear from this structure. Formally, the universe of these structures is represented as $\mathcal{K}_{\rightarrow}$, which may be interpreted as a dependent type, with regard to the transition relation $\longrightarrow$, via the construction in Definition 4.7.

Definition 4.7. Given a state space $X \subseteq \mathcal{X}$, state $x \in X$, and transition relation $\longrightarrow \subseteq X \times \mathcal{E} \times X$, the dependent type $\mathcal{K}_{\rightarrow}$ is defined in the following way:

$$\mathcal{K}_{\rightarrow} \Rightarrow \langle x \rangle_{\rightarrow} \mid \langle x, T \rangle_{\rightarrow} \text{ for } T \subset \mathcal{E} \times \mathcal{K}_{\rightarrow}$$

In an attempt to bring more clarity to Definition 4.7, its two definitional parts are considered separately.

1. The construct $\langle x \rangle_{\rightarrow}$ represents a final node of the tree. This means that from this point on, behavior is modeled via the transition relation $\rightarrow$.
2. The elements $\langle x, T \rangle_{\rightarrow}$ for $T \subset \mathcal{E} \times \mathcal{K}_{\rightarrow}$ represent the actual nodes of the tree. These consist of a state, combined with a continuation of behavior via underlying tree elements, and their corresponding events.

Steps between elements in $\mathcal{K}_{\rightarrow}$ are created in the following way. For each $k, k' \in \mathcal{K}_{\rightarrow}$, and $e \in \mathcal{E}$, a step $k \xrightarrow{e} k'$ can be obtained if one of the following two conditions is satisfied:

1. If $k = \langle x \rangle_{\rightarrow}$ and $k' = \langle x' \rangle_{\rightarrow}$ and $x \xrightarrow{e} x'$, then $k \xrightarrow{e} k'$
2. If $k = \langle x, T \rangle_{\rightarrow}$ and $(e, k') \in T$ then $k \xrightarrow{e} k'$

The construction for $\mathcal{K}_{\rightarrow}$ is a non-standard and non-straightforward expression for structural behavior. This is justified by the obtained clarity in the definition for synthesis, and the ability to capture embedded unfolding via this structure. In Figure 4.4, an example is shown to illustrate an LTS as well as its unfolded partial tree representation. In Figure 4.4a, an LTS is shown

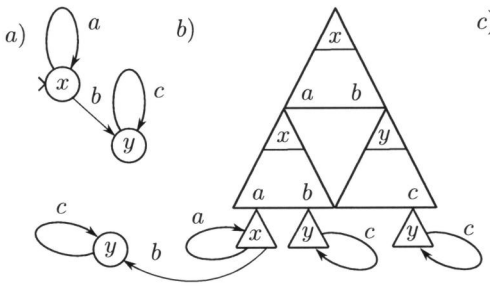

Figure 4.4: The LTS shown in a) is converted into a structure $\mathcal{K}_{\rightarrow}$ illustrated in b), and given as a formal definition in c). Unfolding is applied to depth 2, in this particular case.

which is subsequently unfolded to depth two, and represented via the tree shown in Figure 4.4b. Unfolding up to depth two would be applied if, for instance, the formula $<a>[b]p$ were to be synthesized. The resulting formal definitions for this construction in $\mathcal{K}_{\rightarrow}$ are shown in Figure 4.4c.

Since steps are now defined with respect to $\mathcal{K}_{\rightarrow}$, it is possible to transfer the standard behavioral relations of simulation and bisimulation to this structure. These are shown in Definitions 4.8 and 4.9 respectively.

Definition 4.8. For $k' \in \mathcal{K}_{\rightsquigarrow}, k \in \mathcal{K}_{\rightarrow}$, simulation is defined as the existence of a relation $R \subseteq \mathcal{K}_{\rightsquigarrow} \times \mathcal{K}_{\rightarrow}$, such that $(k', k) \in R$ and the following holds for all $(m', m) \in R$:

1. If x' and x are the respective root states in m' and m, then $L(x') = L(x)$
2. For all $m' \overset{e}{\rightsquigarrow} n'$, there exists a $n \in \mathcal{K}_{\rightarrow}$ such that $m \overset{e}{\longrightarrow} n$ and $(n', n) \in R$

Again, $k' \preceq_R k$ is used to indicate that $k' \preceq k$ as witnessed by R. Note that the different transition relations $\rightsquigarrow$ and $\longrightarrow$ are used to highlight the fact that the respective $\mathcal{K}$- structures are defined with respect to a different underlying transition relation.

Definition 4.9. If $k' \preceq_R k$ and $k' \preceq_{R^{-1}} k$, then k' and k are related via bisimulation (notation: $k' \underline{\leftrightarrow} k$).

Validity for formulas in $\mathcal{F}$ can now be expressed with regard to the structure $\mathcal{K}_{\rightarrow}$, via Definition 4.10.

Definition 4.10. The predicate $\vDash$ over $\mathcal{K}_{\rightarrow} \times \mathcal{F}$ is defined for $k', k \in \mathcal{K}_{\rightarrow}$, $f, f_1, f_2 \in \mathcal{F}$, $e \in \mathcal{E}$, $x \in \mathcal{X}$, $p \in \mathcal{P}$ by the following deduction rules:

$$\frac{}{k \vDash true} \qquad \frac{p \in L(x)}{\langle x \rangle_{\rightarrow} \vDash p} \qquad \frac{p \in L(x)}{\langle x, T \rangle_{\rightarrow} \vDash p} \qquad \frac{p \notin L(x)}{\langle x \rangle_{\rightarrow} \vDash \neg p}$$

$$\frac{p \notin L(x)}{\langle x, T \rangle_{\rightarrow} \vDash \neg p} \qquad \frac{k \vDash f_1 \quad k \vDash f_2}{k \vDash f_1 \wedge f_2} \qquad \frac{k \vDash f_1}{k \vDash f_1 \vee f_2} \qquad \frac{k \vDash f_2}{k \vDash f_1 \vee f_2}$$

$$\frac{\forall k \xrightarrow{e} k' \; k' \vDash f}{k \vDash [e] f} \qquad \frac{k \xrightarrow{e} k' \quad k' \vDash f}{k \vDash \langle e \rangle f}$$

It is clear that each LTS $g \in \mathcal{G}$ can be represented as an element of $\mathcal{K}_{\rightarrow}$, since the structure $\langle x \rangle_{\rightarrow}$ is isomorphic to g if g has x as its initial state and $\rightarrow$ as its transition relation. To convert this structure into an unfolded $\mathcal{K}$-representation, a function $unfold : \mathcal{K} \times \mathbb{N} \mapsto \mathcal{K}$ is given in Definition 4.11. In addition, a test for unfoldedness is given as the predicate $unf \subseteq \mathcal{K} \times \mathbb{N}$ in Definition 4.12. Lemma 4.1 then shows how the unfolded structure is indeed bisimilar to the unmodified structure.

Definition 4.11. Let $\langle x \rangle_{\rightarrow} \in \mathcal{K}_{\rightarrow}$, then for each $n \in \mathbb{N}$, a $k \in \mathcal{K}_{\rightarrow}$ can be constructed which is unfolded to depth n, using the following function $unfold$: $\mathcal{K} \times \mathbb{N} \mapsto \mathcal{K}$:

$$\begin{array}{lcl} unfold\,(k, 0) & = & k \\ unfold\,(\langle x \rangle_{\rightarrow}, n) & = & \langle x, \{(e, unfold\,(\langle x' \rangle_{\rightarrow}, n-1)) \mid x \xrightarrow{e} x'\} \rangle_{\rightarrow} \\ unfold\,(\langle x, T \rangle_{\rightarrow}, n) & = & \langle x, \{(e, unfold\,(k', n-1)) \mid (e, k') \in T\} \rangle_{\rightarrow} \end{array}$$

It is presumed without proof that every result of the $unfold$ function is indeed unfolded up to the given depth, as can be tested by the following predicate.

Definition 4.12. The predicate $unf \subseteq \mathcal{K}_{\rightarrow} \times \mathbb{N}$ is defined for $x \in \mathcal{X}$, $T \subset \mathcal{E} \times \mathcal{K}_{\rightarrow}$ and $n \in \mathbb{N}$ by the following definition:

$$\begin{array}{lcl} unf(\langle x \rangle_{\rightarrow}, n) & \iff & n = 0 \\ unf(\langle x, T \rangle_{\rightarrow}, 0) & \iff & true \\ unf(\langle x, T \rangle_{\rightarrow}, n+1) & \iff & \forall (e, k) \in T.\ unf(k, n) \end{array}$$

The following lemma shows that the result of the *unfold* function is indeed bisimilar to its original input.

Lemma 4.1. For $k \in \mathcal{K}$ and $n \in \mathbb{N}$, it holds that $k \leftrightarrow \mathit{unfold}\,(k, n)$.

Proof. This property can be shown by induction towards n. If $n = 0$ then $\mathit{unfold}\,(k, 0) = k$, by Definition 4.11, and clearly $k \leftrightarrow k$, by reflexivity of bisimilarity. For the inductive case, assume that $\mathit{unfold}\,(k', n) \leftrightarrow k'$, for all $k' \in \mathcal{K}_{\rightarrow}$. It now needs to be shown that $\mathit{unfold}\,(k, n+1) \leftrightarrow k$. A distinction is made between the two forms of k, as given in Definition 4.7.

If $k \equiv \langle x \rangle_{\rightarrow}$, then for each step $x \xrightarrow{e_i} x_i$ it holds that $\mathit{unfold}\,(\langle x_i \rangle_{\rightarrow}, n) \leftrightarrow_{R_i} \langle x_i \rangle_{\rightarrow}$, for some R_i, by induction. Bisimilarity is then shown by choosing $R' = \bigcup_i R_i \cup \{(\mathit{unfold}\,(k, n+1), k)\}$, in order to prove that $\mathit{unfold}\,(k, n+1) \leftrightarrow_{R'} k$.

If $k \equiv \langle x, T \rangle_{\rightarrow}$, then for each $(e, k_i) \in T$ it holds that $\mathit{unfold}\,(k_i, n) \leftrightarrow_{R_i} k_i$ for some R_i, by induction. Again, this is solved by choosing $R' = \bigcup_i R_i \cup \{(\mathit{unfold}\,(k, n+1), k)\}$, in order to show that $\mathit{unfold}\,(k, n+1) \leftrightarrow_{R'} k$. □

Since $k \leftrightarrow \mathit{unfold}\,(k, n)$, it holds that $k \vDash f$ if and only if $\mathit{unfold}\,(k, f) \vDash f$, which is a standard property of bisimulation with respect to HML formulas [31].

In the overview of the synthesis process as illustrated in Figure 4.1, the unfolding step is the first step in the synthesis process. After synthesis has been applied, each resulting partial tree representation $k \in \mathcal{K}_{\rightarrow}$ is again converted into an LTS $g \in \mathcal{G}$. This is indicated as the post-synthesis step *Tree2LTS* in Figure 4.1. This function is provided below in Definition 4.13.

The intuitive explanation behind Definition 4.13 is as follows. Due to the fact that a single state-element $x \in X$ which occurs in a partial tree $k \in \mathcal{K}_{\rightarrow}$ may play different roles at various stages of synthesis, as indicated by x occurring as state-element in multiple parts of k, it is not possible to directly convert k into an LTS having a transition relation defined over X. Instead, a transition relation over the state space $X \times \mathbb{N}$ is defined, where the original transition relation is directly mapped to $X \times \{0\}$. If the top of the partial tree k is unfolded to depth n and if x is the top-most state-element of k, then a new transition relation is defined as as $(x, n) \xrightarrow{e_i} (x_i, n-1)$, if each x_i is the state-element of a sub-tree of k. This process is then continued recursively, as shown in Definition 4.13.

Definition 4.13. Let $k \in \mathcal{K}_{\rightarrow}$ and let $n \in \mathbb{N}$ be the greatest n such that $unf\,(k, n)$. A new LTS $g \in \mathcal{G}$ is constructed having state space $X \times \mathbb{N}$ if X is the state space of k. The initial state of g is then defined as (x, n) and its transition relation $\longrightarrow' \subseteq (X \times \mathbb{N}) \times \mathcal{E} \times (X \times \mathbb{N})$ is defined as:

$$\longrightarrow' = \{(x,0) \xrightarrow{e}' (x',0) \mid x \xrightarrow{e} x'\} \bigcup Tree2LTS\,(k,n)$$

where the function $Tree2LTS$ is defined in the following way:

$$\begin{array}{lcl} Tree2LTS\,(k,0) & = & \emptyset \\ Tree2LTS\,(k,n) & = & \displaystyle\bigcup_{k \xrightarrow{e} k'} Tree2LTS\,(k', n-1) \bigcup \{(x,n) \xrightarrow{e}' (x', n')\} \end{array}$$

In the last clause of this definition, x and x' are the respective initial states of k and k'. Note that in this particular situation, the labeling function L needs to be redefined into a new labeling function L' such that $L'(x, n) = L(x)$ for all $x \in X$ and $n \in \mathbb{N}$.

4.4 Operational Definition of Synthesis

The actual synthesis function $C : \mathcal{K}_{\rightarrow} \times \mathcal{F} \mapsto 2^{\mathcal{K}_{\rightarrow}}$ is defined inductively as a relation, expressed by means of deduction rules. Note that the function C is defined in such a way that it expects the first argument $k \in \mathcal{K}_{\rightarrow}$ to be *unfolded* to at least depth $depth\,(f)$. Since the synthesis function C does not modify the underlying transition relation, this part of the $\mathcal{K}_{\rightarrow}$ structure may be omitted in the following definition of C and the notations $\langle x \rangle$ and $\langle x, T \rangle$ are used instead of $\langle x \rangle_{\rightarrow}$ and $\langle x, T \rangle_{\rightarrow}$. Defining C by means of a relation allows better integration with an induction-style proof as is required to prove the validity of this synthesis construction. It also enables close resemblance with the corresponding definition via an inductive predicate in the Coq proofs, as will be considered later in more detail. Note that C is defined by the smallest relation established by these derivation rules; that is: $m \in C(k, f)$ if and only if this can be derived from Definition 4.14.

Definition 4.14. For $k \in \mathcal{K}_{\rightarrow}$ and $f \in \mathcal{F}$, the set of synthesis results $C(k, f)$ is defined by the following deduction rules for $x \in \mathcal{X}$, $p \in \mathcal{P}$, $T \subset \mathcal{E} \times \mathcal{K}$, $g \in \mathcal{F}$, $k', m \in \mathcal{K}$ and $e, e' \in \mathcal{E}$. Note that the set union operator $\cup$ is interpreted as a *disjoint* union for these rules.

$$\frac{}{k \in C(k, true)}\,[1] \qquad \frac{p \in L(x)}{\langle x \rangle \in C(\langle x \rangle, p)}\,[2] \qquad \frac{p \in L(x)}{\langle x, T \rangle \in C(\langle x, T \rangle, p)}\,[3]$$

$$\frac{p \notin L(x)}{\langle x \rangle \in C(\langle x \rangle, \neg p)}\,[4] \qquad \frac{p \notin L(x)}{\langle x, T \rangle \in C(\langle x, T \rangle, \neg p)}\,[5]$$

$$\frac{m \in C(k, f) \quad m \in C(k, g)}{m \in C(k, f \wedge g)}\,[6] \qquad \frac{k' \in C(k, f) \quad m \in C(k', g \wedge f)}{m \in C(k, f \wedge g)}\,[7]$$

$$\frac{m \in C(k, f)}{m \in C(k, f \vee g)}\,[8] \qquad \frac{m \in C(k, g)}{m \in C(k, f \vee g)}\,[9] \qquad \frac{}{\langle x, \emptyset \rangle \in C(\langle x, \emptyset \rangle, [e]\, f)}\,[10]$$

$$\frac{\langle x, T' \rangle \in C(\langle x, T \rangle, [e]\, f) \quad e \neq e'}{\langle x, \{(e', k)\} \cup T' \rangle \in C(\langle x, \{(e', k)\} \cup T \rangle, [e]\, f)}\,[11]$$

$$\frac{\langle x, T' \rangle \in C(\langle x, T \rangle, [e]\, f) \quad C(k, f) = \emptyset}{\langle x, T' \rangle \in C(\langle x, \{(e, k)\} \cup T \rangle, [e]\, f)}\,[12]$$

$$\frac{\langle x, T' \rangle \in C(\langle x, T \rangle, [e]\, f) \quad m \in C(k, f)}{\langle x, \{(e, m)\} \cup T' \rangle \in C(\langle x, \{(e, k)\} \cup T \rangle, [e]\, f)}\,[13]$$

$$\frac{m \in C(k, f)}{\langle x, \{(e, m), (e, k)\} \cup T \rangle \in C(\langle x, \{(e, k)\} \cup T \rangle, \langle e \rangle f)}\,[14]$$

The deduction rules for C in Definition 4.14 are briefly discussed here. Synthesis is neutral for $true$ as this formula is always satisfied (rule **1**). Synthesis for an atomic proposition p results in the same structure if p is valid in the initial state (i.e. $p \in L(x)$), as shown in rules **2** and **3**. Synthesis for the negated atomic proposition $\neg p$ results in the same structure if $p \notin L(x)$, as can be observed in rules **4** and **5**. The rules **6** and **7** define a fixpoint construction for the synthesis of a conjunction. The condition for termination as described in rule **6** applies when synthesis of both conjuncts results in the same structure. Otherwise, both conjuncts are synthesized alternatingly as shown in rule **7**. The rules **8** and **9** for disjunction are relatively straightforward: an ele-

ment of the synthesized set is a result of the synthesis for one of the disjuncts. The operator $[e]f$ is covered in rules **10**-**13** which are defined inductively on the set T. Rule **10** describes the basic case for this induction where no transitions to underlying structures are present and no modifications are required. Rule **11** details how an e'-transition, such that $e \neq e'$, is left in place for the operator $[e]f$, as the presence of this transition does not influence the satisfiability of an $[e]f$ formula. Rule **12** removes an e-transition for the operator $[e]f$ if no synthesis candidate can be found for this transition. The last rule **13** for $[e]f$ ensures that the original structure after an e-transition is replaced by an appropriate synthesis product, if possible. Finally, a single derivation rule **14** for the synthesis of the formula $<e>f$ is defined. A single witness for a proper e-transition is added to the original structure, which is left unmodified as far as the underlying system is concerned. Consideration of synthesis for $\langle x \rangle_{\rightarrow} \in \mathcal{K}_{\rightarrow}$ for the operators $[e]f$ and $<e>f$ is not required due to the unfoldedness assumption. This means that these cases are not included in the definition of C due to the fact that only the application of C to $\mathcal{K}$-elements which have been sufficiently unfolded needs to be taken into account.

4.5 Termination and Complexity

Three effective sequential steps are identified in the overview of the synthesis process, as seen in Figure 4.1. These are the unfolding step, the actual synthesis itself, and the *Tree2LTS* step. The synthesis step is illustrated here algorithmically, and also includes an analysis of its complexity, as well as the complexity of its preceding unfolding and succeeding *Tree2LTS* step. The synthesis algorithm is shown in Figure 4.5, under the assumption that its input parameter $k \in \mathcal{K}_{\rightarrow}$ is adequately unfolded up to $depth(f)$. The parameter H of the procedure *synthesis* in Figure 4.5 is used to guarantee termination of synthesis for conjunction.

As Figure 4.5 shows, the recursive structure of the synthesis algorithm follows the inductive structure of the HML formulas for all cases except for the case $f \equiv f_1 \wedge f_2$. Due to the fact that synthesis for conjunction might involve multiple invocations of synthesis for the same conjunct, this part of the algorithm is considered the dominating factor in the time-complexity of the algorithm. This case also complicates the termination proof significantly. It is shown that the synthesis algorithm is terminating in Theorem 4.1, followed by the complexity result in Theorem 4.2.

```
     procedure synthesis (k ∈ K_→, f ∈ F, set H of K_→)
       returns set of K_→
     begin
       set R of K_→ := ∅
 1     case (f = true)
         R := {k}
 2-3   case (f = p and (k = ⟨x⟩_→ or k = ⟨x, T⟩_→) and p ∈ L(x))
         R := {k}
 4-5   case (f = ¬p and (k = ⟨x⟩_→ or k = ⟨x, T⟩_→) and p ∉ L(x))
         R := {k}
 6-7   case (f = f_1 ∧ f_2)
         R := synthesis (k, f_1, ∅) ∩ synthesis (k, f_2, ∅)
         for each k' ∈ synthesis (k, f_1, ∅)\H
           R := R ∪ synthesis (k', f_2 ∧ f_1, H ∪ R)
 8-9   case (f = f_1 ∨ f_2)
         R := synthesis (k, f_1, ∅) ∪ synthesis (k, f_2, ∅)
 10    case (f = [e] f' and k = ⟨x, ∅⟩_→)
         R := {⟨x, ∅⟩_→}
 11    case (f = [e] f' and k = ⟨x, {(e', k')} ∪ T⟩_→ and e ≠ e')
         for each ⟨x, T'⟩_→ ∈ synthesis (⟨x, T⟩_→, [e] f', ∅)
           R := R ∪ {⟨x, {(e', k')} ∪ T'⟩_→}
 12    case (f = [e] f' and k = ⟨x, {(e, k')} ∪ T⟩_→ and
             synthesis (k', f', ∅) = ∅)
         R := synthesis (⟨x, T⟩_→, [e] f', ∅)
 13    case (f = [e] f' and k = ⟨x, {(e, k')} ∪ T⟩_→)
         for each ⟨x, T'⟩_→ ∈ synthesis (⟨x, T⟩_→, [e] f', ∅)
           for each m ∈ synthesis (k', f', ∅)
             R := R ∪ {⟨x, {(e, m)} ∪ T'⟩_→}
 14    case (f = <e>f' and k = ⟨x, T⟩_→)
         for each (e, k') ∈ T
           for each m ∈ synthesis (k', f', ∅)
             R := R ∪ {⟨x, T ∪ {(e, m)}⟩_→}
       return R
     end
```

Figure 4.5: Algorithmic representation of the synthesis procedure. This algorithm is a direct translation of the synthesis rules given in Definition 4.14. Corresponding rule numbers in Definition 4.14 are shown in the left column.

Theorem 4.1. For each $k \in \mathcal{K}_{\rightarrow}$ and $f \in \mathcal{F}$, the synthesis procedure in Figure 4.5 terminates in a finite number of steps.

Proof. It is first shown that only finitely many possible synthesis results are obtained using the procedure *synthesis*, by induction upon the structure of f. As considered earlier, the partial tree $k \in \mathcal{K}_{\rightarrow}$ is assumed to be finitely branching. Since the synthesis algorithm in Figure 4.5 can be considered a direct implementation of the synthesis rules in Definition 4.14, it might be helpful to the reader to consider these rules as well. For the cases $f \equiv \mathit{true}$, $f \equiv \mathit{false}$, $f \equiv p$, and $f \equiv \neg p$, for $p \in \mathcal{P}$, it is clear from Figure 4.5 that at most one result is returned. For the cases $f \equiv f_1 \wedge f_2$ and $f \equiv f_1 \vee f_2$, a finite number of synthesis results originates from a recursive call to the function *synthesis* for f_1 or f_2. If $f \equiv [e]\, f'$, then the recursive finite synthesis results are combined over a finite number of branches, resulting again in a finite number of results. If $f \equiv \langle e \rangle f'$ then, again, a finite number of results originate from the recursive call. Also, due to retaining existing behavior, a sub-tree may be added. However, note that sub-trees are not duplicated, since addition of $\{(e, k)\}$ to the set T results in T if $(e, k) \in T$. Therefore, synthesis for $\langle e \rangle f'$ also results in a finite number of synthesis outcomes.

It is now shown that an invocation of $\mathit{synthesis}\,(k, f, \emptyset)$ terminates in a finite number of steps, via induction on the structure of f. The cases where $f \equiv \mathit{true}$, $f \equiv \mathit{false}$, $f \equiv p$, and $f \equiv \neg p$, for $p \in \mathcal{P}$, do not result in any recursive calls, so the function synthesis will terminate directly for these cases. For the cases $f \equiv f_1 \vee f_2$, $f \equiv [e]\, f'$, and $f \equiv \langle e \rangle f'$, the procedure only invokes a finite number of terminating recursive calls, and termination is therefore obtained via induction. Termination for the remaining case for $f \equiv f_1 \wedge f_2$ is derived as follows. Via induction termination for the recursive calls to $\mathit{synthesis}\,(k, f_1, \emptyset)$, $\mathit{synthesis}\,(k, f_2, \emptyset)$ and $\mathit{synthesis}\,(m, f_1, \emptyset)$ is derived. Since, for each recursive invocation of $\mathit{synthesis}\,(m, f_2 \wedge f_1)$, the set H is extended with the set of synthesis results for f_1 and f_2, the recursive call to $\mathit{synthesis}\,(f_2 \wedge f_1)$ will, at some recursion depth, not be invoked, due to finiteness of the number of possible synthesis results □

The number of affected transitions during synthesis is limited by $\mathit{depth}\,(f)$, and may therefore be expressed as $n \cdot \mathit{depth}\,(f)$, where n is linear in the number of transitions. Based upon this observation, the upper bound for the number of solutions may be expressed as $2^{n \cdot \mathit{depth}\,(f)}$. This upper bound is also derived as the upper bound of the computational complexity of the algorithm in Theorem 4.2. Note that this represents a worst-case scenario. For instance, a formula without conjunction may be synthesized in $\Theta\,(n \cdot \mathit{depth}\,(f))$ steps.

Theorem 4.2. For $k \in \mathcal{K}_{\rightarrow}$ and $f \in \mathcal{F}$, the upper bound for the computational complexity of the procedure *synthesis* in Figure 4.5 is determined as $\Theta\left(2^{n \cdot depth\,(f)}\right)$, where n is linear in the number of transitions.

Proof. Induction is applied towards the structure of f. For the cases $f \equiv true$, $f \equiv false$, $f \equiv p$, and $f \equiv \neg p$, for $p \in \mathcal{P}$, the computational complexity can be stated as $\Theta\left(1\right) < \Theta\left(2^{n \cdot depth\,(f)}\right)$. If $f \equiv f_1 \vee f_2$, then synthesis invokes two recursive calls, as observed in Figure 4.5. For this case, complexity is therefore expressed as $2 \cdot \Theta\left(2^{n \cdot depth\,(f)}\right) \approx \Theta\left(2^{n \cdot depth\,(f)}\right)$. If $f \equiv [e]f'$ or $f \equiv \langle e \rangle f'$, then n/m for $1 < m < n$ recursive calls of the synthesis procedure are invoked. The factor m is taken linear in the number of e-transitions. For these cases, the upper bound for the computational complexity is determined as $(n/m) \cdot \Theta\left(2^{n \cdot depth\,(f)}\right) \approx \Theta\left(2^{n \cdot depth(f)}\right)$. The final case to consider is when $f \equiv f_1 \wedge f_2$. As the recursion depth for the synthesis invocation for $f_2 \wedge f_1$ is bounded by the number of possible synthesis results, computational complexity for this case may be expressed as $2 \cdot \Theta\left(2^{n \cdot depth\,(f)}\right) + 2^{n \cdot depth\,(f)} \cdot \Theta\left(2^{n \cdot depth\,(f)}\right) \approx \Theta\left(2^{n \cdot depth\,(f)}\right)$. □

Since the unfolding step only affects the LTS up to the depth of the synthesized formula, an actual realization of the function *unfold* in Definition 4.11 can be implemented in time $\Theta\left(depth\,(f)\right)$, for $f \in \mathcal{F}$. The *Tree2LTS* function, as given in Definition 4.13, only involves a single operation for every constructed transition. Since the size of the transition structure is taken as a linear factor, this term can effectively be ignored in determining the time-complexity of the algorithm. Hence, it may be concluded that the computational complexity of the entire synthesis process is dominated by, and therefore equivalent to, the complexity of the synthesis method itself. A final remark regarding the selection of the maximal candidates cannot be left unmentioned. As synthesis results in a set of satisfying structures, it would seem a natural part of such an algorithm to select the maximal candidates, among the synthesized results. Also, one might wonder why the selection of the maximal candidate is not considered in the analysis of the computational complexity of the algorithm. Multiple solutions arise due to a number of reasons. As Figure 2.4 clearly shows, multiple maximal results may be a result of the synthesis of a disjunctive formula. As indicated in Figure 2.4, these results are essentially incomparable, and therefore no selection is to be made. On the other hand, synthesis for a disjunction might result in multiple solutions of which a single maximal solution may be preferred, as shown in Figure 2.4. However, in the general case, it is not clear how this may be efficiently determined, compared to a direct computation of the maximal candidate, based

on the simulation preorder. Note that multiple results due to synthesis for a formula $<e>f$ do not pose a problem in this respect, as all original behavior is copied when synthesizing for $<e>f$, thus not invalidating maximality.

4.6 Validity

Two theorems are required regarding the validity of the definition of synthesis. In Theorem 4.3 it is shown that every synthesis result satisfies the synthesized formula. Theorem 4.4 details how every synthesis result is related via simulation to the original structure.

Theorem 4.3. For $f \in \mathcal{F}$ and $k, m \in \mathcal{K}_{\rightarrow}$ it holds that $m \in C(k, f)$ implies $m \vDash f$.

Proof. The proof is by induction to the construction of $m \in C(k, f)$, via the deduction rules in Definition 4.14. If $m \in C(k, true)$, then obviously $m \vDash true$. If $m \in C(k, p)$, for some $p \in \mathcal{P}$, then m and k have the same initial state, say x. Since $p \in L(x)$, this results in $m \vDash p$. If $m \in C(k, \neg p)$, then again it can be observed that m and k have the same initial state x, such that $p \notin L(x)$, and therefore $m \vDash \neg p$. For rules **6-7** the following analysis holds: If $m \in C(k, f_1)$ and $m \in C(k, f_2)$, then $m \vDash f_1 \wedge f_2$ by induction. For $k' \in C(k, f_1)$ and $m \in C(k', f_2 \wedge f_1)$, by induction and commutativity of the validity of $\wedge$, $m \vDash f_1 \wedge f_2$. For rules **8-9**, there are again two cases. If $m \in C(k, f_1)$ then $m \vDash f_1 \vee f_2$, and if $m \in C(k, f_2)$ then $m \vDash f_1 \vee f_2$, both by induction. There are four cases corresponding to the rules **10-13**. Trivially, it holds that $\langle x, \emptyset \rangle_{\rightarrow} \vDash [e] f'$. By induction, it holds that $\langle x, \{(e', k)\} \cup T' \rangle_{\rightarrow} \vDash [e] f'$ for each $e \neq e'$ if $\langle x, T' \rangle_{\rightarrow} \vDash [e] f'$. Rule **12** does not alter the structure of $m \in C(k, [e] f)$ and therefore preserves validity. If $\langle x, T' \rangle_{\rightarrow} \vDash [e] f$ and $m \vDash f$ for $m \in C(k, f)$, then it holds that $\langle x, \{(e, m)\} \cup T' \rangle_{\rightarrow} \vDash [e] f$. The last case corresponds to rule **14**. If there exists an $m \in C(k, f')$ and therefore $m \vDash f'$, then by induction $\langle x, \{(e, m), (e, k)\} \cup T \rangle_{\rightarrow} \vDash <e> f'$. □

Theorem 4.4. For $f \in \mathcal{F}$ and $k, m \in \mathcal{K}_{\rightarrow}$ it holds that $m \in C(k, f)$ implies $m \preceq k$.

Proof. The same proof strategy as in Theorem 4.3 is applied: induction to the construction of $m \in C(k, f)$. Note that only a proof sketch is given here, because no actual simulation witness relation is constructed. The cases for rules **1-5** and rule **10** are solved by reflexivity of simulation, while rules **6-9** are covered by induction and transitivity of simulation. The four remaining

cases consider the rules **11-14**. For rule **11**, it may be assumed that $\langle x, T' \rangle_{\rightarrow} \preceq \langle x, T \rangle_{\rightarrow}$ as the induction hypothesis. This directly leads to $\langle x, \{(e', k)\} \cup T' \rangle_{\rightarrow} \preceq \langle x, \{(e', k)\} \cup T \rangle_{\rightarrow}$. For rule **12** it holds that $\langle x, T' \rangle_{\rightarrow} \preceq \langle x, T \rangle_{\rightarrow}$ by induction and therefore $\langle x, T' \rangle_{\rightarrow} \preceq \langle x, \{(e, k)\} \cup T \rangle_{\rightarrow}$. For the case corresponding to rule **13** there are two induction hypotheses: $\langle x, T' \rangle_{\rightarrow} \preceq \langle x, T \rangle_{\rightarrow}$ and in addition, it holds that $m \preceq k$ for $m \in C(k, f)$. This leads to $\langle x, \{(e, m)\} \cup T' \rangle_{\rightarrow} \preceq \langle x, \{(e, k)\} \cup T \rangle_{\rightarrow}$. The proof is concluded by an analysis of the last rule **14**, for which it holds that $m \in C(k, f)$ and therefore $m \preceq k$ via the induction hypothesis. Clearly this leads to $\langle x, \{(e, m), (e, k)\} \cup T \rangle_{\rightarrow} \preceq \langle x, \{(e, k)\} \cup T \rangle_{\rightarrow}$. □

4.7 Maximality

As indicated before, it is desirable for products of synthesis to be modified to the least extent in order to achieve a maximal solution. This is especially required if further analysis is to be applied to the model, for instance if liveness is investigated, or if some kind of optimization procedure is applied post-synthesis. This maximality proof is shown in Theorem 4.5.

Lemma 4.2. For each $f \in \mathcal{F}$, $n \in \mathbb{N}$ and $k, m \in \mathcal{K}_{\rightarrow}$ such that $m \in C(k, f)$ and $unf(k, n)$ it holds that $unf(m, n)$.

Proof. Induction is applied to the construction of $m \in C(k, f)$. The four non-straightforward cases are the rules **11-14**. The first case is resolved under the induction hypothesis $unf(\langle x, T \rangle_{\rightarrow}, n) \Rightarrow unf(\langle x, T' \rangle_{\rightarrow}, n)$. Clearly the premise $unf(\langle x, \{(e', k)\} \cup T' \rangle_{\rightarrow}, n)$ leads to $unf(\langle x, \{(e', k)\} \cup T, \rangle_{\rightarrow}, n)$. Rule **12** does not alter $m \in C(k, \lfloor e \rfloor f)$ and therefore preserves unfoldedness, as shown by induction. For rule **13** there are two induction hypotheses: $unf(m, n)$ and $unf(\langle x, T \rangle_{\rightarrow}, n) \Rightarrow unf(\langle x, T' \rangle_{\rightarrow}, n)$. The premise $unf(\langle x, \{(e, k)\} \cup T \rangle_{\rightarrow}, n)$ immediately leads to the conclusion that $unf(\langle x, \{(e, m)\} \cup T' \rangle_{\rightarrow}, n)$. For rule **14** it holds that $unf(\langle x, T \rangle_{\rightarrow}, n)$, $unf(m, n)$ and therefore $unf(\langle x, \{(e, m)\} \cup T \rangle_{\rightarrow}, n)$ by induction. □

The maximality result follows in Theorem 4.5. If k' is a simulant of k such that $k' \vDash f$ and k is unfolded up to the depth of a formula f, then synthesis produces at least one result m such that $k' \preceq m$.

Theorem 4.5. For each $f \in \mathcal{F}$, $k' \in \mathcal{K}_{\rightsquigarrow}$ and $k \in \mathcal{K}_{\rightarrow}$ with $k' \vDash f$, $k' \preceq k$ and $unf(k, depth\,(f))$, there exists an $m \in C(k, f)$ such that $k' \preceq m$.

Proof. The proof is somewhat involved and relies upon induction towards the structure of f. For all cases, the induction premise for unfoldedness after synthesis is satisfied by Lemma 4.2. If $f \equiv true$ then $k \in C(k, true)$ by rule **1** in Definition 4.14 and clearly $k' \preceq k$. If $f \equiv false$ then this clearly contradicts the assumption that $k' \vDash f$. If $f \equiv p$ or $f \equiv \neg p$ for $p \in \mathcal{P}$ then $k \vDash f$, since strict equality on labels is assumed in Definition 4.8. Application of the corresponding rule **2-5** from Definition 4.14 results in $k \in C(k, f)$, while $k' \preceq k$ was already assumed.

The case for $f \equiv f_1 \wedge f_2$ is not straightforward. Observe that the following two induction hypotheses hold:

IHf1: For all $k' \preceq k$ such that $k' \vDash f_1$ and $unf(k, depth\,(f_1))$ there exists an $m \in C(k, f_1)$ such that $k' \preceq m$.

IHf2: For all $k' \preceq k$ such that $k' \vDash f_2$ and $unf(k, depth\,(f_2))$ there exists an $m \in C(k, f_2)$ such that $k' \preceq m$.

Using these induction hypotheses, an alternating application of synthesis for f_1 and f_2 of arbitrary length may be constructed:

$$k_1 \in C(k, f_1), k_2 \in C(k_1, f_2), k_3 \in C(k_2, f_1), \ldots, k_n \in C(k_{n-1}, f_2)$$

This sequence is obtained in the following way. As $k' \vDash f_1$, the induction hypothesis for IHf1 may be applied to obtain $k_1 \in C(k, f_1)$ and $k' \preceq k_1$. Also, it holds that $k' \vDash f_2$ which allows the application of IHf2 on k_1, resulting in $k_2 \in C(k_1, f_2)$ such that $k' \preceq k_2$. It is clear that this sequence of applications may be applied for an arbitrary number of times.

Assume that each $k_n \in C(k_{n-1}, f_i)$ for $i \in \{1, 2\}$ can be obtained using a finite derivation tree T_n. From Theorem 4.3, it follows that $k_n \vDash f_i$, so clearly the formulas f_1 and f_2, when considered separately, are not contradictory in themselves, since a synthesis result can be readily obtained for each of these conjuncts.

If there exists an $n \geq 1$ such that $k_n \in C(k_{n-1}, f_1)$ and $k_n \in C(k_{n-1}, f_2)$ then $k_n \in C(k_{n-1}, f_1 \wedge f_2)$ can be obtained by $n-1$ applications of rule **6** from Definition 4.14, followed by a single application of rule **5**.

Assume the operator $<e>$ is not contained in both f_1 and f_2, then each derivation tree T_n can be constructed using the rules **1-13** from Definition 4.14. Careful study of these rules shows that each rule either does not modify the model, or results in a synthesized product which has a strictly lower number of transitions. Therefore, in the restricted situation where only rules **1-13**

apply, it is always possible to obtain $n \in \mathbb{N}$ such that $k_n \in C(k_{n-1}, f_1 \wedge f_2)$, because otherwise the number of transitions would decrease below zero, which is clearly impossible.

Application of rule **14** complicates this situation, since this rule introduces additional transitions via copying of original behavior when synthesizing a formula $<e>f$. However, if $k \vDash <e>f$ then $k \in C(k, <e>f)$, which seems to justify the conclusion that no more than two applications of rule **14** are required in order to obtain a stable point. Nevertheless, there is still the possibility that following the application of rule **14**, rule **12** is applied to remove the just created witness for the formula $<e>f$ again. An example has been considered earlier in Figure 4.2.

However, the key observation in Figure 4.2 is the possibility to create a witness for an $<e>f$-formula at multiple points, of which can only be finitely many, in the general case. In detail, suppose that synthesis for a formula $<e>f$ results in $\langle x, \{(e,m), (e,k) \cup T\}\rangle$, via the application of rule **14**. Also, suppose that this (e,m) part of the model is subsequently removed by application of rule **12**. If, by the application of **14**, (e,m) is constructed again in a later synthesis step, then rule **12** was applied to synthesize a formula $[e]f'$ such that $C(m, f') = \emptyset$. This clearly indicates that the formulas $f_1 \wedge f_2$ are contradictory, which contradicts the $k' \vDash f_1 \wedge f_2$ assumption.

The next case to consider is when $f \equiv f_1 \vee f_2$. If $k' \vDash f_1$ then, by induction, there exists an $m \in C(k, f_1)$ such that $k' \preceq m$. This results in $m \in C(k, f_1 \vee f_2)$ by application of rule **7**. The case for $k' \vDash f_2$ is exactly symmetrical.

The next case for $f \equiv [e]f'$ again requires some careful analysis. Observe that the following induction hypothesis holds:

IHf1: For all $k' \preceq k$ such that $k' \vDash f'$ and $unf(k, depth(f'))$ there exists an $m \in C(k, f')$ such that $k' \preceq m$

It is clear that $k = \langle x, T\rangle$, since $k \neq \langle x\rangle$, because $unf(k, 1 + depth(f'))$. For each $k' \xrightarrow{e} k''$, it holds that $k'' \vDash f'$ and a corresponding $n \in \mathcal{K}$ such that $(e,n) \in T$ and $k'' \preceq n$. By induction, then $m' \in C(n, f')$. Repeated application of rules **11**-**13**, and a single application of rule **10**, allows the construction of a set $U \subset \mathcal{E} \times \mathcal{K}$ such that $\langle x, U\rangle \in C(\langle x, T\rangle, [e]f')$ and $k' \preceq \langle x, U\rangle$.

The remaining case is when $f \equiv <e>f'$. As there exists a $k' \xrightarrow{e} k''$ such that $k'' \vDash f'$, there exists a corresponding $(e,n) \in T$ such that $k'' \preceq n$. Application of the induction hypothesis then results in $m \in C(n, f')$. By application of rule **14**, $\langle x, \{(e,m), (e,n)\} \cup T\rangle \in C(\langle x, \{(e,n)\} \cup T, <e>f'\rangle$ can be obtained. The simulation requirement $k' \preceq \langle x, \{(e,m), (e,n)\} \cup T\rangle$ is satisfied because original behavior is retained by rule **14**. □

4.8 Computer Verified Proofs

For Theorems 4.3 and 4.4, computer verified proofs have been constructed using the Coq proof assistant [13]. A number of remarks should be made with regard to these formal proofs. First and foremost, it is not possible to encode the dependent type $\mathcal{K}$, as given in Definition 4.7, directly in Coq. Due to the strict positivity requirement of the inductive types in Coq, it is not possible to define the collection of underlying tree-elements as a set. Instead, a list is used, which has some implications for the definition of equality on $\mathcal{K}$ due to the occurrence of multiple equal elements. Strict positivity for inductive types also implies that rule **12** cannot be encoded precisely in Coq, since no test on emptiness of the type can be used during its definition. The implication is that a broader set of synthesis results is constructed. Still, each result satisfies the aforementioned two Theorems 4.3 and 4.4, and every synthesis result as constructed by Definition 4.14 is still present. Unfortunately, these peculiarities make it impossible to encode the full maximality proof, as shown in Theorem 4.5, in the Coq proof assistant.

A short overview is presented here for the computer verified proofs which support the synthesis construction in this chapter. Initial remarks about some of the applied methodologies within the Coq proof assistant have been discussed in Chapter 3. Therefore, these computer verified proofs are considered here in more compact form. The first part of the proof concerns parameters to the theory defined for this synthesis construction. Note that in contrast to the computer verified proof in the previous chapter, global assumptions for the state space, labeling function and transition relation are applied. In this case, this results in a significant simplification of the proof construction, while this at the same time does not result in loss of generality. These assumptions to the theory are possible in this case because all applied transition relations are of the same type. These initial proof-theoretic parameters are listed in the Coq code below.

```
Parameter P E X : Set.
Parameter L : X -> P -> Prop.
Parameter step : X -> E -> X -> Prop.
```

The next step in the Coq proof for the synthesis construction concerned in this chapter involves the definition of the structure `K` and predicates for retrieving the initial state and to test whether a transition exists. The structure `K` is defined as an inductive type, but Coq is not able to automatically derive an induction hypothesis in the `tree` case. This is due to the fact that the `K`-structure is related indirectly via a `list` type. It is worth mentioning that

this inability to automatically derive an induction hypothesis is *not* due to the fact that this list type is defined over pairs of `E` and `K`. The `init` and `trans` predicates are defined as shown in the Coq listing below.

```
Inductive K :=
  | node : X -> K
  | tree : X -> list (E * K) -> K.

Definition init (k : K) :=
  match k with
  | node x => x
  | tree x T => x
  end.

Inductive trans : K -> E -> K -> Prop :=
  | trans_node : forall x e x', trans (node x) e (node x')
  | trans_tree : forall x T e k, In (e, k) T ->
    trans (tree x T) e k.
```

The predicates for retrieving the initial state and for testing whether a transition exists are henceforth applied to define simulation between two structures of type `K`. Note that in comparison to the definition of partial bisimilarity applied in the previous chapter, simulation is defined here on entire structures of type `K`. Its Coq definition is listed below:

```
Definition sim (k' k : K) : Prop :=
  exists R : K -> K -> Prop, R k' k /\
  forall m' m, R m' m ->
    (forall p, L (init m') p <-> L (init m) p) /\
    (forall e n', trans m' e n' ->
      exists n, trans m e n /\ R n' n).
```

The next two elements in the proof are the definition of the formulas for HML and the validity of such formulas with regard to structures of type `K`. These definitions are very similar to the corresponding Coq code in Section 3.4. What then follows is the key definition of the synthesis construction as applied in this chapter. As discussed in the previous theoretical sections, the synthesis steps only apply to the tree-like structure. This results in an inductive predicate `C`, which is encoded in Coq as shown in the listing below. This type of definition actually encodes the characteristic function for a relation over $\mathcal{K} \times \mathcal{F} \times \mathcal{K}$ and thereby defines a function of type $\mathcal{K} \times \mathcal{F} \mapsto 2^{\mathcal{K}}$. Under this interpretation, the Coq code is quite close to an encoding of the algorithm in Figure 4.5 in formal mathematics. The disjoint union operator applied in

Definition 4.14 corresponds to the pattern matching over list-elements in the Coq code which expresses the C function, as shown below:

```
Inductive C : K -> F -> K -> Prop :=
  | cr_true : forall k, C k true k
  | cr_prop : forall k p, L (init k) p -> C k (prop p) k
  | cr_not : forall k p, ~L (init k) p -> C k (not p) k
  | cr_and_base : forall k m f g, C k f m ->
    C k g m -> C k (and f g) m
  | cr_and_ind : forall k k' m f g, C k f k' ->
    C k' (and g f) m -> C k (and f g) m
  | cr_or_left : forall k f g m, C k f m ->
    C k (or f g) m
  | cr_or_right : forall k f g m, C k g m ->
    C k (or f g) m
  | cr_all_nil : forall x e f,
    C (tree x nil) (all e f) (tree x nil)
  | cr_all_skip : forall x T T' k e e' f, e' <> e ->
    C (tree x T) (all e f) (tree x T') ->
    C (tree x ((e', k) :: T)) (all e f)
      (tree x ((e', k) :: T'))
  | cr_all_ind : forall x T T' k m e f, C k f m ->
    C (tree x T) (all e f) (tree x T') ->
    C (tree x ((e, k) :: T)) (all e f)
      (tree x ((e, m) :: T'))
  | cr_all_rem : forall x T T' k e f,
    C (tree x T) (all e f) (tree x T') ->
    C (tree x ((e, k) :: T)) (all e f) (tree x T')
  | cr_ex_skip : forall x T T' k e e' f,
    C (tree x T) (ex e f) (tree x T') ->
    C (tree x ((e', k) :: T)) (ex e f)
      (tree x ((e', k) :: T'))
  | cr_ex_ind : forall x T k m e f, C k f m ->
    C (tree x T) (ex e f) (tree x T) ->
    C (tree x ((e, k) :: T)) (ex e f)
      (tree x ((e, m) :: (e, k) :: T)).
```

The two proofs of the synthesis construction which are encoded in Coq are then formalized as shown below. A somewhat remarkable observation regarding this proof is that the complexity of the formalization of this synthesis theory lays for the most part in the formal definitions, rather than in the actual proofs themselves. For this reason, the Coq code for the first proof can be included in its entirety in the first listing below. The proof is then quite eas-

ily constructed by induction towards the structure of the `f` variable, followed by an attempt to resolve most inductive cases immediately via application of the **tauto** tactic.

```
Theorem validity : forall f k m, C k f m -> val m f.
Proof.
  intros f k m H ; induction H ; simpl in * ; try tauto.
  intros k' H ; inversion H ; simpl in * ; contradiction.
  intros k' H' ; inversion H' ; simpl in *.
  destruct H5 as [ Heq | HinT' ] ;
    [ inversion Heq ; contradiction | ].
  apply IHC ; apply trans_tree ; auto.
  intros k' H' ; inversion H' ; simpl in *.
  destruct H5 as [ Heq | HinT' ] ; [ inversion Heq | ].
  rewrite <- H6 ; auto.
  apply IHC2 ; apply trans_tree ; auto.
  destruct IHC as [ k' [ Htrans Hval ] ].
  exists k' ; split ; auto.
  inversion Htrans ; apply trans_tree ;
    simpl in * ; auto.
  exists m ; split ; auto.
  apply trans_tree ; simpl ; auto.
Qed.
```

The simulation proof is included below and is created via the construction of a characteristic function for the simulation relation.

```
Theorem simulation : forall f k m, C k f m -> sim m k.
Proof.
  assert (forall x T T' e m k, sim (tree x T') (tree x T) ->
    sim m k -> sim (tree x ((e, m) :: T'))
                   (tree x ((e, k) :: T))) as Hadd.
  intros x T T' e m k H H' ;
    destruct H as [ R [ HinR HrelR ] ].
  destruct H' as [ R' [ HinR' HrelR' ] ].
  exists (fun p q => p = (tree x ((e, m) :: T')) /\
    q = (tree x ((e, k) :: T)) \/ R p q \/ R' p q) ;
      split ; auto.
            ⋮
  (some lines of Coq code omitted)
            ⋮
Qed.
```

4.9 Closing Remarks

In the research presented in this chapter, the synthesis for HML on Kripke-structures with labeled transitions is detailed. A bisimilarity preserving transformation is applied to transform an LTS into an equivalent partial tree representation, which is able to capture an embedded unfolding. Upon this structure, operational rules define the required modifications in order to satisfy the synthesized HML formula. Results in the set of synthesized models are shown to be valid in terms of satisfying the given HML formula, and simulation of the original input LTS. A maximal solution with regard to the simulation preorder is shown to be contained in this set. A significant part of definitions and proofs are computer verified, which contributes to the understanding and assessment of the validity of the proposed theory. The maximality result for all non-deterministic simulants in proves a key property of the synthesis method: the least number of modifications is applied in order to satisfy the synthesized formula. Note that this is an improvement in [41], compared to the work in [40].

Chapter 5

Control Theory and Process Algebra

In this chapter a process-theoretic concurrency model is proposed which is then used to express control synthesis. A process-theoretic [8] expression of plant and controller provides an adequate level of abstraction as well as formal precision. Process theories as described in [8] provide a formal description of discrete event behavior, including termination. Furthermore, communication of state/event observations and control signals, as well as restrictions on behavior can all be expressed in such theories. Further information regarding process theories with propositional signals can be found in [11]. The main purpose of this chapter, within the context of this thesis, is to explore the expression of the earlier defined partial bisimulation preorder using a different formalism and to work towards a subsequent analysis of the first case study considered in Section 3.5.

The process algebra TCP* is studied as a convenient modeling formalism which includes parallelism, iteration and communication features and is able to express non-determinism. The theory TCP* is employed in this chapter as an alternative formalism for modeling structures which describe behavior, compared to the transition systems defined in Chapter 2 and Chapter 4. Like in earlier chapters, the partial bisimilarity preorder is applied to define the relationship between plant and supervisor. This requires an adaptation of Definition 2.6 in terms of process-algebraic notions. It is shown how the precongruence property of partial bisimilarity can be derived from the format of the deduction rules. The partial bisimilarity refinement may be used to express

controllability in the context of process algebra as well. A case for automated guided vehicles (AGV) is modeled using the theory TCP*. However, a different approach needs to be applied to address the issue of non-deterministic plant models. The aforementioned process theory TCP* is extended by constructs for state observation in order to express state based control in a non-deterministic context. Process-theoretic expressions are paired with Boolean valuations which contain information regarding the current state of the plant as propositions. In order to model the various constructs involved in supervisory theory, conditional expressions are employed to model event inhibition, based upon the evaluation of a guarding formula within the Boolean valuation that contains the state information. The industrial printer case from Section 3.5 is then modeled using this extended theory.

5.1 The Process Theory TCP*

In this section the process theory TCP* is presented, which requires some detail when considering its rich syntax. Its various constructs enable it to model a variety of problems in a clear way. Elements of this theory can be used to model the different components in the supervisory control setup, where communicating actions represent the information flow between components, thereby completing the model of the control loop. Synchronizing actions are used to model allowance or denial of plant behavior by the supervisory controller.

A number of preliminary notions in language theory and process algebra are first introduced here. These are required to lay the foundations of TCP*. A finite data alphabet $\mathcal{D}$ as well as a finite set $\mathcal{H}$ of communication channels are assumed as preliminaries for this theory. For each $c \in \mathcal{H}$, the set $\mathcal{A}_c$ is defined as shown below, where $c!_m?_n d$ represents a generic communication action [8] consisting of m send actions and n receive actions.

$$\mathcal{A}_c = \{c!_m?_n d \mid m, n \in \mathbb{N},\ m + n > 0,\ d \in \mathcal{D}\}$$

In addition, the following abbreviated notations are used: $c?d$ for $c!_0?_1 d$, $c!d$ for $c!_1?_0 d$ and $c!?d$ for $c!_1?_1 d$. Intuitively, these events denote respectively that data element d is received, sent or communicated along channel c. Furthermore,

$$\mathcal{A} = \bigcup_{c \in \mathcal{H}} \mathcal{A}_c$$

denotes the entire set of actions. Definition 5.1 is applied to denote all actions relying on the same communication channel:

Definition 5.1. The notation $B \subseteq_{\mathcal{H}} \mathcal{A}$ is used to indicate that there exists an $\mathcal{H}' \subseteq \mathcal{H}$ such that $B = \cup_{c \in \mathcal{H}'} \mathcal{A}_c$

This notation will be convenient when arbitrary subsets of $\mathcal{A}$ need to be handled which have to contain all communications sent over a number of channels. Traces of events $(a_0, a_1, \ldots, a_n) \in \mathcal{A}^*$ are formed in a standard manner, where $\mathcal{A}^* = \{(a_0, a_1, \ldots, a_n) \mid n \in \mathbb{N}\}$. The notation ϵ is used to denote the unique empty trace. If $t_a = (a_0, a_1, \ldots, a_m)$ and $t_b = (b_0, b_1, \ldots, b_n)$, then $t_a \cdot t_b = (a_0, a_1, \ldots, a_m, b_0, b_1, \ldots, b_n)$ denotes the concatenation of traces t_a and t_b. As in previous chapters, the set of events $\mathcal{A} = \mathcal{C} \cup \mathcal{U}$ is strictly partitioned into controllable events $\mathcal{C}$ and uncontrollable events $\mathcal{U}$. The foundations have now been laid to properly define the actual elements of the process theory TCP*, as shown in Definition 5.2.

Definition 5.2. The set of terms $\mathcal{T}$ of the process theory TCP* is generated by the grammar shown below. Assume that $E \subseteq \{c!_m?_n \mid c \in \mathcal{C}, m, n \in \mathbb{N}\}$ in the following definition:

$$\mathcal{T} ::= 0 \mid 1 \mid \mathcal{A} \mid \mathcal{T} + \mathcal{T} \mid \mathcal{T} \cdot \mathcal{T} \mid \mathcal{T}^* \mid \mathcal{T} \parallel \mathcal{T} \mid \partial_E(\mathcal{T})$$

The various elements of $\mathcal{T}$ are considered briefly here. The constant process 0 denotes inaction or deadlock. The constant process 1 denotes successful termination. For each action $a \in \mathcal{A}$, the process corresponding to the term a executes the action a, followed by successful termination. The expression $\mathcal{T} + \mathcal{T}$ denotes alternative composition. The process term $p + q$, for $p, q \in \mathcal{T}$, expresses a non-deterministic choice for a process that can either behave as p or as q. The sequential composition operator $\mathcal{T} \cdot \mathcal{T}$ first executes the left-hand side process and then, upon successful termination of this operand, executes the right-hand side process. The binary operator $\mathcal{T} \parallel \mathcal{T}$ denotes a parallel composition of two terms that is able to perform interleaving as well as synchronous communication. The term $p \parallel q$, for $p, q \in \mathcal{T}$, can behave as 1) a unilateral step of either p or q, while the other operand remains unchanged, or 2) a synchronous communication step in both p and q, upon which data is communicated over a specified channel. The operator $\mathcal{T}^*$ or *Kleene star* is used to express iteration. It unfolds with respect to sequential composition. The term p^*, for $p \in \mathcal{T}$, either terminates or behaves as p, followed by p^*. The unary operator $\partial_E(\mathcal{T})$ encapsulates a process p in such a way that all (incomplete) communication actions (e.g. $c?d$ and $c!d$) are blocked for all data,

so that bilateral communication is enforced. An example might be illustrative in this regard. If communication between k processes on channel c needs to be enforced, then E needs to be chosen in the following way:

$$E = \{c!_m?_n \mid 0 < m + n < k, c \in \mathcal{C}\}$$

If E is chosen as indicated, it includes all generic communication actions (excluding data) in such a way that it becomes possible to communicate between at most k process terms.

Structural operational semantics for each process term $p \in \mathcal{T}$ now needs to be defined in order to formalize how the terms in $\mathcal{T}$ express behavior. This relates to two predicates: a *transition relation* $\longrightarrow \subseteq \mathcal{T} \times \mathcal{A} \times \mathcal{T}$ and a *termination* property $\downarrow \subseteq \mathcal{T}$. The transition relation $\longrightarrow$ defines possibly non-deterministic steps between process terms, which are eventually used to model process dynamics within the plant. Terminating process terms define process end points, which model final or completed tasks in the plant. Operational rules are given in Definition 5.3. Infix and postfix notation is applied, such that $p \xrightarrow{a} p'$ denotes that $(p, a, p') \in \longrightarrow$ and $p\downarrow$ denotes that $p \in \downarrow$.

Definition 5.3. A step relation $\longrightarrow \subseteq \mathcal{T} \times \mathcal{A} \times \mathcal{T}$ and a termination predicate $\downarrow \subseteq \mathcal{T}$ are defined as shown below. Assume that $p, q, p', q' \in \mathcal{T}$, $a \in \mathcal{A}$, $c \in \mathcal{H}$ and $d \in \mathcal{D}$ in the following set of derivation rules:

$$\frac{}{1\downarrow} \qquad \frac{p\downarrow}{p+q\downarrow} \qquad \frac{q\downarrow}{p+q\downarrow} \qquad \frac{p\downarrow \quad q\downarrow}{p\cdot q\downarrow} \qquad \frac{}{p^*\downarrow} \qquad \frac{p\downarrow \quad q\downarrow}{p \parallel q\downarrow}$$

$$\frac{p\downarrow}{\partial_E(p)\downarrow} \qquad \frac{}{a \xrightarrow{a} 1} \qquad \frac{p \xrightarrow{a} p'}{p+q \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{a} q'}{p+q \xrightarrow{a} q'} \qquad \frac{p \xrightarrow{a} p'}{p\cdot q \xrightarrow{a} p'\cdot q}$$

$$\frac{p\downarrow \quad q \xrightarrow{a} q'}{p\cdot q \xrightarrow{a} q'} \qquad \frac{p \xrightarrow{a} p'}{p^* \xrightarrow{a} p'\cdot p^*} \qquad \frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q} \qquad \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'}$$

$$\frac{p \xrightarrow{c!_l?_k d} p' \quad q \xrightarrow{c!_m?_n d} q'}{p \parallel q \xrightarrow{c!_{l+m}?_{k+n} d} p' \parallel q'} \qquad \frac{p \xrightarrow{a} p' \quad a \notin \{c!_m?_n d \mid c!_m?_n \in E, d \in \mathcal{D}\}}{\partial_E(p) \xrightarrow{a} \partial_E(p')}$$

The operational rules in Definition 5.3 are briefly discussed here. The first rule states that the constant process 1 can terminate. The subsequent two rules indicate that if one operand of an alternative composition can terminate

then the entire expression has the termination property. In contrast, for the termination of a sequential composition it is required that both operands are able to terminate. Any term under iteration can terminate, which intuitively corresponds to iterating zero times. Similar to sequential composition, for a parallel composition to be able to terminate it is required that both operands can terminate. Termination of an encapsulated term depends upon termination of the term itself.

The first rule for the definition of the step relation states that any action induces an outgoing transition having the same label. The two rules for alternative composition enable a non-deterministic choice between the steps offered by the two respective operands. The succeeding two rules semantically define how the sequential composition behaves: 1) The left operand may be able to terminate in which case a continuation may be provided by the right operand, or 2) the left side may take an independent step. The Kleene star unfolds with respect to sequential composition. It executes an underlying step and may eventually continue executing itself again. The parallel composition operator relies on three different rules. Two of these rules express unilateral behavior, while the last rule for parallel composition expresses lock-step behavior which includes communication. The last rule in Definition 5.3 defines behavior of an encapsulated term, which depends upon the set of encapsulated actions. The work in [8] and in [11] contains more information about the operators in $\mathcal{T}$ and their precise semantics.

One of the fundamental requirements of the control loop has been highlighted various times before in this thesis: a supervisor cannot disallow an uncontrollable event [76]. The solution via partial bisimilarity was considered earlier on in this thesis and formally defined in terms of transition systems in Definition 2.6. This solution needs to be adapted in order to accommodate a behavioral description by process terms. This adapted variant of partial bisimulation can be found in Definition 5.4.

Definition 5.4. Let R be a relation on $\mathcal{T}$. Then R is a partial bisimulation with respect to the bisimulation action set $B \subseteq_{\mathcal{H}} \mathcal{A}$ if for all $p, q \in \mathcal{T}$ such that $(p, q) \in R$ the following holds:

1. $p \downarrow$ if and only if $q \downarrow$;
2. for all $p' \in \mathcal{T}$ and $a \in \mathcal{A}$ such that $p \xrightarrow{a} p'$, there exists a $q' \in \mathcal{T}$ such that $q \xrightarrow{a} q'$ and $(p', q') \in R$; and
3. for all $q' \in \mathcal{T}$ and $b \in B$ such that $q \xrightarrow{b} q'$, there exists a $p' \in \mathcal{T}$ such that $p \xrightarrow{b} p'$ and $(p', q') \in R$.

Process term p is defined to be *partially bisimilar* to q with respect to the bisimulation action set B (notations: $p \preceq q$ and $p \preceq^B q$), if there exists a partial bisimulation R, in terms of B, such that $(p, q) \in R$. If the partial bisimulation set R is of particular relevance, the notation $p \preceq_R^B$ will be applied.

It can be easily shown that partial bisimilarity is a preorder relation [77]. Also, it is not difficult to prove that mutual partial bisimilarity is an equivalence relation [77]. Note that if the bisimulation set B is empty, then the partial bisimilarity preorder coincides with the standard (strong) similarity preorder. When $B = \mathcal{A}$, the partial bisimilarity preorder becomes strong bisimilarity [31]. Lemma 5.1 proves an important transitivity property for dependence upon the bisimulation action set B.

Lemma 5.1. If $p \preceq^B q$, then $p \preceq^C q$ for every $C \subseteq B$.

Proof. Let $p \preceq^B q$ be given by R. If for an arbitrary $(p, q) \in R$ it holds that $q \xrightarrow{c} q'$ for some $c \in C \subseteq B$ and $q' \in \mathcal{T}$, then there exists a $p' \in \mathcal{T}$ such that $p \xrightarrow{c} p'$ and $(p', q') \in R$. The remaining conditions for partial bisimilarity remain unaltered and it may therefore be concluded that R is also a partial bisimulation such that $p \preceq^C q$. □

The remainder of this section considers a proof in which it is shown that partial bisimilarity $\preceq^B$ for $B \subseteq_{\mathcal{H}} \mathcal{A}$ is a precongruence with respect to the operators of TCP*. In [66], it is shown that for operational rules in the *tyft* format, congruence with respect to bisimilarity can be automatically derived. This precongruence proof is set up in a relatively general way. The notation $C(\mathcal{T})$ is used to denote the closed terms in $\mathcal{T}$. Two definitions are adapted from [66]:

Definition 5.5. An operational rule is in *tyft* format if it is of the form defined below. In this definition, I is a set of indices, $a \in \mathcal{A}$, f is an n-ary operator in the process theory TCP*, $t' \in \mathcal{T}$ and $x_0, \ldots, x_{n-1}, y_i$ are all distinct process variables. Furthermore, for all $i \in I$, it holds that $a_i \in \mathcal{A}$ and $t_i \in \mathcal{T}$ in the following definition:

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_0, \ldots, x_{n-1}) \xrightarrow{a} t'}$$

In addition, the abbreviation $X_p = \{x_0, \ldots, x_{n-1}\}$ is used to denote the set of process variables in the source of the conclusion. and $Y_p = \{y_i \mid y \in I\}$

denotes the set of variables in the target of the premises. As a necessary requirement, it is assumed that the sets X_p and Y_p are disjoint. A set of derivation rules conforms to the *tyft* format if all rules adhere to the *tyft* format. The formalization of the prerequisites continues with the definition of the *acyclicity* of the variable dependency graph. These variable dependency graphs are defined in the following way:

Definition 5.6. For every deduction rule depending on premises $P_1, \ldots, P_N$ with their respective sets of process variables S_i in the source of the premise i and T_i in the target of the premise i, a variable dependency graph is defined in the following way:

1. Every variable in $\cup_i (S_i \cup T_i)$ is a node; and
2. There exists an edge (v_s, v_t) if there exists an i such that $v_s \in S_i$ and $v_t \in T_i$.

This graph is defined to be *acyclic* if it does not contain any cycles. The *rank* (x) is defined for each process variable x as the maximum length of a backward chain starting in x in the variable dependency graph. The rank of a premise is the rank of its target variable.

A subsequent definition which is required is the closure of a relation under precongruence:

Definition 5.7. Let $R \subseteq C(\mathcal{T}) \times C(\mathcal{T})$. The relation $\tilde{R} \subseteq C(\mathcal{T}) \times C(\mathcal{T})$ is defined to be the smallest reflexive precongruence on $C(\mathcal{T})$ such that the relation R is contained in $\tilde{R}$. The relation $\tilde{R}$ can be formally specified as follows:

1. $\tilde{R}$ is reflexive;
2. $R \subseteq \tilde{R}$;
3. $(f(p_0, \ldots, p_{n-1}), f(q_0, \ldots, q_{n-1})) \in \tilde{R}$ for every n-ary $f \in \mathcal{T}$, and all $p_0, \ldots, p_{n-1}, q_0, \ldots q_{n-1} \in C(\mathcal{T})$ such that $(p_i, q_i) \in \tilde{R}$ for $0 \leq i < n$.

Lemma 5.2. Let $R \subseteq C(\mathcal{T}) \times C(\mathcal{T})$ and $t \in \mathcal{T}$. For any two process substitutions σ and σ' such that $(\sigma(x), \sigma'(x)) \in \tilde{R}$, where x is a process variable in t, it holds that $(\sigma(t), \sigma'(t)) \in \tilde{R}$.

Proof. By induction towards the structure of the process term t. See [33]. □

Lemma 5.3. Partial bisimilarity is a precongruence for each of the operators in $\mathcal{T}$.

Proof. It may be straightforwardly observed from Definition 5.3 that each of these derivation rules adheres to the aforementioned *tyft* format. Therefore, the now following adaptation of the proof in [66] for bisimilarity is sufficient.

Let f be an n-ary process function and let p_i, q_i be closed process terms for $0 \leq i < n$. Suppose that $B \subseteq_{\mathcal{H}} \mathcal{A}$ and $p_i \preceq^B q_i$ for $0 \leq i < n$. This means that for every $0 \leq i < n$ there exists a partial bisimulation relation R_i that witnesses these partial bisimilarities. Let $R = \cup_{i=0}^{n-1} R_i$ be the union of these relations. It is quite obvious that R is a partial bisimulation, with respect to B, as well. It is now sufficient to show that the relation $\tilde{R}$ contains the pair $(f(p_0, \ldots p_{n-1}), f(q_0, \ldots, q_{n-1}))$ and that it adheres to the partial bisimulation property as well. The first claim follows directly from the definition of $\tilde{R}$.

In [66] it is shown for $(p, q) \in \tilde{R}$ that for $a \in \mathcal{A}, p' \in C(\mathcal{T})$ such that $p \xrightarrow{a} p'$, there exists a $q' \in C(\mathcal{T})$ such that $q \xrightarrow{a} q'$ and $(p', q') \in \tilde{R}$. This proof is completed here for partial bisimilarity by showing that for $b \in B$ such that $q \xrightarrow{b} q'$, there exists a $p' \in C(\mathcal{T})$ such that $p \xrightarrow{b} p'$ and $(p', q') \in \tilde{R}$.

The proof proceeds by induction towards the depth of the derivation of a transition. The proof for the induction base is omitted because it is a direct instance of the proof of the induction step where there are no premises.

For the induction step a distinction needs to be made between three cases, based on the definition of $\tilde{R}$. In case $(p, q) \in \tilde{R}$, due to reflexivity or due to $(p, q) \in R \subseteq \tilde{R}$, the result follows directly and no inductive reasoning is required. For the remaining case, $p = f(p_0, \ldots, p_{n-1})$ and $q = f(q_0, \ldots, q_{n-1})$ for some $p_0, \ldots, p_{n-1}, q_0, \ldots, q_{n-1}$ such that $(p_i, q_i) \in \tilde{R}$ for all $0 \leq i < n$, the format of the last derivation needs to be understood. The last step in the deduction tree for the transition of q is due to the application of a derivation rule of the following form:

$$\frac{\{t_i \xrightarrow{b_i} y_i \mid i \in I\}}{f(x_0, \ldots, x_{n-1}) \xrightarrow{b} t'}$$

To prove the result for partial bisimilarity, an additional condition for every deduction rule is required: If $b \in B$ then $\forall i \in I\, b_i \in B$. This means that there exists a process substitution σ such that $\sigma(x_i) = q_i$ for all $0 \leq i < n$ and $\sigma'(t') = q'$. Furthermore, for each $i \in I$ there exists a derivation of $\sigma(t_i) \xrightarrow{b_i} \sigma(y_i)$ with smaller depth. For each process variable x in the vari-

ables of t_i of each premise $t_i \xrightarrow{a_i} y_i$, it holds that $\mathit{rank}(x) < \mathit{rank}(y_i)$. The process substitution σ' can be defined as follows:

$$\sigma'(x) = \begin{cases} q_i & \text{if } x = x_i, \\ \sigma(x) & \text{if } x \notin X_p \cup Y_p. \end{cases}$$

Note that this process substitution remains to be defined for variables in Y_p. This definition will be extended in the remainder of this proof. For all $i \in I$ such that $r_i = \mathit{rank}(P_i) = \mathit{rank}(y_i)$ of premise P_i, three essential properties are shown here:

A. $(\sigma(t_i), \sigma'(t_i)) \in \tilde{R}$; and

B. $\sigma'(t_i) \xrightarrow{b_i} \sigma'(y_i)$; and

C. $(\sigma(y_i), \sigma'(y_i)) \in \tilde{R}$.

Again, the proof of the induction base ($r_i = 0$) is not shown here, as it is an instance of the proof of the induction step. For the inductive part, assume that $r_i \geq 1$. Let $t_i \xrightarrow{b_i} y_i$ for some $i \in I$ be a premise of rank r_i. First, property (A) is shown. Let x be a variable in t_i, and distinguish between the following cases:

1. If $x \in X_p$ then $x = x_i$ for some $0 \leq i < n$. From the definition of σ' it holds that $\sigma(x) = \sigma(x_i) = q_i$ and $\sigma'(x_i) = p_i$ and, as $(p_i, q_i) \in \tilde{R}$, it holds that $(\sigma(x), \sigma'(x)) \in \tilde{R}$.

2. If $x \notin X_p$ and $x \notin Y_p$ then it holds that $\sigma(x) = \sigma'(x)$. Since identity is included in $\tilde{R}$, it directly follows that $(\sigma(x), \sigma'(x)) \in \tilde{R}$.

3. If $x \in Y_p$ then $x = y_j$ for some $j \in I$. Because in this case $\mathit{rank}(y_j) < \mathit{rank}(y_i)$, so the induction hypothesis gives rise to: $(\sigma(y_j), \sigma'(y_j)) \in \tilde{R}$. Moreover, as $x = y_j$, it also holds that: $(\sigma(x), \sigma'(x)) \in \tilde{R}$.

Because of the fact that $(\sigma(x), \sigma'(x)) \in \tilde{R}$ for all variables x in t_i, it holds that $(\sigma(t_i), \sigma'(t_i)) \in \tilde{R}$ by Lemma 5.2, which proves property (A).

Since there exists a derivation of smaller depth for $\sigma(t_i) \xrightarrow{b_i} \sigma(y_i)$, by the induction hypothesis, the existence of a process term p_i' such that $\sigma'(t_i) \xrightarrow{b_i} p_i'$ and $(\sigma(y_i), p_i') \in \tilde{R}$ may be assumed. Define $\sigma'(y_i) = p_i'$ and observe that this shows existence of an appropriate process term $\sigma'(y_i)$. This gives rise to $\sigma'(t_i) \xrightarrow{b_i} \sigma'(y_i)$ and $(\sigma(y_i), \sigma'(y_i)) \in \tilde{R}$, which proves properties (B) and (C).

The proof is completed using process substitution σ' for the aforementioned deduction rule. Observe that $\sigma'(f(x_0, \ldots, x_{n-1})) = f(q_0, \ldots, q_{n-1}) = p$. In property (B) it was shown that there exist derivations for all premises using the process substitution σ'. Then, according to the same deduction rule and using σ' instead of σ, it holds that $\sigma'(f(x_0, \ldots, x_{n-1})) \xrightarrow{b} \sigma'(t)$. Since $\sigma'(f(x_0, \ldots, x_{n-1})) = f(p_0, \ldots, p_{n-1}) = p$, it follows that $p \xrightarrow{b} \sigma'(t')$.

It remains to be shown that $(\sigma(t'), \sigma'(t')) \in \tilde{R}$. By Lemma 5.2, only the fact that $(\sigma(x), \sigma'(x)) \in \tilde{R}$ for variables x in t' needs to be shown. The proof may now be completed by considering the following three cases:

1. If $x \in X_p$ then $x = x_i$ for some $0 \leq i < n$. It now holds that $\sigma(x_i) = p_i$ and $\sigma'(x_i) = p_i$ and that $(p_i, q_i) \in \tilde{R}$ and $x_i = x$ was already known. Therefore, it holds that $(\sigma(x), \sigma'(x)) \in \tilde{R}$.

2. If $x \notin X_p$ and $x \notin Y_p$ then $(\sigma(x), \sigma'(x)) \in \tilde{R}$, since $\sigma(x) = \sigma'(x)$ and identity is included in $\tilde{R}$.

3. If $x \in Y_p$ then $x = y_j$ for some $j \in I$. Furthermore, it holds that $(\sigma(y_j), \sigma'(y_j)) \in \tilde{R}$, by property (C). Since $x = y_j$ the inclusion $(\sigma(x), \sigma'(x)) \in \tilde{R}$ also holds.

□

5.2 Controllability

Controllability is now defined and analyzed within the framework of process algebra. A number of observations related to its application in a nondeterministic context are also illustrated in this section. The previously introduced language-based constructs in Section 2.1 now need to be extended to a process-algebraic context. The reflexive transitive closure $\longrightarrow^*$ of the step-relation $\longrightarrow$ is defined as follows: For $p, p' \in \mathcal{T}$ it holds that $p \xrightarrow{\epsilon} {}^* p$, and if $t, v \in \mathcal{A}^*$ with $t = a \cdot v$ then $p \xrightarrow{t} {}^* p'$ if and only if there exists a $q \in \mathcal{T}$ such that $p \xrightarrow{a} q$ and $q \xrightarrow{v} {}^* p'$. To each process term $p \in \mathcal{T}$ corresponds a language $\mathcal{L}(p)$ which is defined as:

$$\mathcal{L}(p) = \left\{ t \in \mathcal{A}^* \mid \exists p' \in \mathcal{T} : p \xrightarrow{t} {}^* p' \right\}$$

As stated previously, a language $L \subseteq \mathcal{A}^*$ is defined to be *prefix closed* if $L = \overline{L}$ where:

$$\overline{L} = \{t \in \mathcal{A}^* \mid \exists t' \in \mathcal{A}^* : t \cdot t' \in L\}$$

The notation $L \cdot L'$ is used to denote the concatenation of the two languages L and L'; more formally:

$$L \cdot L' = \{t \cdot t' \mid t \in L, t' \in L'\}$$

This section now further studies models for the plant, specification and supervisor as closed process terms. For plant $p \in \mathcal{T}$ and supervisor $s \in \mathcal{T}$, the notation s/p is used to denote the *supervised plant*, which acts as a model of the effective realization of the plant under supervisory control. This model is referred to as the controlled system earlier on in this thesis. In order to ensure that s/p does not disable accessible uncontrollable behavior, it is required that s/p is *controllable* with respect to p. This is formalized in Definition 5.8 under the re-iterated assumption that $\mathcal{A} = \mathcal{U} \cup \mathcal{C}$:

Definition 5.8. Let $p \in \mathcal{T}$ be a model of the plant and $s/p \in \mathcal{T}$ be a model of the supervised plant, then p is *language controllable* with respect to s/p if and only if:

$$\mathcal{L}(s/p) \cdot \mathcal{U} \cap \mathcal{L}(p) \subseteq \mathcal{L}(s/p)$$

Definition 5.8 requires that any trace in the supervised plant, followed by an uncontrollable action, should be a trace in the supervised plant, if this trace also occurs in the language of the plant itself. Note how this is a straightforward interpretation of controllability as given in Definition 2.4 for the process algebra framework. In any realistic model of the supervised plant, additional conditions need to be stated to relate s/p to the specification of desired behavior. A useful and straightforward option is to require that $\mathcal{L}(r) = \mathcal{L}(s/p)$, for some specification of desired behavior $r \in \mathcal{T}$.

Language-based controllability according to Definition 5.8 poses a problem in the non-deterministic setting, as briefly introduced in Section 1.2 and as further noticed while exploring the examples in previous chapters. If a decision about event allowance can be made on a per-state basis, problems regarding control and non-determinism can usually be avoided. The remainder of this section is meant to illustrate how to appropriately handle controllability for non-deterministic process terms.

Intuitively, state-based control can be understood in the following manner: the plant communicates its current state to the supervisor, upon which the latter decides the set of actions that can be taken in this state. To define a state based notion of controllability, states therefore need to be taken into account. A solution is therefore proposed in terms of state communication from

the plant to the supervisor. The approach proposed here is loosely based on the work in [57]. State-based controllability is also investigated in the context of non-determinism in [27]. A formalization of state-controllability is shown in Definition 5.9:

Definition 5.9. Let $p \in \mathcal{T}$ and $s/p \in \mathcal{T}$ be process terms representing respectively the plant and the supervised plant. It is then defined that p is *state controllable* with respect to s/p and $\mathcal{U}$ if for all $t \in \mathcal{L}(s/p)$ and $u \in \mathcal{U}$ such that $tu \in \mathcal{L}(p)$ it holds that for all $s/p \xrightarrow{t}{}^{*} q$ there exists a $q' \in \mathcal{T}$ such that $q \xrightarrow{u} q'$.

From this definition it is clear that state controllability implies language controllability. However, there remains an intrinsic problem in this definition of state-controllability, due to the fact that it is not reflexive. That is, if a strictly non-deterministic plant is equal to the supervised plant, in general state controllability as given in Definition 5.9 does not hold, as shown in [6]. The absence of the reflexivity property implies that state-controllability cannot be defined as a preorder. The definition of partial bisimilarity according to Definition 5.4 may be applied to resolve this problem by expressing the necessary conditions for the supervised plant by following Definition 5.10.

Definition 5.10. Let $p \in \mathcal{T}$ and $r \in \mathcal{T}$ be process-theoretic expressions of respectively the plant and control specification. If $\mathcal{A} = \mathcal{U} \cup \mathcal{C}$, then $s \in \mathcal{T}$ is a *supervisor* for p that *satisfies* r if:

$$s/p \preceq^{\mathcal{U}} p \text{ and } s/p \preceq^{\emptyset} r$$

where s/p is again the model of the *supervised plant*, i.e. the model of the plant under supervisory control, also previously referred to as the *controlled system*. It will be shown that setting s/p to $p \parallel s$ will often be appropriate in the context of process algebra, under a number of additional conditions.

From the first condition in Definition 5.10 it is clear that no accessible uncontrollable actions are disabled in the supervised plant, since $\mathcal{U}$ is included in the bisimulation action set. It is therefore immediately clear that the aforementioned condition of $\mathcal{L}(p \parallel s) \cdot \mathcal{U} \cap \mathcal{L}(p) \subseteq \mathcal{L}(p \parallel s)$ is satisfied. The second condition in Definition 5.10 states that the supervised plant is an actual realization of the behavior as formulated in the requirements.

The following lemma states that partial bisimilarity is a less coarse notion compared to state-controllability. It will be shown that the existence of a partial bisimulation implies state-controllability, however, the inversion of this statement is not true.

Lemma 5.4. If $p, q \in \mathcal{T}$ such that $q \preceq^{\mathcal{U}} p$, then p is state controllable with respect to q.

Proof. Let $p, q \in \mathcal{T}$ such that $q \preceq^{\mathcal{U}} p$ for the partial bisimulation $R \subseteq \mathcal{T} \times \mathcal{T}$. It needs to be shown that the following condition is satisfied: for $s \in \mathcal{L}(q)$ and $a \in \mathcal{U}$ such that $sa \in \mathcal{L}(p)$ and for all $q \xrightarrow{s}{}^{*} q'$ there exists a $q'' \in \mathcal{T}$ such that $q' \xrightarrow{a} q''$.

Let $s \in \mathcal{L}(q)$ and $q' \in \mathcal{T}$ such that $q \xrightarrow{s}{}^{*} q'$ and let $a \in \mathcal{U}$ such that $sa \in \mathcal{L}(p)$ with $p \xrightarrow{s}{}^{*} p'$ for $p' \in \mathcal{T}$. It is clear that $q \xrightarrow{s}{}^{*} q'$ and $(q', p') \in R$ and therefore there exists a $q'' \in \mathcal{T}$ such that $q' \xrightarrow{a} q''$ as follows directly from partial bisimilarity according to Definition 5.4. □

Using the process theory TCP* in conjunction with partial bisimilarity, precise formulations of the elements and functionality within the control loop can be given. In general, the plant and specification can be modeled appropriately using TCP*, as there are appropriate constructions such as nondeterministic choice, sequentiality, iteration and communication available. This allows even complicated plants and specifications to be modeled at the right abstraction level. Allowance of uncontrollable behavior and prevention of controllable events is modeled by parameterizing the partial bisimilarity preorder with the set of uncontrollable events, as shown in Definition 5.10. Encapsulation in conjunction with communication can be used to enforce occurrence of only complete communication actions. An important remark has to be made with regard to these communication actions, as shown in the following example:

Consider a simple plant p in which two parallel machines m_1 and m_2 are signaled by local controller g that a product is ready for further processing. This process repeats itself indefinitely and is modeled by the following definitions. Let $p, m_1, m_2, g \in \mathcal{T}$ such that:

$$\begin{aligned} m_1 &\equiv (c?\mathit{ready} \cdot \mathit{process}_1)^* \\ m_2 &\equiv (c?\mathit{ready} \cdot \mathit{process}_2)^* \\ g &\equiv (c!\mathit{ready})^* \\ p &\equiv m_1 \parallel m_2 \parallel g \end{aligned}$$

Furthermore, it is assumed that in this example all actions are controllable. According to the specification that first m_1 and then m_2 needs to be executed repeatedly, the now following formulation is required:

$$r \equiv (c!?_2\mathit{ready} \cdot \mathit{process}_1 \cdot \mathit{process}_2)^*$$

Assume that a supervisor s needs to be constructed which satisfies the condition of $s/p \preceq^{\emptyset} r$. As mentioned before, s/p can be set to $p \parallel s$. This means that the allowed controllable traces from $p \parallel s$ should be simulated by r. Due to the fact that it is assumed that $\mathcal{U} = \emptyset$, it might seem straightforward to choose $s = r$. However, using the operational semantics of the parallel composition operator in Definition 5.3, it can be observed that synchronizations are essentially 'multiplied' in $p \parallel s$. For instance, if $c!_1?_2$*ready* occurs in a trace of p, then $c!_2?_4$*ready* occurs in a trace of $p \parallel s$, which is clearly not simulated by r.

To solve the issue raised in the previous example, an appropriate *renaming* operator $\xi : \mathcal{T} \to \mathcal{T}$ is introduced which traverses the term to which it is applied recursively. This renaming operator is introduced in Definition 5.11. It has to be partially redefined in each instance it is used to list the exact renamed actions.

Definition 5.11. The renaming operator $\xi : \mathcal{T} \to \mathcal{T}$ is defined according to the following pattern. Let $p, q \in \mathcal{T}$ in the following definition:

$$\begin{aligned} \xi(0) &= 0 \\ \xi(1) &= 1 \\ \xi(p+q) &= \xi(p) + \xi(q) \\ \xi(p \cdot q) &= \xi(p) \cdot \xi(q) \\ \xi(p^*) &= \xi(p)^* \\ \xi(p \parallel q) &= \xi(p) \parallel \xi(q) \end{aligned}$$

The renaming function ξ in Definition 5.11 is applied in the example in Section 5.3, where the process algebra TCP* is used to model a case study of automated guided vehicles.

5.3 Event-Based Supervision: AGV Case

In this section, the approach to supervisory control and the model of the control loop is illustrated. A relatively simple example concerning coordination of an automated guided vehicle (AGV) in an automated production line is depicted in Figure 5.1. The AGV is responsible for transferring the preproduct made by Workstation M to Workstation N and transferring the finished product from Workstation N to the Delivery station. These are two phases that need to be executed in sequence.

The workstations and the AGV are coordinated by a supervisor, which sends the corresponding control signals. It will be shown here how to model

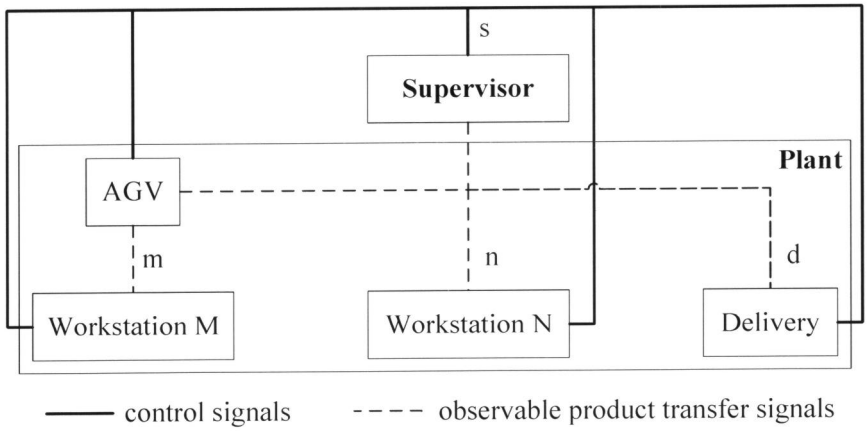

Figure 5.1: Illustrating the case for automated guided vehicles. The preproduct made by workstation M is transferred to workstation N and subsequently transferred to the delivery station. This entire process is coordinated by a supervisor.

the automated production system from Figure 5.1 using TCP*. The process terms M, N, A and S are used to model Workstation M, Workstation N, the AGV and the supervisor respectively. Note that this model abstracts from the delivery station (modeled by a single event *deliver*), as it does not contribute to any relevant behavior. The same communication channel names as shown in Figure 5.1 are used. The data elements are $\mathcal{D} = \{make, move2N, preproduct, product\}$. Uncontrollable events are $\mathcal{U} = \{m, n, produce, process, move, deliver\}$ and as controllable event there is $\mathcal{C} = \{s\}$. The following instantiations are applied for this example:

$$
\begin{aligned}
M &\equiv (s?make \cdot produce(preproduct) \cdot m!preproduct)^* \\
N &\equiv (n?preproduct \cdot process(preproduct) \cdot n!product\cdot)^* \\
A &\equiv (m?preproduct \cdot s?move2N \cdot move(preproduct) \cdot n!preproduct + \\
&\quad\ n?product \cdot deliver(product))^* \\
S &\equiv (s!make \cdot s!move2N)^*
\end{aligned}
$$

Workstation M repeatedly waits for a command from the supervisor to make a preproduct, which is offered to the AGV once it is made. Workstation N waits for a preproduct from the AGV, which is thereafter processed and offered back to the AGV. The AGV can either pick up a preproduct at workstation M, after which it asks for permission to move the preproduct to

Workstation N, or pick up a finished product at Workstation N, and deliver it. Now, the unsupervised plant is modeled by the process term:

$$U \equiv \partial_F(M \parallel N \parallel A), \text{ where } F = \{m?, m!, n?, n!\}$$

At this point, encapsulation is applied to enforce meaningful communication within the plant. This type of encapsulation does not restrict the behavior of the unsupervised plant, but only ensures its meaningful behavior. Following the framework outlined above, it can be readily observed that the plant $U \in \mathcal{T}$ follows the outlined syntax.

In this first modeling instance, it is assumed that the AGV is responsible for delivering the final product and a supervisor is proposed as given by the process S. Note that the supervisor $S \in \mathcal{T}$ follows the outlined syntax and it does not make use of any observed information. Supervisor S repeatedly gives orders to Workstation M for new products to be made, followed by orders to the AGV to transfer the preproduct to Workstation N. Thus, the automated production system is modeled as:

$$U/S \equiv \partial_E(S \parallel U), \text{ where } E = \{s?, s!\}$$

This enforces communication of control signals and transfer of the products. One can directly check that S is a valid supervisor by establishing that the supervised plant is partially bisimulated by the original plant with respect to the uncontrollable events. To this end, renaming of events must be employed, as the original plant has open communication actions that wait for synchronization with the supervisor. This renaming function ξ traverses the process terms and renames all open communication actions to succeeded communication actions. The aforementioned renaming function is applied to the communication action names as well. When considering renaming, only the actions which are actually renamed are mentioned. Now, in order to verify that the supervisor does not disable accessible uncontrollable events, it is sufficient to verify that the following holds:

$$U/S \preceq^{\mathcal{A}_U} \xi(U), \text{ where } \xi : s?d \mapsto s!?d \text{ for } d \in \mathcal{D}$$

This can be directly verified. No restriction is imposed upon the control specification, which in this case coincide with the plant and are, therefore, trivially satisfied.

Unfortunately, this automated production system has a deadlock. The main reason for the deadlock is that a second preproduct can come too early, before the first product is completely finished and delivered, which is set off

by sending a $s!$*make* command too early, that is, before the processed product has left Workstation N. Then, the AGV picks up the preproduct from Workstation M, but it cannot deliver it to Workstation N, as the latter also waits for a finished product to be picked.

Such form of blocking behavior appears often, so in many cases the supervisor is additionally required to prevent situations in which blocking behavior such as deadlock and livelock occurs. This is a typical example of a situation where marked states are introduced in a supervisory control setting such as described in the previous chapters. Note that these states roughly correspond to successful termination in our setting. The correspondence is not strict, mainly due to the absence of sequential composition and the Kleene star operator in the supervisory control literature and the role of the successful termination in these contexts. Note that the marked states do not contribute to the formation of the recognized language of an automaton, which is different from its marked language.

So, besides the control specification, an additional deadlock freeness requirement is imposed on the supervisor, stated formally as: there exists no trace $t \in \mathcal{A}^*$ such that $U/S \xrightarrow{t}{}^* u$, for $u \in \mathcal{T}$ and $u \underline{\leftrightarrow} 0$. To ensure this additional nonblocking requirement, the supervisor needs to be modified to accept requests for making a new preproduct only after the finished product has been loaded on the AGV, to be transferred to the delivery station.

To this end, the supervisor should allow for a new product to be made only after the finished product has been loaded to the AGV at Workstation N, which can be achieved by observing this additional information on channel n. To this end, the supervisor is modified as follows:

$$S \equiv (s!\textit{make}.s!\textit{move2N}.n?\textit{product})^*$$

At this point, note that communication on the channel n now must occur between three parties. This situation occurs between Workstation N that sends information and the AGV and the supervisor which receives it. In order to enforce this communication, communication actions are employed. All (incomplete) communication actions in N are encapsulated, except for $n!_1?_2\textit{product}$. The definition of the deadlock-free supervised plant now becomes:

$$U/S' \equiv \partial_{E'}(S' \parallel U), \text{ where } E' = \{s?, s!, n?, n!?\}$$

Again, one directly verifies that the supervisor is valid by establishing partial bisimilarity between the supervised plant and the original plant model following an appropriate definition of renaming of the incomplete communication actions, given by $\xi : s?d \mapsto s!?d, n!?d \mapsto n!_1?_2 d$ for $d \in \mathcal{D}$.

5.4 The Process Theory $\mathrm{TCP}^*_\perp$

This section proposes the theory $\mathrm{TCP}^*_\perp$, an extension of TCP^*, with propositional signals and guarded commands in order to support the modeling of a control loop with state-based observations. The end purpose of this setup is to model asymmetrically supervised plants, tailored towards the specific needs for plants as well as supervisors. The asymmetric nature of this construction becomes clear from the following intuitive explanation of the purpose of plant-supervisor combinations modeled in $\mathrm{TCP}^*_\perp$. The plant communicates its current state to the supervisor, upon which a list of enabled signals is sent back to the plant by the supervisor. Therefore, besides the standard process terms as discussed before, the plant has to be able to perform outward communication of its state. On the other hand, the supervisor has to be able to receive this information and to either allow or disallow it, thereby obviously taking into account the fact that an accessible uncontrollable events should never be disallowed. In this section, the concurrency theory $\mathrm{TCP}^*_\perp$ is defined in order to achieve this. A different grammar for terms is applied which is designated to model plant components compared to supervisors. However, they both rely upon the same operational semantics. Definition 5.12 outlines the grammars for the components in $\mathrm{TCP}^*_\perp$:

Definition 5.12. If $\mathcal{P}$ is a set of propositional symbols, then a standard Boolean algebra $\mathcal{B}$ is defined by the grammar below:

$$\mathcal{B} ::= \mathit{true} \mid \mathit{false} \mid \mathcal{P} \mid \neg\mathcal{B} \mid \mathcal{B} \wedge \mathcal{B} \mid \mathcal{B} \vee \mathcal{B}$$

The set of plant-specific process terms $\mathcal{T}$ is then defined in terms of $\mathcal{B}$ by the grammar shown below. Assume that $E \subseteq \{f!_m?_n \mid f \in \mathcal{H}, m, n \in \mathbb{N}\}$, $c \in \mathcal{C}$, $u \in \mathcal{U}$ and $k, l \in \mathbb{N}$ in the following definition:

$$\mathcal{T} ::= 0 \mid 1 \mid c?d \mid u!_l?_k \mid \mathcal{T}+\mathcal{T} \mid \mathcal{T}\cdot\mathcal{T} \mid \mathcal{T}^* \mid \mathcal{T} \parallel \mathcal{T} \mid \partial_E(\mathcal{T}) \mid \mathcal{B} :\rightarrow \mathcal{T} \mid \mathcal{B} \trianglerighteq \mathcal{T} \mid \perp$$

Finally, the set of supervisor-specific process terms $\mathcal{S}$ is defined here. Assume that $c \in \mathcal{C}$ and $d \in \mathcal{D}$ in the following definition:

$$\mathcal{S} ::= 1 \mid c!d \mid \mathcal{S} + \mathcal{S} \mid \mathcal{S} \cdot \mathcal{S} \mid \mathcal{B} :\rightarrow \mathcal{S} \mid \mathcal{S}^*$$

The Boolean algebra $\mathcal{B}$ in Definition 5.12 includes the constants true and false, negation, conjunction and disjunction and may also be used to express implication. Boolean expressions $b \in \mathcal{B}$ are evaluated with respect a valuation function $v : \mathcal{B} \mapsto \{\mathit{true}, \mathit{false}\}$. The universe of all valuations is denoted by $\mathcal{V}$.

The set of process terms $\mathcal{T}$ in Definition 5.12 is enriched with the *inaccessible process, guarded commands* and *signal emission* [8], compared to Definition 5.2. The inaccessible process, notation $\perp$, specifies the process in which there are inconsistencies between the valuation of the propositional variables and the emitted propositional signals. A guarded command, notation $\phi :\rightarrow p$, specifies a formula $\phi \in \mathcal{B}$ that functions as a delimiter of a process $p \in \mathcal{T}$. If the guard ϕ evaluates to *true*, then the process p is allowed to continue as usual. If ϕ evaluates to *false*, then the guarded process p deadlocks. The root signal emission operator $\phi \curvearrowright p$ emits the propositional signal $\phi \in \mathcal{B}$ until the process $p \in \mathcal{T}$ takes an outgoing transition. A prerequisite is that the propositional signal is consistent with the valuation. To be able to evaluate the Boolean formulas, process terms are coupled to valuations, notation: $\langle p, v\rangle \in \mathcal{T} \times \mathcal{V}$. The dynamics of the valuations, with respect to outgoing labeled transitions, is captured by the predefined valuation *effect* function. This function has the signature $\mathit{effect} : \mathcal{A} \times \mathcal{V} \rightarrow 2^{\mathcal{V}}$. With respect to the valuation, the successful termination predicate needs to be extended to $\downarrow \subseteq \mathcal{T} \times \mathcal{V}$ and the step relation to $\longrightarrow \subseteq \mathcal{T} \times \mathcal{V} \times \mathcal{A} \times \mathcal{T} \times \mathcal{V}$. An additional consistency predicate $\searrow \in \mathcal{T} \times \mathcal{V}$ which checks whether the state is consistent needs to be introduced. The operational rules in Definition 5.13 give the semantics of the new predicate and the transition relation with respect to the new operators.

Definition 5.13. An operational semantics for $\mathrm{TCP}^*_\perp$ is described below. Assume that $v, v', v'' \in \mathcal{V}$, $p, q, p', q' \in \mathcal{T}$, $a \in \mathcal{A}$, $c \in \mathcal{H}$, $k, l, m, n \in \mathbb{N}$ and $E \subseteq \{f!_m?_n \mid f \in \mathcal{H}, m, n \in \mathbb{N}\}$ in the set of derivation rules listed here:

$$\frac{}{\langle 0, v\rangle \searrow} \qquad \frac{}{\langle 1, v\rangle \searrow} \qquad \frac{}{\langle 1, v\rangle \downarrow} \qquad \frac{}{\langle a.p, v\rangle \searrow} \qquad \frac{\langle p, v'\rangle \searrow \quad v' \in \mathit{effect}(a, v)}{\langle a.p, v\rangle \xrightarrow{a} \langle p, v'\rangle}$$

$$\frac{\langle p, v\rangle \xrightarrow{a} \langle p', v'\rangle \quad \langle q, v\rangle \searrow}{\langle p + q, v\rangle \xrightarrow{a} \langle p', v'\rangle} \qquad \frac{\langle p, v\rangle \searrow \quad \langle q, v\rangle \xrightarrow{a} \langle q', v'\rangle}{\langle p + q, v\rangle \xrightarrow{a} \langle q', v'\rangle}$$

$$\frac{\langle p, v\rangle \searrow \quad \langle q, v\rangle \downarrow}{\langle p + q, v\rangle \downarrow} \qquad \frac{\langle p, v\rangle \downarrow \quad \langle q, v\rangle \searrow}{\langle p + q, v\rangle \downarrow} \qquad \frac{\langle p, v\rangle \searrow \quad \langle q, v\rangle \searrow}{\langle p + q, v\rangle \searrow}$$

$$\frac{\langle p, v\rangle \downarrow \quad \langle q, v\rangle \downarrow}{\langle p \cdot q, v\rangle \downarrow} \qquad \frac{\langle p, v\rangle \downarrow \quad \langle q, v\rangle \xrightarrow{a} \langle q', v'\rangle}{\langle p \cdot q, v\rangle \xrightarrow{a} \langle q', v'\rangle}$$

$$\frac{\langle p, v\rangle \xrightarrow{a} \langle p', v'\rangle \quad \langle p' \cdot q, v'\rangle \searrow}{\langle p \cdot q, v\rangle \xrightarrow{a} \langle p' \cdot q, v'\rangle} \qquad \frac{\langle p, v\rangle \downarrow \quad \langle q, v\rangle \searrow}{\langle p \cdot q, v\rangle \searrow}$$

Definition 5.13 (cont.) The definition of the operational semantics for $\mathrm{TCP}^*_\perp$ is continued below:

$$\frac{\langle p,v\rangle \searrow \quad \langle p,v\rangle \not\downarrow}{\langle p\cdot q,v\rangle \searrow} \qquad \frac{\langle p,v\rangle \searrow}{\langle p^*,v\rangle \downarrow} \qquad \frac{\langle p,v\rangle \searrow}{\langle p^*,v\rangle \searrow}$$

$$\frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle}{\langle p^*,v\rangle \xrightarrow{a} \langle p'\cdot p^*,v'\rangle} \qquad \frac{\langle p,v\rangle \downarrow \quad \langle q,v\rangle \downarrow}{\langle p \parallel q,v\rangle \downarrow} \qquad \frac{\langle p,v\rangle \searrow \quad \langle q,v\rangle \searrow}{\langle p \parallel q,v\rangle \searrow}$$

$$\frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle \quad \langle q,v\rangle \searrow \quad \langle q,v'\rangle \searrow}{\langle p \parallel q,v\rangle \xrightarrow{a} \langle p' \parallel q,v'\rangle}$$

$$\frac{\langle p,v\rangle \searrow \quad \langle p,v'\rangle \searrow \quad \langle q,v\rangle \xrightarrow{a} \langle q',v'\rangle}{\langle p \parallel q,v\rangle \xrightarrow{a} \langle p \parallel q',v'\rangle}$$

$$\left[\frac{\langle p,v\rangle \xrightarrow{c!_l?_k d} \langle p',v'\rangle \quad \langle q,v\rangle \xrightarrow{c!_m?_n d} \langle q',v''\rangle \quad \langle p' \parallel q',v''\rangle \searrow \quad v' \in \mathit{effect}(c!_{l+m}?_{k+n}d,v)}{\langle p \parallel q,v\rangle \xrightarrow{c!_{l+m}?_{k+n}d} \langle p' \parallel q',v'''\rangle}\right]$$

$$\frac{\langle p,v\rangle \downarrow}{\langle \partial_E(p),v\rangle \downarrow} \qquad \frac{\langle p,v\rangle \searrow}{\langle \partial_E(p),v\rangle \searrow}$$

$$\frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle \quad a \notin \{c!_m?_n d \mid c!_m?_n \in E, d \in \mathcal{D}\}}{\langle \partial_E(p),v\rangle \xrightarrow{a} \langle \partial_E(p'),v'\rangle}$$

$$\frac{\langle p,v\rangle \downarrow \quad v(\phi) = \mathit{true}}{\langle \phi :\to p,v\rangle \downarrow} \qquad \frac{\langle p,v\rangle \searrow \quad v(\phi) = \mathit{true}}{\langle \phi :\to p,v\rangle \searrow} \qquad \frac{v(\phi) = \mathit{false}}{\langle \phi :\to p,v\rangle \searrow}$$

$$\frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle \quad v(\phi) = \mathit{true}}{\langle \phi :\to p,v\rangle \xrightarrow{a} \langle p',v'\rangle} \qquad \frac{\langle p,v\rangle \downarrow \quad v(\phi) = \mathit{true}}{\langle \phi \trianglerighteq p,v\rangle \downarrow}$$

$$\frac{\langle p,v\rangle \searrow \quad v(\phi) = \mathit{true}}{\langle \phi \trianglerighteq p,v\rangle \searrow} \qquad \frac{\langle p,v\rangle \xrightarrow{a} \langle p',v'\rangle \quad v(\phi) = \mathit{true}}{\langle \phi \trianglerighteq p,v\rangle \xrightarrow{a} \langle p',v'\rangle}$$

Some brief comments on the rules in Definition 5.13 are given here. The first two rules indicate that the valuations are consistent with respect to the constant terms 0 and 1. An equivalent rule to the one in Definition 5.3 ex-

presses how the constant 1 can always terminate. The next rule shows how action symbols do not influence consistency. A subsequent rule determines how any action is allowed to take a step based on the result of the *effect* function. Alternative composition behaves as in TCP^*, provided that consistency is given. In addition, consistency is required for termination of alternative composition. Consistency of both operands is transferred to consistency of a sum, as expressed in the next rule. It is further detailed how termination of sequential composition and a transition from the right operand behaves as in TCP^*. In the case of a left operand transition, consistency is required for the resulting term $p' \cdot q$. Termination of the left operand transfers consistency for sequential composition. Consistency for the left side $\langle p, v\rangle$ only transfers to the product $\langle p \cdot q, v\rangle$ on the condition that the left side $\langle p, v\rangle$ does not terminate. The two next rules state that termination and consistency for parallel composition depend on both operands. Two derivation rules are required to detail how unilateral steps in the parallel operator $\|$ take place. Note that this depends upon consistency of the term which remains constant. The compositional rule shows how a bilateral parallel step can be enabled by the *effect* function, and is further similar to that of TCP^*. The three rules for encapsulation show the behavior of this operator in this new setting. A *true* valuation is required for termination, as well as consistency of a guarded term. The next somewhat remarkable rule shows that a guarded term is consistent, even if its corresponding valuation evaluates to *false*. A subsequent rule details the behavior of a guarded process, enabling the underlying term only if the valuation ϕ is *true*. The last three rules consider the signal emission operator. A *true* valuation is required for the signal emitting term to terminate and to be consistent. The last rule details how a *true* valuation is required for a transition step as well.

An additional property of the *effect* function is required in order for it to be well-defined [8]. Let $c \in \mathcal{H}$, $d \in \mathcal{D}$, and $l, k, m, n \in \mathbb{N}$ with $l + k > 0$ and $m + n > 0$ in the following additional requirement:

$$\begin{gathered} \mathit{effect}(c!_{l+m}?_{k+n}d) \subseteq \\ \mathit{effect}(c!_m?_n d, \mathit{effect}(c!_l?_k d, v)) \bigcap \mathit{effect}(c!_l?_k d, \mathit{effect}(c!_m?_n d, v)) \end{gathered}$$

Definition 5.4 needs to be adapted in such a way that it correctly handles valuations. The approach is based on work in [8], where this extension is investigated for (strict) bisimilarity.

Definition 5.14. A relation $R \subseteq \mathcal{T} \times \mathcal{T}$ is defined to be a partial bisimulation with respect to the bisimulation action set $B \subseteq \mathcal{A}$ if for all $(p, q) \in R$ it holds that:

1. If $\langle p, v \rangle \downarrow$ for some $v \in \mathcal{V}$ then $\langle q, v \rangle \downarrow$; and
2. If $\langle q, v \rangle \downarrow$ for some $v \in \mathcal{V}$ then $\langle p, v \rangle \downarrow$; and
3. If $\langle p, v \rangle \xrightarrow{a} \langle p', v' \rangle$ for some $v, v' \in \mathcal{V}$ and $a \in \mathcal{A}$ then there exists a $q' \in \mathcal{T}$ such that $\langle q, v \rangle \xrightarrow{a} \langle q', v' \rangle$ and $(p', q') \in R$; and
4. If $\langle q, v \rangle \xrightarrow{b} \langle q', v' \rangle$ for some $v, v' \in \mathcal{V}$ and $b \in B$ then there exists a $p' \in \mathcal{T}$ such that $\langle p, v \rangle \xrightarrow{b} \langle p', v' \rangle$ and $(p', q') \in R$.

5.5 Case Study

The process theory $\mathrm{TCP}^*_\perp$ is employed to model the coordination of maintenance procedures of a printing process of a high-end industrial printer [63]. This is an approach similar to that of Section 3.5. However, a short new description is required in terms of the formalisms introduced in this chapter. The Status Procedure is responsible for coordinating the other procedures given the input from the controllers. It will be implemented as a supervisory coordinator. The coordination rules are given in terms of guarded process terms below. The Current Power Mode procedure sets the power mode to Run or Standby depending on the enabling signals from the Status Procedure *Stb2Run* and *Run2Stb*, respectively. The confirmation is sent back via the signals *_InRun* and *_InStb*, respectively. Maintenance Operation either carries out a maintenance operation or it is idle. The triggering signal is *OperStart* and the confirmation is sent back by *_OperFinished*. The Page Counter procedure counts the printed pages since the last maintenance and sends signals when soft and hard deadlines have been reached using *_ToSoftDln* and *_ToHardDln*, respectively. The counter is reset each time the maintenance is finished, by receiving the confirmation signal *_OperFinished* from Maintenance Operation. The controller Target Power Mode defines which mode is requested by the manager by sending the control signals *_TargetStb* and *_TargetRun* to the Status Procedure. Maintenance Scheduling receives a request for maintenance from Status Procedure via the signal *SchedOper*, which it forwards to a manager. The manager confirms the scheduling with the other functions and sends a response back to the Status Procedure via the control signal *_ExecOperNow*. It

also receives feedback from Maintenance Operation that the maintenance is finished in order to reset the scheduling.

All previously described procedures are modeled by means of processes. The names of the control signals will be inherited, turning them into communication actions where appropriate. The controllable communicating channels are then given by $\mathcal{C} = \{Run2Stb, Stb2Run, SchedOper, OperStart\}$, modeled as receive communication actions in the plant. Note that an abstraction is made from data elements as communication should only enforce the correct order of events. The other actions are uncontrollable, as indicated by the underscore prefix, where only *_OperFinished* is modeled as a communication action, as the procedure Maintenance operation must send signals and reset Page Counter and Maintenance Scheduling. The signals emitted from the plant uniquely identify the state of the plant. Page Counter is modeled by the process C, where *_OperFinished* is modeled as a receive action, to be synchronized with Maintenance Operation. Maintenance Operation is specified by the process O, where *_OperFinished* broadcasts that the maintenance operation has finished. Target Power Mode is modeled by T, Current Power Mode is given by P and Maintenance Scheduling is modeled as M in the list of process terms shown below:

$$
\begin{aligned}
C \quad \equiv \quad & \Big(in(\textbf{NoDeadline}) \,^\frown (\\
& \quad \_OperFinished? + \\
& \quad \_ToSoftDln \cdot \big(in(\textbf{SoftDeadline}) \,^\frown (\\
& \quad\quad \_OperFinished? + \\
& \quad\quad \_ToHardDln \cdot in(\textbf{HardDeadline}) \,^\frown \_OperFinished?)) \big) \Big)^* \\
\\
O \quad \equiv \quad & \big(in(\textbf{OperIdle}) \,^\frown OperStart? \cdot in(\textbf{OperInProg}) \,^\frown \_OperFinished! \big)^* \\
\\
T \quad \equiv \quad & (in(\textbf{TargetStandby}) \,^\frown \_TargetRun \cdot \\
& \; in(\textbf{TargetRun}) \,^\frown \_TargetStandby)^* \\
\\
P \quad \equiv \quad & (in(\textbf{Standby}) \,^\frown Stb2Run \cdot in(\textbf{Starting}) \,^\frown \_InRun \cdot \\
& \; in(\textbf{Run}) \,^\frown Run2Stb \cdot in(\textbf{Stopping}) \,^\frown \_InStb)^* \\
\\
M \quad \equiv \quad & (in(\textbf{NotScheduled}) \,^\frown SchedOper \cdot in(\textbf{Scheduled}) \,^\frown \_ExecOperNow \cdot \\
& \; in(\textbf{ExecuteNow}) \,^\frown \_OperFinished?)^*
\end{aligned}
$$

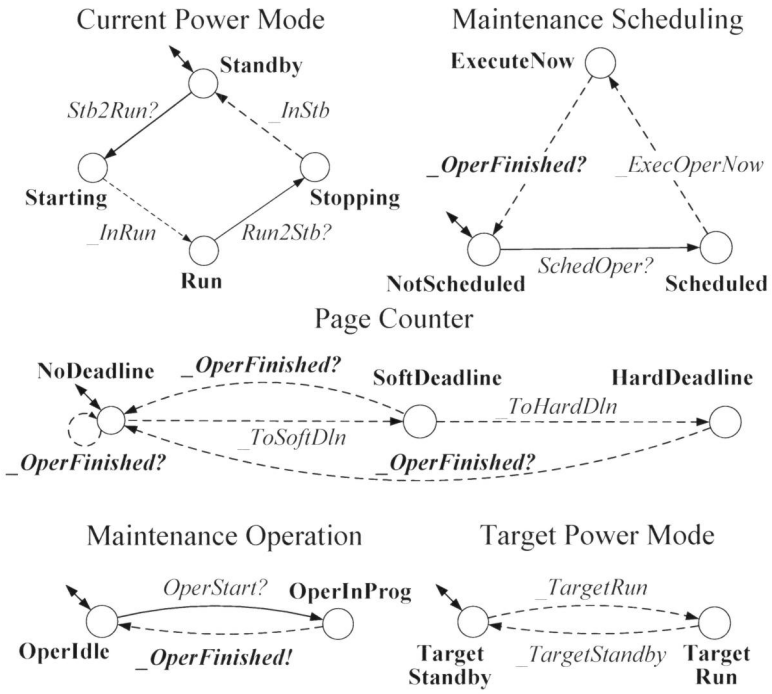

Figure 5.2: A graphical view of the various components in an industrial printer which correspond to the respective process terms C, M, P, O and T. Note that uncontrollable events are drawn by dashed lines in this illustration.

In addition, the unsupervised plant can be specified as $U \in \mathcal{T}$, which may be defined in the following way:

$$U \equiv \partial_F(C \parallel O \parallel T \parallel P \parallel M)$$

For clarity, the relevant processes are depicted in Figure 5.2, where the signal names are given next to the states that emit them. A coordinator which implements Status Procedure can now be constructed. The purpose of this coordinator is to regulate the maintenance procedures with the rest of the printing process. The following coordination requests specify the behavior of the Status Procedure, as described both informally and formally below:

1. Maintenance operations can be performed only when the printing process is in standby. A formalization of this specification therefore re-

quires that the maintenance procedure is performed if the process emits the signal *OperInProg*, while emitting the signal *Standby* as well:

$$R_1 \equiv OperInProg \Rightarrow Standby$$

2. Maintenance operations can be scheduled only if a soft deadline has been reached and there are no print jobs in progress or a hard deadline has passed. For the control signal *SchedOper!* to be sent to Maintenance Scheduling, either one of the following must hold: (1) A soft deadline has been passed, identified by emission of the signal *SoftDeadline*, and there are no print jobs waiting, meaning that the target power mode is not in run, identified by the signal *TargetRun*; or (2) A hard deadline has been passed, indicated by the signal *HardDeadline*. This is captured by the following control specification:

$$R_2 \equiv \xrightarrow{SchedOper!} \Rightarrow (SoftDeadline \wedge \neg TargetRun) \vee HardDeadline$$

3. Maintenance operations can be started only after being scheduled. This means that the maintenance operation can be started by sending the control signal *OperStart!* only if it has been scheduled, prompted by the emission of the signal *_ExecOperNow*:

$$R_3 \equiv \xrightarrow{OperStart!} \Rightarrow ExecuteNow$$

4. The power mode of the printing process must follow the power mode dictated by the managers, unless overridden by a pending maintenance operation. If a switch is made from standby to run power mode, indicated by sending the control signal *Stb2Run!*, then this has been requested by the target power mode manager by emitting the signal *TargetRun*, provided that there are no maintenance operations scheduled, for which the signal *ExecuteNow* should be checked:

$$R_{41} \equiv \xrightarrow{Stb2Run} \Rightarrow TargetRun \wedge \neg ExecuteNow$$

When switching from run to standby power mode, indicated by sending the control signal *Run2Stb!*, the target power mode should be in standby, given by emission of the signal *TargetStandby*. An exception is made when a maintenance operation is scheduled to be executed, given by emission of the signal *ExecuteNow*:

$$R_{42} \equiv \xrightarrow{Run2Stb} \Rightarrow TargetStandby \vee ExecuteNow$$

With respect to the control specification, a deadlock-free supervisor was synthesized [63]. The supervisor sends the control signals upon observation of certain signal combinations, which are given in the form of guards. The indexes of the guards correspond to the indexes of the control specifications which concern the respective control signal:

$$\begin{aligned} g_2 &\equiv (in(\textbf{SoftDeadline}) \wedge in(\textbf{TargetStandby})) \vee in(\textbf{HardDeadline}) \\ g_3 &\equiv in(\textbf{Standby}) \wedge in(\textbf{ExecuteNow}) \\ g_{41} &\equiv \neg in(\textbf{ExecuteNow}) \wedge in(\textbf{TargetRun}) \wedge \neg in(\textbf{OperInProg}) \\ g_{42} &\equiv (\neg in(\textbf{ExecuteNow}) \wedge in(\textbf{TargetStandby})) \vee in(\textbf{ExecuteNow}). \end{aligned}$$

An appropriate supervisor is given by $S \in \mathcal{S}$ as:

$$\begin{aligned} S \equiv \Big(& g_2 :\rightarrow SchedOper! + g_3 :\rightarrow OperStart! + \\ & g_{41} :\rightarrow Run2Stb! + g_{42} :\rightarrow Stb2Run! \Big)^* \end{aligned}$$

Now, the supervised plant U/S is given by:

$$U/S \equiv \partial_E(S \parallel U), \text{ where } E = \{c!, c? \mid c \in \mathcal{H}\}$$

Again, it can be shown that the supervised plant is partially bisimilar to the original plant with respect to the uncontrollable events, by showing that:

$$U/S \preceq^{\mathcal{U}} \xi(U), \text{ where } \xi\colon c? \mapsto c!? \text{ for } c \in \mathcal{H}$$

The above form of the supervisor does not provide much information regarding the choices which are made. For example, it is not difficult to deduce that due to the fact that the initial signal is *Standby*, the event *Run2Stb* is not possible. In addition, *StartOper* is unavailable as the signal *ExecuteNow* is not emitted. In order to better understand the control choices made by the supervisor, an alternative supervisor is depicted in Figure 5.3. Both variants produce equivalent supervised behavior; note that the guards remain the same. The difference is that the supervisor depicted in Figure 5.3 reveals the consequences of choosing a particular controllable action. It may now be observed that if the operation is scheduled while the printing process is in standby power mode, then it can be directly executed, returning the supervisor to the initial state. If the power mode is changed to run, then the operation can still be scheduled, but the system has to switch to standby power mode for it to be executed.

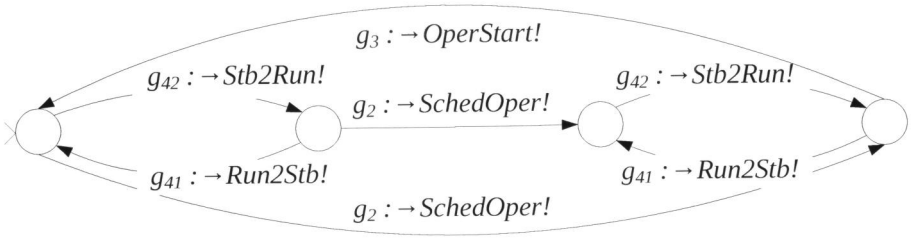

Figure 5.3: An alternative form of the supervisor S may reveal insight into its actual operation. The main difference between the model depicted in Figure 5.3 and S is that the consequences of choosing a particular controllable action are now clear.

5.6 Closing Remarks

The theoretical foundations that were laid out in this chapter can be viewed as more than just a modeling aid which acts as a stepping stone between informal specifications and the creation of an actual supervisor. Instead, it can be argued that a process-theoretic approach to supervisory control theory effectively bridges this gap. The feedback loop for supervisory control is advantageous in the sense that it abstractly models the realistic distinction between applicable control and uncontrollable behavior. This abstract model is given a concrete refinement by providing a convenient formalism to model the plant and supervisor using the process theory TCP*, which has been described extensively in this chapter. This framework is further streamlined by allowing the control specifications for desired behavior to be stated using the very same formalism.

The distinct parts of the process theory TCP* correspond to elementary modeling needs in concrete situations. Parallelism allows the expression of multiple adjoined components in the plant which operate in conjunction. Interactions can be modeled using parallel communication functionality, as clearly present in the theory TCP*. Encapsulation is used to model effective restrictions on communicating processes, thereby providing flexibility in the expression of individual components while retaining the ability to restrict communications if processes are combined. Iteration allows the expression of continuous plant components that have the option to terminate, while allowance of non-determinism provides the ability to integrate all of the aforementioned features in a model at the desired level of abstraction.

Partial bisimilarity is applied as a means to consider process theoretic terms under a preorder modulo structural behavior. This relates to the con-

cept of bisimulation of uncontrollable events, while enabling restrictions on control by means of unilateral simulation. The precongruence property of partial bisimulation was related to the format of the operational rules, thereby allowing an easy generalization towards extensions of the theory.

Controllability was specified as avoiding disallowance of accessible uncontrollable behavior. For language based control this is realized in the definition of language based controllability, which is unsuitable for non-deterministic system models. State based control can be an effective means to capture non-determinism. This is discussed in the light of an earlier approach that does not satisfy reflexivity in its behavioral preorder between plant and supervisor. As the plant is required to be state controllable to itself, the definition of partial bisimilarity was adapted for state-based control and it was showed that it satisfies the required properties. A renaming operator was introduced to enable a parallel and communicating construction of plant and supervisor in the control loop. This feature was used in an example case of automated guided vehicles where TCP^* is used to model parallel components under language based supervisory control.

The process theory TCP^* was extended to include state-based Boolean valuations, guarded commands and signal emission. This enables the expression of outward communicating plant models that adhere to state based control signals. This setup retains the required controllability property by redefining a suitable partial bisimilarity preorder in terms of state-based valuations. The definition of $\mathrm{TCP}^*_\perp$ was followed by an extensive case study into the supervisory control of an industrial printer where five parallel plant components are modeled as communicating processes which operate under state-based control.

Chapter 6

Conclusions

This concluding chapter is set up to first give a general overview of the obtained results, followed by a short analysis. Subsequently, future research questions are formulated. As a first observation, one might say that the research objectives were met in the sense that a sound methodology was outlined which achieves maximally permissive controlled system synthesis for a reasonably expressive modal logic upon non-deterministic behavioral models. The resulting main methodology as described in Chapter 2 and Chapter 3 was derived by first studying basic cases and then working incrementally by expanding the synthesized logic. The formal verification by means of the Coq proofs attributes more certainty to the validity of the proposed theories. The resulting synthesis technique may be straightforwardly expressed in algorithmic form, as shown in Chapter 3. This contributes to the practical usability of the synthesis theory.

The projection of the transition relation of the plant onto a new transition relation over the state-formula product space preserves bisimulation and thereby creates a synthesis starting point which incorporates as much original system structure as possible. The succeeding steps of transition removal conform to an intuitive interpretation of behavioral restriction; behavior is removed until the control objectives are met. It is intended that this fixpoint characterization of control synthesis may one day contribute to a more abstract, broader interpretation of control theory. In such a — perhaps — coinductive model of control theory, more clarity might be obtained regarding the applicability of partial bisimilarity, or a different similar preorder. Regarding the latter notion, partial bisimilarity was employed in this thesis due to earlier research, the fact that it implies controllability from a Ramadge-Wonham

perspective, and its coinductive nature. However, it may not be the definitive answer to all control-theoretic needs.

Synthesis for Hennessy-Milner logic in such a way that it results in multiple solutions offers an interesting perspective on control synthesis due to the fact that it shows how hard it is to find multiple non-deterministic maximally permissive solutions while at the same time preserving a behavioral model close to that of a transition relation. The approach in Chapter 4 is therefore not easily extensible beyond HML, which resulted in the synthesis treatment considered in earlier chapters. However, the research effort was still fruitful in the sense that many initial discoveries for how to apply maximally permissive synthesis upon non-deterministic models were made.

Control synthesis for process algebra as considered in Chapter 5 provides an interesting perspective on how to integrate control theory with an existing formal framework. It is clear from Chapter 5 and earlier research that control synthesis is definable in process algebra and that many useful constructs may be added to a process theory in order to aid in modeling a certain problem. However, the most beneficial part of such a setup probably lays in features already present in process algebra itself, such as abstract modeling of communication and integral treatment of synchronization. A more extensive comparison to the work in [35] may contribute to a better understanding of the exact benefits process algebra can offer in the context of control synthesis.

A first point of analysis concerns the omission in this thesis to generally concern the applicability of non-deterministic models in supervisory control theory. The precise consequences of *not* being able to derive a strictly separated controller such as in [76] therefore remain unclear. Another general remark is that the new techniques for control synthesis proposed in Chapters 2-4 describe a complicated construction to derive a new *transition relation*, which may be considered important from a modeling perspective. However, a number of strong results were obtained in related research [5, 6], which construct a completely different behavioral model. It is therefore unclear to which extent the intrinsic value of the work in this thesis relates to the fact that the described methodologies result in a transition relation, as opposed to any different behavioral model.

Partial bisimilarity is applied and considered in detail at a number of instances in this work. Both partial bisimilarity as given in Definition 2.6 as well as its similar definition in [81] only imply controllability as required in Ramadge-Wonham supervisory control theory if both operands are deterministic. Furthermore, it certainly does not preserve a relevant set of μ-calculus formulas in a control synthesis context, thereby leading to various complications in the proofs in Chapter 3. Its dual application in this research for both

expressing controllability and maximal permissiveness may be too strict, although this clearly can be interpreted as a sound construction. Computational aspects of partial bisimilarity remain unclear but are probably in the same order of complexity as strict bisimulation [31]. A broad analysis of the way in which practical examples of non-deterministic plant models with and without control relate to each other may reveal a better or more suitable ordering relationship between these, compared to partial bisimilarity.

A somewhat critical view upon the work in this thesis cannot omit the fact that it would certainly have benefit from the inclusion of more and larger non-deterministic examples and a more in-depth and optimized analysis of its scalability. However, omission of the latter improvement relates to the fact that non-optimized algorithms were presented in order to preserve close correspondence between these and the mathematical constructs they intend to implement. Furthermore, the applied modal logics were chosen in such a way that soundness of the theories could be derived, rather than logics which were tailor-made towards the expression of relevant practical problems [45, 46].

Future Work

Possible future developments which may evolve from the research in this thesis are briefly considered here. A first and foremost research objective concerns partial bisimulation. It would be fruitful to seek clearance to the question whether partial bisimulation is in fact a well-founded way to coinductively capture controllability in a non-deterministic setting. A number of arguments have been given in this thesis which answer this question in the positive, although a final decisive argument has not been found.

The synthesis construction as defined in this thesis may also be the subject of future new developments. For instance, the synthesized logic may be extended to include basic expressions of a more complex nature. This may be done by including expressions over discrete and/or continuous variables. Modifications have already been added to the CIF toolset [17] as a first attempt to build such an implementation.

A different possible extension may be to expand the synthesized logic itself, by including more complex reachability expressions or a limited class of fixpoint expressions. There are good indications that some type of greatest fixpoint expression would not make the synthesized logic unsound with regard to obtaining unique maximally permissive solutions. The entire synthesis setup may also be subject to change. For instance, a viable option may be

to implement the synthesis construction by means of guards instead of direct transition removal. This may possibly simplify the relationship between the synthesis result and the synthesized logic. However, this change would not directly bring a huge simplification, since guarded expressions would have to be defined in terms of other guards at reachable transitions.

Partial observability relates to non-determinism due to the fact that the latter may be applied to model the former. It might therefore be worthwhile to compare the approach in this thesis to other approaches such as [59] and [71] which study computational hardness caused by partial observability.

The approach to control synthesis in this thesis relies upon both the removal of transitions as a means to achieve proper control flow, as well as the evaluation of formulas in modal logic, for instance in reachability expressions. Both these objectives may be optimized from a BDD-based perspective [36] or dynamic programming approaches [86]. Therefore, a study into future improvements from a computational point of view should determine whether it is possible to apply such techniques in order to achieve a dual improvement.

A number of future research questions are stated below. Some of these may constitute quite daunting tasks and the latter two mainly reflect the research interest of the author:

1. Is partial bisimilarity the most appropriate way to capture controllability in a coinductive context? Clearly, a more extensive comparison between partial bisimilarity as given in Definition 2.6 and the slightly different variant in [81] is required to provide an answer to this question.

2. What is the largest strict subset of the μ-calculus such that a unique maximally permissive control synthesis solution can be found for non-deterministic plant models and specifications of desired behavior from this aforementioned set?

3. Is it possible to coinductively define a preorder which implies controllability but also preserves a subset of the μ-calculus? A solution to this question would avoid the cumbersome task of proving solution validity via induction towards the structure of formulas, as applied in this thesis.

4. Is it possible to modify Definition 2.14 in such a way that a reference to synthesizability at state-formula pairs reachable over $\mathcal{U}^*$ can be avoided? The definition in its present form may still be characterized as too language-theoretic, and a purely coinductive variant would be preferred.

5. Is it possible to apply constructive proof theory in such a way that an algorithm for the construction of a controller can be automatically derived from the combined instances of a controllability and a maximality proof?

6. Is it possible to prove that a complete finite set of axioms for all closed terms over the process algebra with 0, 1 and Kleene-star exists? A solution to this question would have been helpful for the material in Chapter 5. However, it seems to be a very hard problem, see [12].

Bibliography

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Automata, Languages and Programming*, volume 372 of *LNCS*, pages 1–17. Springer, 1989.

[2] L. Alberucci and A. Facchini. On modal μ-calculus and Gödel-Löb logic. *Studia Logica*, 91(2):145–169, 2009.

[3] M. Antoniotti. *Synthesis and Verification of Discrete Controllers for Robotics and Manufacturing Devices with Temporal Logic and the Control-D System*. PhD thesis, New York University, 1995.

[4] M. Antoniotti and B. Mishra. Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers. In *Proceedings of Robotics and Automation*, volume 2, pages 1441–1446. IEEE, 1995.

[5] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

[6] A. Arnold and I. Walukiewicz. Nondeterministic controllers of nondeterministic processes. In *Proceedings of Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 29–52. Amsterdam University Press, 2008.

[7] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 1–20.

[8] J. Baeten, T. Basten, and M. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, 2010.

[9] J. Baeten, B. van Beek, A. van Hulst, and J. Markovski. A Process Algebra for Supervisory Coordination. In *Proceedings of PACO*, volume 60 of *EPTCS*, pages 36–55. Open Publishing Association, 2011.

[10] J. Baeten, B. van Beek, A. van Hulst, and J. Markovski. A Process Algebra for Supervisory Control. Technical Report SE Report 12-01, Eindhoven University of Technology, 2012.

[11] J. Baeten and J. Bergstra. Process Algebra with Propositional Signals. Technical Report 123, Utrecht University, 2008.

[12] J. Baeten, F. Corradini, and C. Grabmayer. A characterization of regular expressions under bisimulation. *Journal of the ACM*, 54(2):1–28, 2007.

[13] B. Barras, S. Boutin, C. Cornes, J. Courant, J. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, and C. Murthy. The Coq proof assistant reference manual: Version 6.1. Technical report, INRIA, 1997.

[14] M. Barveau, F. Kabanza, and R. St-Denis. A method for the synthesis of controllers to handle safety, liveness, and real-time constraints. *IEEE Transactions on Automatic Control*, 43(11):1543–1559, 1998.

[15] S. Basu and R. Kumar. Quotient-based control synthesis for non-deterministic plants with mu-calculus specifications. In *Proceedings of CDC*, pages 6041–6046. IEEE, 2006.

[16] S. Basu and R. Kumar. Quotient-based control synthesis for partially observed non-deterministic plants with mu-calculus specifications. In *Proceedings of CDC*, pages 5294–5299. IEEE, 2007.

[17] D. van Beek, W. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. van de Mortel-Fronczak, and M. Reniers. CIF 3: Model-Based Engineering of Supervisory Controllers. In *Proceedings of TACAS*, volume 8413 of *LNCS*, pages 575–580. Springer, 2014.

[18] A. Bergeron. A unified approach to control problems in discrete event processes. *Informatique Théorique et Applications*, 27(6):555–573, 1993.

[19] B. Bonet, H. Palacios, and H. Geffner. Automatic Derivation of Memoryless Policies and Finite-State Controllers Using Classical Planners. In *Proceedings of ICAPS*. AAAI, 2009.

[20] R. Bull and K. Segerberg. Basic modal logic. In *Handbook of Philosophical Logic*, pages 1–81. Springer, 2001.

[21] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems, Second Edition*. Springer, 2008.

[22] D. Dams and K. Namjoshi. Automata as Abstractions. In *Proceedings of VMCAI*, volume 3385 of *LNCS*, pages 216–232. Springer, 2005.

[23] A. Deshpande and P. Varaiya. Control of discrete event systems in temporal logic. Technical Report 94-4, PATH technical note, 1994.

[24] A. Deshpande and P. Varaiya. Semantic tableau for control of PLTL formulae. In *Proceedings of CDC*, volume 2, pages 2243–2248. IEEE, 1996.

[25] P. Dietrich, R. Malik, M. Wonham, and B. Brandin. Implementation Considerations in Supervisory Control. In *Synthesis and Control of Discrete Event Systems*, pages 185–201. Springer, 2002.

[26] R. Ehlers, S. Lafortune, S. Tripakis, and M. Vardi. Bridging the Gap between Supervisory Control and Reactive Synthesis: Case of Full Observation and Centralized Control. In *Proceedings of WODES*, pages 222–227. IEEE, 2014.

[27] M. Fabian. *On object oriented nondeterministic supervisory control*. PhD thesis, Chalmers University of Technology,, 1995.

[28] M. Fabian and B. Lennartson. A Class of Non-Deterministic Specifications for Supervisory Control. *European Journal of Control*, 3(1):81–90, 1997.

[29] G. Fainekos, A. Girard, and G. Pappas. Hierarchical Synthesis of Hybrid Controllers from Temporal Logic Specifications. In *Proceedings of HSCC*, volume 4416 of *LNCS*, pages 203–216. Springer, 2007.

[30] H. Flordal, R. Malik, M. Fabian, and K. Åkesson. Compositional synthesis of maximally permissive supervisors using supervision equivalence. *Discrete Event Dynamic Systems*, 17(4):475–504, 2007.

[31] R. van Glabbeek. The Linear Time – Branching Time Spectrum II. In *Proceedings of CONCUR*, volume 715 of *LNCS*, pages 66–81. Springer, 1993.

[32] C. Golaszewski and P. Ramadge. Control of discrete event processes with forced events. In *Proceedings of CDC*, volume 26, pages 247–251. IEEE, 1987.

[33] J. Groote and F. Vaandrager. Structured Operational Semantics and Bisimulation as a Congruence. *Information and Computation*, 100(2):202–260, 1992.

[34] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[35] M. Heymann. Concurrency and Discrete Event Control. *IEEE Control Systems Magazine*, 10(4):103–112, 1990.

[36] G. Hoffmann and H. Wong-Toi. Symbolic synthesis of supervisory controllers. In *Proceedings of ACC*, number 29, pages 2789–2793. IEEE, 1992.

[37] L. Holloway and B. Krogh. Synthesis of feedback control logic for a class of controlled Petri nets. *IEEE Transactions on Automatic Control*, 35(5):514–523, 1990.

[38] J. Hopcroft. *Introduction to Automata Theory, Languages, and Computation*. Pearson Education, 1979.

[39] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. The coq proof assistant a tutorial. *Rapport Technique*, 178, 1997.

[40] A. van Hulst, M. Reniers, and W. Fokkink. Maximal Synthesis for Hennessy-Milner Logic. In *Proceedings of ACSD*, pages 1–10. IEEE, 2013.

[41] A. van Hulst, M. Reniers, and W. Fokkink. Maximal synthesis for Hennessy-Milner logic. *ACM Transactions on Embedded Computing*, 14(1):10:1–10:21, 2014.

[42] A. van Hulst, M. Reniers, and W. Fokkink. Maximal Synthesis for Hennessy-Milner Logic with the Box-Modality. In *Proceedings of WODES*, pages 278–285. IEEE, 2014.

[43] A. van Hulst, M. Reniers, and W. Fokkink. Maximally Permissive Controlled System Synthesis for Modal Logic. In *Proceedings of SOFSEM*, volume 8939 of *LNCS*, pages 230–241. Springer, 2015.

[44] A. van Hulst, M. Reniers, and W. Fokkink. Maximally Permissive Controlled System Synthesis for Non-Determinism and Modal Logic. *Discrete Event Dynamic Systems*, pages 10:1–10:21, 2015.

[45] N. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel. Synthesis of live behaviour models. In *Proceedings of FSE*, pages 77–86. ACM, 2010.

[46] N. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel. Synthesizing nonanomalous event-based controllers for liveness goals. *ACM Transactions on Software Engineering and Methodology*, 22(1):9:1–9:36, 2013.

[47] M. Jackson. The world and the machine. In *Proceedings of ICSE*, pages 283–283. IEEE, 1995.

[48] S. Jansen. Design and implementation of model-based controllers for baggage handling systems. From: TU/e Master Thesis archive at: http://www.tue.nl/, 2014.

[49] S. Jiang and R. Kumar. Supervisory control of discrete event systems with CTL* temporal logic specifications. *SIAM Journal on Control and Optimization*, 44(6):2079–2103, 2006.

[50] R. Kamphuis. Design and real-time implementation of a supervisory controller for a part of Veghel airport. From: TU/e Master Thesis archive at: http://www.tue.nl/, 2013.

[51] J. Komenda and J. van Schuppen. Modular Control of Discrete-Event Systems with Coalgebra. *IEEE Transactions on Automatic Control*, 53(2):447–460, 2008.

[52] D. Kozen. Results on the Propositional μ-Calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[53] R. Kumar. *Supervisory synthesis techniques for discrete event dynamical systems*. PhD thesis, University of Texas at Austin, 1992.

[54] R. Kumar and V. Garg. *Modeling and Control of Logical Discrete Event Systems*. Springer, 2012.

[55] R. Kumar, V. Garg, and S. Marcus. On controllability and normality of discrete event dynamical systems. *Systems & Control Letters*, 17(3):157–168, 1991.

[56] R. Kumar, S. Jiang, C. Zhou, and W. Qiu. Polynomial synthesis of supervisor for partially observed discrete-event systems by allowing nondeterminism in control. *IEEE Transactions on Automatic Control*, 50(4):463–475, 2005.

[57] R. Kumar and M. Shayman. Non-blocking supervisory control of nondeterministic discrete event systems. In *Proceedings of ACC*, volume 1, pages 1089–1093. IEEE, 1994.

[58] O. Kupferman, P. Madhusudan, P. Thiagarajan, and M. Vardi. Open Systems in Reactive Environments: Control and Synthesis. In *Proceedings of CONCUR*, volume 1877 of *LNCS*, pages 92–107. Springer, 2000.

[59] H. Lamouchi and J. Thistle. Effective control synthesis for DES under partial observations. In *Proceedings of CDC*, volume 1, pages 22–28. IEEE, 2000.

[60] J. Lin and D. Ionescu. Analysis and synthesis procedures of discrete event systems in a temporal logic framework. In *Proceedings of Intelligent Control*, pages 184–191. IEEE, 1992.

[61] P. Madhusudan. *Control and synthesis of open reactive systems*. PhD thesis, University of Madras, 2009.

[62] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proceedings of STACS*, volume 900 of *LNCS*, pages 229–242. Springer, 1995.

[63] J. Markovski. Coordination of resources using generalized state-based requirements. In *Proceedings of WODES*, pages 287–292. IEEE, 2010.

[64] J. Markovski. A process-theoretic approach to supervisory control theory. In *Proceedings of ACC*, pages 4496–4501. IEEE, 2011.

[65] N. Meuleau, L. Peshkin, K. Kim, and L. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of UAI*, pages 427–436. Morgan Kaufmann, 1999.

[66] M. Mousavi, M. Reniers, and J. Groote. Notions of bisimulation and congruence formats for SOS with data. *Information and Computation*, 200(1):107–147, 2005.

[67] F. Mushtaq. *The safe design of computer controlled pipeless batch plants*. PhD thesis, Loughborough University, 2000.

[68] J. Ostroff. Synthesis of controllers for real-time discrete event systems. In *Proceedings of CDC*, volume 1, pages 138–144. IEEE, 1989.

[69] A. Overkamp. Control of Non-deterministic Discrete Event Systems using Failure Semantics. In *Proceedings of ECC*, volume 95. IEEE, 1995.

[70] S. Pinchinat and S. Riedweg. Quantified Mu-Calculus for Control Synthesis. In *Proceedings of MFCS*, volume 2747 of *LNCS*, pages 642–651. Springer, 2003.

[71] S. Pinchinat and S. Riedweg. A decidable class of problems for control under partial observation. *Information Processing Letters*, 95(4):454–460, 2005.

[72] A. Pnueli, E. Asarin, O. Maler, and J. Sifakis. Controller synthesis for timed automata. In *Proceedings of SSC*. Elsevier, 1998.

[73] A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *Proceedings of POPL*, pages 179–190. ACM, 1989.

[74] C. Pralet, G. Verfaillie, M. Lemaître, and G. Infantes. Constraint-Based Controller Synthesis in Non-Deterministic and Partially Observable Domains. In *Proceedings of ECAI*, volume 215, pages 681–686. IOS, 2010.

[75] P. Ramadge. Some tractable supervisory control problems for discrete-event systems modeled by buchi automata. *IEEE Transactions on Automatic Control*, 34(1):10–19, 1989.

[76] P. Ramadge and M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[77] J. Rutten. Coalgebra, Concurrency, and Control. In *Discrete Event Systems: Analysis and Control*, pages 31–38. Springer, 2000.

[78] Giovanni Sambin and Silvio Valentini. The modal logic of provability. the sequential approach. *Journal of Philosophical Logic*, 11(3):311–342, 1982.

[79] C. Seatzu, M. Silva, and J. van Schuppen. *Control of Discrete-Event Systems: Automata and Petri Net Perspectives*. Springer, 2013.

[80] M Shayman and R. Kumar. Supervisory control of nondeterministic discrete event dynamical systems. In *Proceedings of CDC*, volume 2, pages 1188–1193. IEEE, 1993.

[81] R. Su. Supervisor synthesis based on abstractions of nondeterministic automata. In *Proceedings of WODES*, pages 412–418. IEEE, 2008.

[82] P. Tabuada. An Approximate Simulation Approach to Symbolic Control. *IEEE Transactions on Automatic Control*, 53(6):1406–1418, 2008.

[83] J. Thistle and M. Wonham. Control problems in a temporal logic framework. *International Journal of Control*, 44(4):943–976, 1986.

[84] J. Thistle and M. Wonham. Control of Infinite Behavior of Finite Automata. *SIAM Journal on Control and Optimization*, 32(4):1075–1097, 1994.

[85] J. Thistle and M. Wonham. Supervision of Infinite Behavior of Discrete-Event Systems. *SIAM Journal on Control and Optimization*, 32(4):1098–1113, 1994.

[86] J. Tsitsiklis. On the control of discrete-event dynamical systems. *Mathematics of Control, Signals and Systems*, 2(2):95–107, 1989.

[87] E. Wolff, U. Topcu, and R. Murray. Efficient reactive controller synthesis for a fragment of linear temporal logic. In *Proceedings of ICRA*, pages 5033–5040. IEEE, 2013.

[88] M. Wonham and P. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25(3):637–659, 1987.

[89] C. Zhou, R. Kumar, and S. Jiang. Control of nondeterministic discrete-event systems for bisimulation equivalence. *IEEE Transactions on Automatic Control*, 51(5):754–765, 2006.

Summary

This thesis describes the developments within four years of research into the automated synthesis of controlled systems. The main research question is as follows: given a non-deterministic behavioral description of a system in terms of states and state transitions and given a logical specification of desired behavior, how is it possible to restrict system behavior such that the system conforms to the specification. In deriving a new behavioral description we require that it conforms to a number of properties of supervisory control theory. For instance, it is not allowed to forbid accessible uncontrollable behavior (controllability property) and furthermore the behavioral restriction is required to be minimally restrictive. Such research finds applications in the control of for example manufacturing networks or systems of conveyor belts.

The chosen approach is of a formal mathematical nature and based upon an abstract description of the underlying system as a Kripke-model, combined with a specification in modal logic of desired behavior. Part of the performed research is for which logical specifications this research question is solvable at all. In addition, a precise formulation of the problem to be resolved has been the subject of research, in particular establishing a relational connection between the original system and the synthesis result via partial bisimulation. A solution mechanism for a reasonably expressive logical formalism including invariant and reachability formulas has been investigated extensively.

Central issue within the applied methodology are modifications to the transition relation which expresses state transitions. Hereby states are coupled to logical expressions which have to be locally valid. Following this step, transitions may be removed based upon a validity approximation of expressions which have been assigned to target states of a transition. Based upon the assumption that the original system is modeled by finitely many transitions, we can prove that stabilization takes place. This approach has

been formally verified during the research and may thereby considered to be valid.

Special attention deserves the research approach via the Coq proof assistant. Using this software it is possible to very precisely verify the validity of mathematical definitions and the proofs based on those. This relates to both the research results and the research path. In the end, there is more certainty about the validity of the obtained results, but significant time has to be invested into formally establishing mathematical structures and proof steps.

The research within this thesis is closed by a chapter about process algebraic descriptions in relation to the synthesis problem. Using this alternative formalism, expression of the synthesis problem is well possible, as also shown in earlier research, having the additional advantage that the process algebra considered here provides a rich expression formalism which enables the detailed description of existing system models.

Samenvatting

Dit proefschrift beschrijft de ontwikkelingen binnen vier jaar onderzoek naar de automatische synthese van controlled systems. Centraal staat de volgende onderzoeksvraag: gegeven een non-deterministische gedragsbeschrijving van een systeem in termen van toestanden en toestandsovergangen en gegeven een logische specificatie van gewenst gedrag, hoe is het mogelijk om een systeem-gedrag te beperken zodat het systeem voldoet aan de specificatie? Bij het afleiden van een nieuwe gedragsbeschrijving is het nodig dat deze voldoet aan een aantal eigenschappen van supervisory control theory. Zo is het bijvoorbeeld niet toegestaan om bereikbaar oncontrolleerbaar gedrag te verbieden (controllability-eigenschap) en bovendien dient de gedragsaanpassing minimaal-restrictief te zijn. Dergelijk onderzoek kent toepassingen binnen de aansturing van bijvoorbeeld fabricagesystemen of systemen van transportbanden.

De geijkte aanpak is formeel-wiskundig van aard en gebaseerd op een abstracte beschrijving van het onderliggende systeem als Kripke-model, gecombineerd met een modaal-logische specificatie van gewenst gedrag. Onderdeel van het gedane onderzoek is voor welke logische specificaties deze onderzoeksvraag überhaupt oplosbaar is. Bovendien is de precieze formulering van het op te lossen probleem onderwerp van onderzoek geweest, met name het vastleggen van een relationeel verband tussen het oorspronkelijke systeem en het synthese-resultaat via partiële bisimulatie. Een oplossingsmethodiek voor een redelijkerwijs expressief logisch formalisme met invariant- en bereikbaarheidsformules is uitgebreid onderzocht.

Centraal binnen de gevonden aanpak staan wijzigingen aan de transitierelatie die de toestandsovergangen beschrijft. Hierbij worden toestandsnamen gekoppeld aan logische expressies die lokaal geldig dienen te zijn. Vervolgens kan men transities verwijderen op basis van een geldigheidsbenadering van expressies die aan doeltoestanden van transities zijn toegewezen. Op ba-

sis van de aanname dat het oorspronkelijke systeem wordt beschreven door eindig veel transities is dan afleidbaar dat stabilisatie plaatsvindt. Deze aanpak is gedurende het onderzoek formeel geverifieerd en kan daarmee als valide worden beschouwd.

Speciale aandacht verdient de onderzoeksaanpak via de bewijsassistent Coq. Met behulp van deze programmatuur is het mogelijk de validiteit van wiskundige definities en daarop gebaseerde bewijzen uiterst precies te onderzoeken. Dit houdt verband met zowel onderzoeksresultaat als onderzoeksverloop. Er is uiteindelijk meer zekerheid over de validiteit van verkregen resultaten, maar er dient ook een aanzienlijke tijd te worden geïnvesteerd in het formeel vastleggen van wiskundige structuren en bewijsstappen.

Het onderzoek binnen dit proefschrift wordt afgesloten met een hoofdstuk over procesalgebraïsche beschrijvingen in relatie tot het syntheseprobleem. Binnen dit alternatieve formalisme is uitdrukking van het syntheseprobleem goed mogelijk, zoals reeds aangetoond in eerder onderzoek, met als bijkomend voordeel dat de procesalgebra zoals uiteengezet in dit proefschrift een rijk formalisme kent dat kan worden ingezet voor het gedetailleerd modelleren van bestaande systemen.

Curriculum Vitae

Allan Carolus van Hulst was born on the 10th of May 1985 in Nijmegen. After obtaining the VWO (Atheneum) diploma from the Dominicus College, he went to study computer science at the Katholieke Universiteit Nijmegen (KUN), later renamed into the Radboud University Nijmegen (RU). During his five-year studies in computer science, Allan completed the Honours Program, resulting in a publication in the bi-annual magazine Honours Review. In addition, he participated in a three-month summer development program for Open Source, which was noted by the Volkskrant newspaper.[6] Besides his studies in computer science, Allan was a teaching assistant for many introductory programming courses and a course on computational complexity theory.

Following his studies in computer science, Allan started his work as a PhD-student at the Systems Engineering group at Eindhoven University of Technology. Under the supervision of his advisors prof.dr. Jos Baeten and prof.dr. Wan Fokkink and under the daily guidance of dr.ir. Michel Reniers, Allan worked on the topic of controlled system synthesis. This thesis is a direct result of his work. Furthermore, Allan participated in several teaching efforts such as a course on the verification of discrete event systems using the UPPAAL toolset, which was very positively rewarded by the students. In the future, Allan intends to actively contribute to scientific developments in theoretical computer science and computer verified mathematics.

[6] Available at: `http://www.volkskrant.nl/economie/-ik-wil-hier-niet-te-veel-profiteren~a844535/`

Titles in the IPA Dissertation Series since 2013

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. Mamane. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. van den Heuvel. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. van der Burg. *A Reference Architecture for Distributed Software Deployment*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2013-11

D.H.P. Gerrits. *Pushing and Pulling: Computing push plans for disk-shaped robots, and dynamic labelings for moving points*. Faculty of Mathematics and Computer Science, TU/e. 2013-12

M. Timmer. *Efficient Modelling, Generation and Analysis of Markov Automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-13

M.J.M. Roeloffzen. *Kinetic Data Structures in the Black-Box Model*. Faculty of Mathematics and Computer Science, TU/e. 2013-14

L. Lensink. *Applying Formal Methods in Software Development*. Faculty

of Science, Mathematics and Computer Science, RU. 2013-15

C. Tankink. *Documentation and Formal Mathematics — Web Technology meets Proof Assistants*. Faculty of Science, Mathematics and Computer Science, RU. 2013-16

C. de Gouw. *Combining Monitoring with Run-time Assertion Checking*. Faculty of Mathematics and Natural Sciences, UL. 2013-17

J. van den Bos. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. Hadziosmanovic. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. Jeckmans. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

C.-P. Bezemer. *Performance Optimization of Multi-Tenant Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04

T.M. Ngo. *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05

A.W. Laarman. *Scalable Multi-Core Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06

J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. Faculty of Science, Mathematics and Computer Science, RU. 2014-07

W. Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2014-08

A.F.E. Belinfante. *JTorX: Exploring Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09

A.P. van der Meer. *Domain Specific Languages and their Type Systems*. Faculty of Mathematics and Computer Science, TU/e. 2014-10

B.N. Vasilescu. *Social Aspects of Collaboration in Online Software Communities*. Faculty of Mathematics and Computer Science, TU/e. 2014-11

F.D. Aarts. *Tomte: Bridging the Gap between Active Learning and Real-World Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2014-12

N. Noroozi. *Improving Input-Output Conformance Testing Theories*. Faculty of Mathematics and Computer Science, TU/e. 2014-13

M. Helvensteijn. *Abstract Delta Modeling: Software Product Lines and Beyond*. Faculty of Mathematics and Natural Sciences, UL. 2014-14

P. Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2014-15

F.W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. Faculty of Mathematics and Natural Sciences, UL. 2014-16

M.P. Schraagen. *Aspects of Record Linkage*. Faculty of Mathematics and Natural Sciences, UL. 2014-17

G. Alpár. *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World*. Faculty of Science, Mathematics and Computer Science, RU. 2015-01

A.J. van der Ploeg. *Efficient Abstractions for Visualization and Interaction*. Faculty of Science, UvA. 2015-02

R.J.M. Theunissen. *Supervisory Control in Health Care Systems*. Faculty of Mechanical Engineering, TU/e. 2015-03

T.V. Bui. *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness*. Faculty of Mathematics and Computer Science, TU/e. 2015-04

A. Guzzi. *Supporting Developers' Teamwork from within the IDE*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

T. Espinha. *Web Service Growing Pains: Understanding Services and Their Clients*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

S. Dietzel. *Resilient In-network Aggregation for Vehicular Networks*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

E. Costante. *Privacy throughout the Data Cycle*. Faculty of Mathematics and Computer Science, TU/e. 2015-08

S. Cranen. *Getting the point — Obtaining and understanding fixpoints in model checking*. Faculty of Mathematics and Computer Science, TU/e. 2015-09

R. Verdult. *The (in)security of proprietary cryptography*. Faculty of Science, Mathematics and Computer Science, RU. 2015-10

J.E.J. de Ruiter. *Lessons learned in the analysis of the EMV and TLS security protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2015-11

Y. Dajsuren. *On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems*. Faculty of Mathematics and Computer Science, TU/e. 2015-12

J. Bransen. *On the Incremental Evaluation of Higher-Order Attribute Grammars*. Faculty of Science, UU. 2015-13

S. Picek. *Applications of Evolutionary Computation to Cryptology*. Faculty of Science, Mathematics and Computer Science, RU. 2015-14

C. Chen. *Automated Fault Localization for Service-Oriented Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-15

S. te Brinke. *Developing Energy-Aware Software*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-16

R.W.J. Kersten. *Software Analysis Methods for Resource-Sensitive Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2015-17

J.C. Rot. *Enhanced coinduction*. Faculty of Mathematics and Natural Sciences, UL. 2015-18

M. Stolikj. *Building Blocks for the Internet of Things*. Faculty of Mathematics and Computer Science, TU/e. 2015-19

D. Gebler. *Robust SOS Specifications of Probabilistic Processes*. Faculty of Sciences, Department of Computer Science, VUA. 2015-20

M. Zaharieva-Stojanovski. *Closer to Reliable Software: Verifying functional behaviour of concurrent programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-21

R.J. Krebbers. *The C standard formalized in Coq*. Faculty of Science, Mathematics and Computer Science, RU. 2015-22

R. van Vliet. *DNA Expressions – A Formal Notation for DNA*. Faculty of Mathematics and Natural Sciences, UL. 2015-23

S.-S.T.Q. Jongmans. *Automata-Theoretic Protocol Programming*. Faculty of Mathematics and Natural Sciences, UL. 2016-01

S.J.C. Joosten. *Verification of Interconnects*. Faculty of Mathematics and Computer Science, TU/e. 2016-02

M.W. Gazda. *Fixpoint Logic, Games, and Relations of Consequence*. Faculty of Mathematics and Computer Science, TU/e. 2016-03

S. Keshishzadeh. *Formal Analysis and Verification of Embedded Systems for Healthcare*. Faculty of Mathematics and Computer Science, TU/e. 2016-04

P.M. Heck. *Quality of Just-in-Time Requirements: Just-Enough and Just-in-Time*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2016-05

Y. Luo. *From Conceptual Models to Safety Assurance – Applying Model-Based Techniques to Support Safety Assurance*. Faculty of Mathematics and Computer Science, TU/e. 2016-06

B. Ege. *Physical Security Analysis of Embedded Devices*. Faculty of Science, Mathematics and Computer Science, RU. 2016-07

A.I. van Goethem. *Algorithms for Curved Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2016-08

T. van Dijk. *Sylvan: Multi-core Decision Diagrams*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2016-09

I. David. *Run-time resource management for component-based systems*. Faculty of Mathematics and Computer Science, TU/e. 2016-10

A.C. van Hulst. *Control Synthesis using Modal Logic and Partial Bisimilarity – A Treatise Supported by Computer Verified Proofs*. Faculty of Mechanical Engineering, TU/e. 2016-11