Supervisory controller synthesis for timed automata

van der Graaf, P.

Award date:
2019

Technische Universiteit
Eindhoven
University of Technology

De Rondom 70, 5612 AP Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Supervisory Controller Synthesis for Timed Automata

Report - Graduation phase II
Academic year 2019-2020, Quartile 1

Author/ ID:	P. van der Graaf	0890738
Thesis supervisors:	A. Rashidinejad, MSc	
	dr.ir. M.A. Reniers	
Master/ Department:	Mechanical Engineering	
Research Group:	Control Systems Technology	
CST Number:	CST2019.078	

TU/e - Mechanical Engineering
Eindhoven, 4 oktober 2019

Contents

Abbreviations

DES - Discrete Event System
DFA - Deterministic Finite Automaton
DMTA - Deterministic Markable Timed Automaton
DSSTA - Direct Supervisory Synthesis of Timed Automata
EFA - Extended Finite Automaton
FA - Finite Automaton
MTA - Markable Timed Automaton
QG - Quotient Graph
RA - Region Automaton
REQG - Region Equivalen Quotient Graph
RG - Region Graph
RTDES - Real Time Discete Event System
se-FSA - Set-Exp Finite State Automaton
SCT - Supervisory Control Theory
SSEFA - Supervisory Synthesis of Extended Finite Automata
TA - Timed Automaton
TDES - Timed Discete Event System
TEFA - Time Extended Finite Automaton
TPDS - Time Preemptable Direct Synthesis
TTM - Timed Transition Model
TUDS - Time Uncontrollable Direct Synthesis

Abstract

Discrete event systems (DES) are systems with discrete states in which transitions are only based on instantaneous events. In order to restrict the behavior of DES to within a given set of specifications, Supervisory Control Theory (SCT) has been developed for DES. In DES, the time at which events may occur is neglected. However, it is important for many real applications to consider timing in the system. Therefore, we are interested in DES extended with real time, real-time discrete event systems, modelled as timed automata (TA).

There are two problems with applying synthesis to TA. The first problem is that the underlying state space of TA is infinite. This is the result of the dense nature of the clock variables that are used to model time in TA. Therefore, the original DES SCT cannot be applied directly to TA since it only works for automata with a finite set of states and a finite set of transitions. Existing methods try to solve the problem by transforming a TA into a finite automaton (FA) through abstraction of exact time values. The problem that results is state space explosion. Secondly, the introduction of time results in problems that may only be solvable by allowing the supervisor to force events to occur at certain points in time. To tackle these problems, existing methods use the notion of forcible events from SCT of Timed Discrete Event Systems (TDES) and apply it to the time abstracted model. The problem is that there seems to be no known method to reconstruct the TA supervisor from the FA supervisor that results from TDES synthesis, only for DES synthesis.

To tackle the problem of including forcible events in TA, we introduce the idea to not only adapt guards of TA as a means of control, but also adapt the invariants to model the states that become unreachable due to forcing. Invariants model the time that a TA can be in a state such that states that do not satisfy them, simply do not exist. This provides a link between the invariants and the notion of forcibility that is used in TDES synthesis. We show that adapting invariants as well as guards can provide a more permissive FA supervisor, by applying an existing time abstraction approach to the well known Bus-Pedestrian example and comparing the results of DES and TDES synthesis. Next, we show a method to reconstruct the TA supervisor from the FA supervisor that results from synthesis. This approach works for both DES and TDES, under the assumption that the TA supervisor may use a less restricted set of expressions than the input TA. We refer to this abstraction, synthesis and supervisor reconstruction as the indirect synthesis approach.

Finally, we tackle the problem of state space explosion by means of new algorithms that can directly apply synthesis on a TA. Two direct synthesis approaches are proposed, Time Uncontrollable Direct Synthesis (TUDS) and Time Preemtable Direct Synthesis (TPDS). Only TPDS allows forcible events and invariant adaptation. Both direct synthesis approaches are based on an existing synthesis approach for extended finite automata (EFA), and extend it to take into account the continuous time evolution of the clock variables in each location. We show that TUDS gives the same results as indirect DES synthesis and that for a less restrictive set of invariants and guards, TPDS gives the same results as indirect TDES synthesis.

1 Introduction

Before defining the problem that is investigated in this project, we provide an informal summary of the background theory and literature on the topic of supervisory control for systems that include timing information, such that a well founded goal can be formulated.

1.1 Discete Event Systems & SCT

A great number of systems fall into the catagory of discrete-event systems (DES). DES are systems with a discrete set of states in which transitions only occur based on asynchronous occurrences of discrete events [1]. For example, a vending machine or a cash machine, which change state through the occurrence of an asynchronous event such as the push of a button. One of the frameworks that is used for modelling DES is finite automata (FA) [2]. FA have been extended to include discrete variables in extended finite automata (EFAs) [3]. EFAs represent the state of a system via locations and variables. Between the locations, there are transitions labelled with guards, events, and updates to describe the (discrete) dynamics [4]. Both FAs and EFAs are used quite often for modelling DES, compared to other formalisms, as they are the most intuitive ones [5].

The model of an uncontrolled system, represents all of its possible behaviour. For many applications, it may be necessary to limit their behaviour such that it is safe and in accordance with what is desired. For example, in a train-gate controller system, it is not safe for the gate to be up while the train is on the crossing. Some type of control is needed to restrict this behaviour [4]. For the purpose of control, Supervisory control theory (SCT) has been developed by Ramadge & Wonham in [6]. SCT is a model-based theoretical framework for synthesizing a supervisor, that restricts the behaviour of a given plant towards a given specification [5]. Additionally, a maximally permissive result is pursued, i.e. a supervisor should only restrict behaviour when it is absolutely necessary.

1.2 Real-Time Discrete Event Systems

Time does not play a role in DES, as a change in state can only occur due to the occurrence of a discrete event. However, it is not always sufficient to solely consider the behaviour of discrete events without taking into account the time at which these events occur. For example, the control systems of aeroplanes, or train gates crucially depend upon real-time considerations. We will refer to such systems as real-time discrete event systems (RTDES) as in [7]. According to [7], "Real-time discrete event systems (RTDES) are systems for which, in addition to the correct order of events, constraints on delays separating certain events must be satisfied." To model RTDES, two main approaches have been proposed:

1. Discrete-time models
2. Dense-time models

In discrete-time models, time is represented by discrete events with a fixed time unit while, in dense-time models, time is modelled by means of real-valued clocks. Both methods are briefly explained below.

1.2.1 Discrete-Time Models

In discrete-time models, also known as timed DES (TDES), the discrete event *tick* is used to represent a tick of a global clock in the system [8]. Instead of considering real time, they choose to measure time with a global digital clock that maps real time to integer time. This sampling of time by fixing the smallest measurable time unit, keeps the state space of the model finite. To model TDES, the typical DES model, is augmented with time bounds for each event [9]. Such a model is referred to as a Timed Transition Model (TTM) [10]. However, representing each tick of the clock by an event *tick*, makes the model very sensitive to changes in the choice of time unit. This makes it hard to model systems with a broad range of timing constraints, as the state space can quickly become very large. Additionally, adding timing constraints to each event, to create timed events, limits the expressiveness as each event can only have one lower and one upper limit.

A different approach to discrete-time modelling is the Timed Extended Finite Automata (TEFA) model, as defined in a more recent study [5]. In this work, the EFA is extended with discrete valued clocks and time is modelled implicitly. The TEFA is then converted to a tick-EFA, which models time by introducing self loops with *tick* events

in the locations. This proves to be a much more compact discrete-time model than the TTM, which models each state after each tick event. It is also more expressive, as clock constraints on transitions between states are used instead of time bounds on each event.

1.2.2 Dense-Time Models

The second option to include timing in a DES is to use dense-time models, Alur and Dill presented Timed Automata (TA) [11] for this purpose. Such models use a finite set of locations that represent discrete states and a finite set of real-valued clocks to include the timing behaviour [9]. The introduction of clocks, makes TA much more expressive than the TDES [10]. The reason is that these clocks are event-independent, such that they allow multiple and different constraints on the transitions between the locations. Additionally, they provide a more compact graphical model, as time evolution is not shown explicitly.

1.3 Synthesis Approaches for RTDES

There are two main approaches to synthesize a supervisor for RTDES; supervisory control theory (SCT) in control engineering and reactive synthesis in computer science. According to [9], the seminal work on theory of supervisory control in DES was pioneered by Ramadge and Wonham [12]. Therefore, the term RW-framework is often used to refer to SCT. In reactive synthesis, the approach is to see synthesis as a game between the environment and the system [13]. Taking into account the two models for RTDES, three main synthesis approaches will be discussed: Discrete-time SCT, Dense-time SCT and Dense-time reactive synthesis.

In all of these approaches, a distinction is made between controllable and uncontrollable events. An example of an uncontrollable event is pressing a button, a controllable event would be that a light can be switched on as a result. Controllable events can be disabled by the supervisor and uncontrollable events cannot.

1.3.1 Discrete-Time SCT

The RW-framework has been extended for TDES by using a discrete-time model [14]. Timed Transition Graphs (TTG), which are based on TTM, are used as an input for the synthesis approach. The difference with respect to other DES SCT approaches is that TDES SCT can influence the controllability of tick events. This is done by introducing a third type of event, the forcible event. Forcible events allow the possibility to preempt a tick of the clock by leaving the current state before the tick occurs. The main benefit of this approach is that the supervisor can be found by using an algorithm of the same complexity as in DES [9], but with the ability to solve a broader range of problems. The main problem is still the large state space of the tick models. To reduce the large state space increase due to the tick events, Pohari & Wonham introduced a reduction method in 2003 [15]. Their approach was to round non-integer lower and upper time bounds down or up respectively. The problem is that the resulting supervisor is not maximally permissive. Therefore, it is a trade-off between maximal permissiveness and computational complexity.

A more recent study on the topic of supervisory control of TDES has been done in [5]. They present a framework, where for a given set of TEFA, the supervisor is computed using Binary decision diagrams (BDDs) and represented in a modular manner based on the computed logical constraints. They also proposed a method to eliminate the tick events while still obtaining the same behaviour. This is used to tackle the sensitivity of the state size of the *tick* models used in [14] to the clock frequency. This is done by acknowledging that in a sequence of tick events, which must eventually occur, all subsequent states are reachable. This approach has a number of benefits with respect to the original one devised by Brandin & Wonham. Firstly, it reduces the computational complexity by eliminating the *tick* events and using BDDs to compute a supervisor. Secondly, as the resulting supervisor is modular, it has an easy structure to maintain and adjust.

1.3.2 Dense-Time SCT

As previously mentioned, TA are the most natural and expressive way to model RTDES. However, the real valued clocks in TA result in an infinite state space. Since the RW-framework only applies to finite state automata, it is not possible to apply a SCT-based synthesis approach to TA. Therefore, the TA must be abstracted to an FA. Alur and Dill introduced region graphs and later zone graphs [11] for model verification. As explained in [16], zone graphs do not contain enough information to compute a proper supervisor, region graphs do.

According to [9], Wong-Toi & Hoffmann were the first to investigate supervisory control of RTDES by modelling the plant and specification as timed automata [17]. The most important concepts that they introduce are their versions of *untiming* and *timing*. The first of these converts a trace of a timed automaton into a region graph trace and the latter converts a trace of a region automaton into a set of timed automaton traces[1]. The synthesis procedure that they discuss consists of four main steps:

1. compose the language of the specification and the plant,
2. untime this language,
3. solve the untimed supervisory control problem,
4. extract a TA supervisor by timing the untimed supervisor.

In this approach, it is assumed that the supervisor may only enable or disable events as in DES. The main problem is the complexity of constructing a region graph from a TA, as it drastically increases with the number of clocks and the maximum clock values.

To reduce the complexity, an alternative approach was introduced, to transform a TA into a minimal and equivalent Set-Exp finite state automaton (se-FSA) [18] [7] [19] [16]. They introduce a new event *Set*, which is used to set or reset a clock timer and the event *Exp* to indicate when this clock has expired. The synthesis procedure consists of combining the plant P and specification S into one TA, describing this timed automaton PS as an se-FSA and then computing a supervisor based on the TDES procedure in [14]. The benefit of this is that a broader range of problems can be solved with respect to the DES-based supervisory control in [17]. In terms of state space reduction, two major improvements can be seen with respect to the region graph abstraction approach. Converting a TA to an se-FSA produces a smaller state space than the same conversion to a region graph. Additionally, state space explosions due to the value of the largest clock constraint constant are prevented, as a larger value does not result in a larger number of regions [18]. However, the resulting supervisor is an se-FSA, which requires an alternative control architecture. They have not yet found a method to transform an se-FSA back into a timed automaton [16].

1.3.3 Dense-Time Reactive Synthesis

A method that differs from the above is the game theoretic approach of [20], by opting for a state-oriented model instead of the language-oriented approach of Ramadge and Wonham. This is referred to as a reactive synthesis approach as it is based on a two-player game between the program and the environment. In such games, there is a distinction between two control objectives: reachability and invariance. The former refers to finding a strategy that guarantees that a set of given states is eventually reached and the latter refers to remaining in a given set of states. Invariance can also be formulated as a safety objective, where it is guaranteed that a given set of bad states is never reached. The plant is modelled by means of timed game automata (TGA), which are timed automata with discrete transitions marked as controllable or uncontrollable. When it is necessary, abstractions to zone graphs [11] are used. What makes this method interesting, is that they do not simply solve the discrete synthesis problem after completely discretizing the timed automaton to a finite state automaton. Instead, their method works directly on timed automata, only applying discretization where it is necessary to solve the control problem. However, according to [9] the method is still complex due to the zone abstraction. The most well-known model verification tool, UPPAAL, also uses a TGA-based approach for the purpose of synthesis [21]. The problem with these approaches is that the goal is to find any strategy that satisfies the specifications, not necessarily the maximally permissive one [5].

1.4 Literature Overview

An overview of different approaches to model RTDES and to synthesize a supervisory controller for them, is shown in Table 1.1.

[1] It should be noted that the region graph that Wong-Toi & Hoffmann compute is augmented with τ events that represent transitions between the clock regions. This is necessary for the *timing* operator, as it provides the possibility of reconstructing the timed language of an untimed trace.

Table 1.1: *Classification table - Supervisory controller synthesis approaches for RTDES*

Source(s)	Synthesis approach	Modelling approach	Time-Abstraction	Tool(s)
[14], [15] Brandin & Wonham 1994-2003	SCT	Discrete-time (TDES)	Fixing time unit (tick) (TTG)	TTCT [22]
[23] Cofer & Garg 1996	SCT	Discrete-time (TDES, remote events only)	max-algebra model of Timed Event Graph	-
[5] Miremadi 2015	SCT	Discrete-time (TEFAs)	*tick*-eliminated models	Supremica [24]
[17] Wong-Toi & Hoffmann 1991	SCT	Dense-time (TA)	untime: TA to region graph	FlySynth [25]
[18], [7], [19], [16] Khoumsi 2002-2010	SCT	Dense-time (TA)	SetExp FSA	SEATool [16]
[20], [26] Maler 1995-1998	reactive (game)	Dense-time (TGA)	zone-graph	UPPAAL -tiga [21]

Based on Table 1.1, it can be concluded that no studies on the topic of direct synthesis of TA have been conducted. Additionally, it can be seen that a number of tools are linked to the different approaches. However, it is not clear from this table what these tools are capable of. Therefore a more tool specific overview is required, which is given in Table 1.2.

Table 1.2: *Tool classification table*

Tool	Plant	Specification	Time Abstraction	Synthesis approach	Output
CIF3 [27]	EFA, TA, hybrid	requirement automaton	-	SCT (DES)	Max. Permissive supervisor (EFA)
TTCT [22]	TDES	TDES	TTG	SCT (TDES)	?
FlySynth [25]	controllable graph (CG)	temporal logic (invariance or inevitability)	-	On The Fly	any supervisor (decision points)
UPPAAL-tiga [25]	NTGA	TCTL	quotient graphs	On The Fly	Any Winning strategy
SEATool [16]	TA	TA	se-FSA	SCT (TDES)	Max. Permissive supervisor se-FSA
Supremica [24], [5]	TEFA	TEFA	tick-EFA	SCT (TDES)	Max. Permissive modular supervisor (Adapt guards)

In addition to the tools that were shown in Table 1.1, CIF3 is added to the tools of interest as the first entry of the

table. The reason is that it is familiar within the Control Systems Technology group at Eindhoven University of Technology. The plant and the specification are modelled as EFA and there is the possibility to synthesize a maximally permissive EFA supervisor from these models. It also provides the possibility to model timed and hybrid automata, but it does not have the ability to abstract these models for the purpose of synthesis. Additionally, CIF3 allows for interaction with a scalable vector graphics file (svg), which is very useful for the purpose of simulation and presentation.

According to [9] the work of Brandin & Wonham [14] has been implemented in the TTCT tool. Based on this information, it is assumed that the plant and specification are modelled as TDES, the time abstraction is provided by a timed transition graph (TTG) and the synthesis approach falls under the category of TDES SCT. If the BW-theory is applied properly, a maximally permissive supervisor should result, but this is not clear from the literature on the tool.

The on the fly approach that is used in FlySynth [25], takes controllable graphs (CG) as an input and produces a controller that is not necessarily maximally permissive. The output is a list of decision points, which are vertices of a CG and lists of controllable edges [25]. It does not perform the abstraction from a (controllable) timed automaton to a CG itself, the tool Kronos (Minim) is used for this purpose by means of a time-abstracting bisimulation graph [28].

In [21], the functionalities of UPPAAL-tiga are described. The plant is modelled by means of a network of timed game automata (NTGA). The specification is a set of winning conditions, either related to reachability or to safety, which is modelled as a subset of Timed Computational Tree Logic (TCTL). The result is a winning controller strategy. They claim that this is the only efficient tool to apply synthesis to control problems modelled as Timed Game Automata [21].

As in the case of the TTCT tool, there is not much specific information on the SEATool. In [16] it is stated that SEATool implements SetExp, which is the time abstraction approach. If the theory has been fully implemented, then the tool should use TA as a means to model the plant and specification, compute the synchronous product, determine the se-EFA and produce a maximally permissive se-EFA supervisor using TDES SCT.

It is preferred to have a tool that is able to model timed automata, to abstract them and that can produce a maximally permissive supervisor in the form of an automaton or preferably a timed automaton. It can be seen that both CIF3, and SEATool have the potential to provide this option, where it should be taken into account that the abilities of SEATool have been assumed based on the provided literature. Therefore, it may prove to be useful to take into account the structure of existing algorithms for DES synthesis when researching synthesis of RTDES.

1.5 Problem statement

Based on the literature review, a number of conclusions can be drawn. First, the game theoretic approach to synthesis has already been worked out extensively in UPPAAL-tiga. Next to that, a well founded TDES approach has been implemented in Supremica. This still suffers from the size of tick models, but already shows strong improvements with respect to the original work of Brandin & Wonham. The main problems seem to lie with the dense time synthesis approaches. The region graph based approach of Wong-Toi suffers from major state space explosions, due to its sensitivity to the number of clocks and the maximum values of those clocks. Additionally, as it is based on DES SCT, time delays are always considered uncontrollable. Therefore, a smaller number of synthesis problems can be tackled than when using TDES SCT. Finally, the implementation in FlySynth, is restrictive as it does not consider the full state space. The other dense-time approach, using the SetExp transformation, does use TDES SCT notions, but it cannot provide a supervisor in the form of a TA. Additionally, the state space increase due to the transformation from TA to se-FSA, is still sensitive to the number of clocks. Therefore, a state space explosion can still occur.

From this, two main problems can be derived:

1. for dense-time SCT approaches: no method exists to adapt TDES SCT to TA, and reconstruct a TA supervisor via a transformation FA$\rightarrow$TA,
2. state space explosions result due to time abstraction.

Therefore, the problem statement is defined as follows:

(1) Find a means to apply TDES SCT notions to TA, such that the problems that can be tackled with TA supervisors become broader and (2) investigate a method that directly applies synthesis to TA, such that computationally complex abstraction approaches are no longer necessary.

Additionally, the possibility to implement such methods in CIF3 is considered.

1.6 Outline

In Section 2, FA are defined formally, followed by an introduction of the graphical representation and formal notation of the TA that will be used to model the uncontrolled behaviour in this project. This is followed by an explanation of SCT for FA and TA in Section 3, where the notion of foricble events from TDES is linked to TA. After having introduced the background concepts, we first consider an indirect synthesis approach in Section 4, along the lines of the region abstraction that is used in [17]. Here, both DES and TDES SCT approaches are considered and an alternative derivation of the TA supervisor is proposed. Next to that, implementation of the first step of the indirect approach in CIF3 is discussed. Next, two direct synthesis approaches are proposed in Section 5, one only allowing guard adaptation and the other allowing both guard and invariant adaptation. Finally, the conclusions and recommendations are discussed in Section 6.

2 Modelling Frameworks

In order to provide a clear build up to the extended timed automaton that is used in this project, both finite automata and timed automata are discussed in this section.

2.1 Finite Automata

First, a formal definition of finite automata is given, followed by the definition of blocking, determinism and the synchronous product.

Finite Automata (FA) are formally defined as follows [2][2].

Definition 2.1 – Finite Automaton: a finite automaton is a 5-tuple $(L, E, \rightarrow, L^m, l^0)$ where

- L is a finite set of locations,
- E is a finite set of events (also referred to as alphabet),
- $\rightarrow \subseteq L \times E \times L$ is the transition relation, i.e. given the occurrence of an event $e \in E$, a transition from a source location $s \in L$ to a target location $t \in L$ is given by: $s \xrightarrow{e} t$. Such a transition is also referred to as an edge. A number of subsequent transitions leads to a word $w \in E^*$, which is a concatenation of events from the finite set of E-events. The set of all possible words is referred to as the language $\mathcal{L}(A)$ over an automaton A. A location l is considered reachable from the initial location l_0 if it can be reached after zero or more transitions.
- $L^m \subseteq L$ is a set of marked locations[3],
- $l^0 \in L$ is the initial location.

It should be noted that often, the term state q is used instead of location l as the state of an FA only consists of its current location. However, as variables will play a role in TA we choose to refer to states as pairs (l, u) that consider the location l and the values of the variables u. This is in line with the distinction that is made for extended FA (EFA), which include discrete-valued data variables [3].

The modeller decides which locations are marked, which is usually a location that is considered to be stable. For instance, a model of a vending machine could have a marked location *Idle* whilst it waits for a customer to insert a coin. The purpose of these marked locations is to determine if the automaton is non-blocking. This is the case if none of its reachable locations are blocking.

Definition 2.2 – Blocking location: a location l_1 is blocking if there does not exist a sequence of events $w \in E^*$ and a location $l_2 \in L^m$ such that $l_1 \xrightarrow{w} l_2$.

A location from which a marked location can be reached is considered a non-blocking location. Reaching marked locations can be interpreted as achieving intermediate steps in a process, thereby indicating that progress is being made. A possible reason for the prevention of progress is deadlock. Deadlock exists in a location without any outgoing edges, such that it can never be left.

Furthermore, a distinction should be made between non-deterministic and deterministic automata. Deterministic finite automata (DFA) allow transitions to one and only one location from each location [2], for a given event. Effectively, this means that the next location can always be determined after the execution of that event.

Definition 2.3 – Deterministic Finite Automaton: a finite automaton, is called a deterministic finite automaton (DFA), if for each location $l \in L$ and event $e \in E$ of that automaton, there exists at most one outgoing edge, labelled with the same event e.

A simple system may only need to be described by one automaton. However, in more complex systems it is desirable to model different parts of that system separately. These separate models may share events, which can only occur synchronously in all automata. To find the model of the global behaviour, the synchronous product of the automata is used.

[2] Note that different symbols are used than in the original work in line with how the Supervisory Control course is tought at the TU/e [4].
[3] In computing science often referred to as *accepting* or *final* states.

Definition 2.4 – Synchronous Product: the synchronous product of two automata $A_1 = (L_1, E_1, \rightarrow_1, L_1^m, l_1^0)$ and $A_2 = (L_2, E_2, \rightarrow_2, L_2^m, l_2^0)$, $A_p = A_1||A_2$ is the automaton $A_p = (L_1 \times L_2, E_1 \cup E_2, \rightarrow_p, L_1^m \times L_2^m, l_1^0 \times l_2^0)$, where each transition in $\rightarrow_p$ is defined by:

- for event $e \in E_1 \cap E_2$, $\rightarrow_p$ contains $\langle (s_1, s_2), e, (t_1, t_2) \rangle$ for every $\langle s_1, e, t_1 \rangle$ in $\rightarrow_1$ and $\langle s_2, e, t_2 \rangle$ in $\rightarrow_2$.
- for event $e \in E_1 \setminus E_2$, $\rightarrow_p$ contains $\langle (s_1, s_2), e, (t_1, s_2) \rangle$ for every $\langle s_1, e, t_1 \rangle$ in $\rightarrow_1$ and $s_2 \in L_2$.
- for event $e \in E_2 \setminus E_1$, $\rightarrow_p$ contains $\langle (s_1, s_2), e, (s_1, t_2) \rangle$ for every $\langle s_2, e, t_2 \rangle$ in $\rightarrow_2$ and $s_1 \in L_1$.

The synchronous product can be applied to more than two automata in a similar fashion.

2.2 Timed Automata

As a starting point, Alur's Timed Automata (TA) [29] are used. Such dense-time models use a finite set of locations that represent discrete states and a finite set of real-valued clocks to include the timing behaviour [9]. First, we introduce the graphical notation of the TA, extended with marked locations for the purpose of supervisory controller synthesis. The "simple light control" example of [30] is used for this purpose. This is followed by a formal definition of the extended TA and its semantics. Next, a comparison between TA and FA notions such as non-blockingness and deadlock is made including an introduction of notions that only play a role in TA. Finally, the synchronous product for TA is defined formally.

Example 2.1 – Simple light control: Consider the case where a light should turn on when the button is pressed and it should burn brighter when the button is pressed twice within three seconds. If it takes longer than three seconds, it should turn off. In addition to the original example, progress should be enforced by restricting the maximum time delay in the LIGHT location to ten seconds, after which it should turn off. The initial location OFF is considered stable and therefore it is modelled as a marked location. This is represented by a location with a double circle.

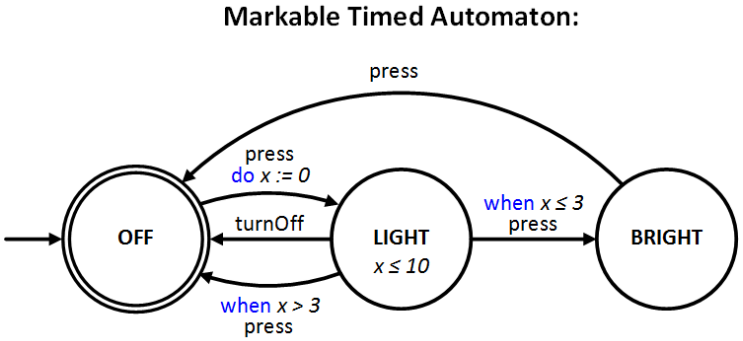


Figure 2.1: *Bright light example [30], Timed automaton with marked locations.*

It is clear from Figure 2.1 that the evolution of time is not modelled explicitly. Nonetheless, the value of the clock x increases at a rate of $x' = 1$ in each location. By using constraints on the value of this clock, the transition to the BRIGHT location is limited to within three seconds after entering LIGHT. However, as guards are enabling conditions and not enforcing ones, they cannot enforce progress. In the bright light application this may not be a problem, but it will be in production applications. Therefore, Alur uses the concept of location invariants in [29] to determine how long an automaton may remain in a location. In this case, the local timing constraint $x \leq 10$ is used to indicate that the LIGHT location must be left within ten seconds. The result is a transition back to OFF at $x == 10$ at the latest, via the event $turnOff$. This is an example of how progress is enforced from the LIGHT location by means of a location invariant.

At this point, the meaning of marked locations in TA may not directly be clear, due to the time evolution in those locations. Before explaining what the difference is with respect to a marked location in an FA, the formal definition of the markable TA (MTA) and the semantics of the underlying transition system will be given.

Definition 2.5 – Markable Timed Automaton: a markable timed automaton is a 7-tuple $(C, L, E, \rightarrow, L^m, l^0, I)$ where

- C is a finite set of clocks $x \in \mathbb{R}_{\geq 0}$ that can be defined locally and read by other automata,
- L is a finite set of locations,
- E is a finite set of events,
- $\rightarrow \subseteq L \times B(C) \times E \times 2^C \times L$ is the set of edges (transition relation)[4],
- L^m is the set of marked locations,
- $l_0 \in L$ the initial location,
- $I : L \mapsto B(C)$ assigns invariants to locations.

For the formal representation of the simple light control example, see Appendix A. For the MTA that are used to model the uncontrollable behaviour, the set of permitted clock constraints $B(C)$ as invariants or guards are taken over from [29], the grammar is given by:

$$\varphi := x \leq n \mid n \leq x \mid x < n \mid n < x \mid \varphi_1 \wedge \varphi_2. \tag{2.1}$$

Here, $x \in C$ is a clock and n a rational number in $\mathbb{Q}$ to compare it with by means of a clock assignment[5]. A clock assignment is defined as a mapping u from the set of clocks C to their values in $\mathbb{R}_{\geq 0}$ [31]. Additionally, we assume that each constraint in an invariant is of the form $x \sim n$, with $\sim \in \{<, \leq\}$, which is not restrictive when considering guards that follow the grammar in Equation 2.1 [16].

Every TA has an underlying transition system consisting of an infinite number of states. These states are pairs of the form (l, u), where l is the location and u is the set of clock assignments. Each state can evolve via an elapse-of-time transition or a location-switch transition. The former is the result of a time delay Δ as long as all clock values satisfy the location invariant. The latter is the result of an event e, which can occur if the clock values satisfy the timing constraints on the edges. Additionally, a clock reset can occur, e.g. $x := 0$ as shown in the transition between the OFF and LIGHT location of Figure 2.1. The value to which the clock is reset is always zero, extensions such as updatable TA [32] are not considered here. Additionally, the initial valuation of the clocks in the initial location is assumed to be zero.

Definition 2.6 – MTA semantics: the semantics of an MTA (equal to the TA semantics in [31]) are a timed transition system where states are pairs (l, u) and transitions are defined by the following rules:

- time delay: $(l, u) \xrightarrow{\Delta} (l, u + \Delta)$ if $u \in I(l)$ and $(u + \Delta) \in I(l)$ for a non-negative real $\Delta \in \mathbb{R}_{\geq 0}$
- location switch: $(s, u_s) \xrightarrow{e} (t, u_t)$ if $s \xrightarrow{g,\, e,\, r} t, u_s \in g, u_t = [r \mapsto 0]u_s$ and $u_t \in I(t)$

Note that l is substituted by s and t in the location switch transition, to represent source and target locations respectively. Additionally, a distinction is made between u_s and u_t to represent the current value of the clocks in the source and target states respectively. The notation $u_s \in g$ indicates that a location switch is only possible when the clock values u_s in state (s, u_s) satisfy the guard g of the edge between location s and t. The notation $u_t = [r \mapsto 0]u_s$ shows that the target clock values u_t are the same as the source clock values u_s apart from the values of the clocks in r, which are reset to zero. Finally, $u_t \in I(t)$ indicates that in addition to satisfying the guards, the target clock values u_t mus also satisfy the target invariant. Example 2.2 is used to visualize the underlying infinite state transition system of an MTA.

Example 2.2 – Underlying transition system: Take a simple MTA with the event a, protected by a clock constraint $x \geq 2$ that models a delay of two time units before the transition can be taken. The location invariant $x \leq 3$ forces the transition at $x == 3$.

[4] $2^C = P(C) = \{0, 1\}^C$ is the power set of C, i.e. if $C = \{x, y, z\}$ then $2^C = \{0, 1\}^C = \{\emptyset, \{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x, y, z\}\}$, here it represents the possible combination of clocks for which the value should be reset to zero.

[5] Clock interpretation in [29], Clock assignment in [31]

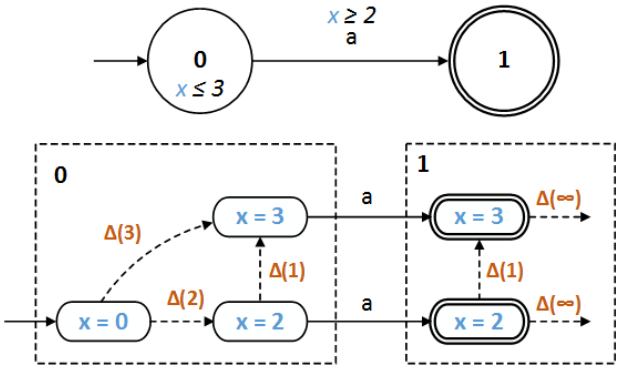


Figure 2.2: *Underlying infinite transition system of an MTA*

Figure 2.2 shows how the underlying infinite transition system of the MTA is imagined. Based on the constraints, the most relevant states of the TA are shown. The blue numbers refer to the value of clock x in state (l_i, u_i) and the dashed squares in which the state is positioned indicates which location l_i belongs to that state with $i = 1, 2$. In accordance with the semantics, either a time delay Δ or an event a leads to a transition to another state. Note that for the clarity of the figures, we do not include self looping delays $\Delta(0)$ in each state. Additionally, no state $(l_0, x > 3)$ exists as a result of the location invariant. In between states that are separated by a delay, there are actually an infinite number of states as a result of the infinite number of possible delays. Finally, as location l_1 has no time bound, time can delay forever as indicated by the $\Delta(\infty)$.

To keep the model simple, it is assumed that all possible values of a clock (i.e. satisfying the location invariants) in a marked location are considered marked. Therefore, a marked location is a location for which all possible states in the underlying transition system are marked. This can also be seen in Figure 2.2, where all states in the transition system within l_1 are marked.

As for FA, an MTA is non-blocking if none of its reachable locations have blocking. However, a marked location can now be reached not only directly through events, but also via time delays that lead to satisfaction of guards that bound clocks from below. Conversely, time delays can also lead to dissatisfaction of guards that bound clocks from above. First, we give a definition of a blocking state and based on that the definition of a blocking location.

Definition 2.7 – Blocking state (TA): a state (l_1, u_1) in the underlying transition system of an MTA is blocking if there does not exist a path from that state, consisting of events $e \in E$ and time delays $\Delta \in \mathbb{R}_{\geq 0}$, to a marked state (l_2, u_2) with $l_2 \in L^m$.

Note that it is sufficient to reach a marked state in the underlying transition system due to the assumption that all states with a marked location are marked.

Definition 2.8 – Blocking location (TA): A location $l \in L$ in an MTA has blocking if any reachable state (l, u) in the underlying transition system is blocking.

The MTA in Example 2.2, is non-blocking because l_1 is marked and each state of l_0 leads to marked location l_1 after delaying for at least two time units and taking event a to l_1. Additionally, states $(0, x > 3)$ simply do not exist due to the invariant $I_0 = x \leq 3$.

The meaning of a deadlock location effectively remains the same as for an FA, i.e. a location from which no discrete transition can be taken, regardless of the time delay [33]. However, timing introduces another interesting concept: timelock.

Example 2.3 – Marked deadlock location with invariant: Take the single-location automaton given in Figure 2.3.

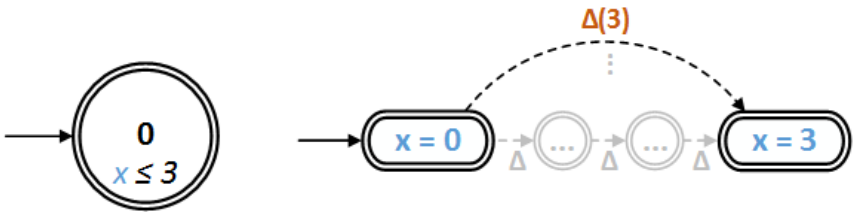


Figure 2.3: *Non-blocking MTA with dead- and timelock location l_0*

Clearly, location l_0 is a deadlock location, since no outgoing events exist. Regardless of the invariant, l_0 is non-blocking, as the underlying transition system shows that only marked states exist. The reason for this is that according to the MTA semantics it is not possible to delay after reaching the limiting value of the invariant. However, the invariant would usually force the automaton out of the location. In this case, this is not possible due to the fact that there is no way out of the location. The direct result is that time can no longer advance to infinity, therefore a timelock exists at state $x = 3$ [28].

Definition 2.9 – Timelock state: a timelock state in the underlying transition system of a TA, is a state from which no paths exist, such that the sum of the delays in that path is infinite.

An MTA is called timelock-free if none of its reachable states in the underlying transition system have timelock. Another problem that is closely related to timelock, is zenoness. The behaviour of a TA is zeno if an infinite number of discrete events can occur in finite time, e.g. a self-looping event a with a possible timed behaviour $(a, 0)(a, \frac{1}{2})(a, \frac{3}{4})...(a, 1 - \frac{1}{2}^n)$. Therefore, deadlock and timelock can also exist separately. Namely, a location with self looping event a is not a deadlock location, but it is a timelock location because an infinite number of discrete events can occur without time passing. More information on this topic can be found in [28]. Both timelock and Zeno behaviour are not considered in this work.

Because a TA is not determinizable in general [34], only deterministic MTA (DMTA) are considered.

Definition 2.10 – Deterministic MTA: a deterministic MTA is an MTA for which the underlying transition system, as defined by the semantics, is deterministic (Definition 2.3).

Consider Example 2.1. Even though there are two of the same outgoing events *press* from the *LIGHT* location, the MTA is deterministic. The reason for this is that the underlying transition system is deterministic as a result of the guards in the MTA. As long as the guards do not allow overlapping clock values, no non-determinism will be present.

The synchronous product is also defined for MTA. It is assumed that all clocks increase in value at the same rate.

Definition 2.11 – Synchronous product of MTA: the synchronous product of two MTA $A_1 = (C_1, L_1, E_1, \rightarrow_1, L_1^m, l_1^0, I_1)$ and $A_2 = (C_2, L_2, E_2, \rightarrow_2, L_2^m, l_2^0, I_2)$, i.e. $A_1||A_2$, under the assumption that C_1 and C_2 are disjunct, is the MTA automaton $A_p = (C_1 \cup C_2, L_1 \times L_2, E_1 \cup E_2, \rightarrow_p, L_1^m \times L_2^m, l_1^0 \times l_2^0, I_p)$, where $I(l_1, l_2) = I(l_1) \wedge I(l_2)$ and each transition in $\rightarrow_p$ is defined by:

- for event $e \in E_1 \cap E_2$, $\rightarrow_p$ contains $\langle (l_{s,1}, l_{s,2}), e, g_1 \wedge g_2, r_1 \cup r_2, (l_{t,1}, l_{t,2}) \rangle$ for every $\langle l_{s,1}, e, g_1, r_1, l_{t,1} \rangle$ in $\rightarrow_1$ and $\langle l_{s,2}, e, g_2, r_2, l_{t,2} \rangle$ in $\rightarrow_2$.
- for event $e \in E_1 \setminus E_2$, $\rightarrow_p$ contains $\langle (l_{s,1}, l_{s,2}), e, g_1, r_1, (l_{t,1}, l_{s,2}) \rangle$ for every $\langle l_{s,1}, e, g_1, r_1, l_{t,1} \rangle$ in $\rightarrow_1$ and $l_{s,2} \in L_2$.
- for event $e \in E_2 \setminus E_1$, $\rightarrow$ contains $\langle (l_{s,1}, l_{s,2}), e, g_2, r_2, (l_{s,1}, l_{t,2}) \rangle$ for every $\langle l_{s,2}, e, g_2, r_2, l_{t,2} \rangle$ in $\rightarrow_2$ and $l_{s,1} \in L_1$.

Note that a marked location in the synchronous product can only result from two marked locations in the original timed automata. The product of more than two automata can be computed by subsequently applying the above definition.

In the following sections, the MTA will be referred to as TA, as it is the only version of timed automata that will be used throughout this report.

3 Supervisory Control Theory

The purpose of this section is to introduce the concepts from supervisory control theory (SCT), extend them for TA and to define the SCT synthesis problems that are considered for both FA and TA.

3.1 SCT of FA

The concept of feedback control, as known from motion control, can be used to model the closed loop behaviour of a supervised discrete-event system.

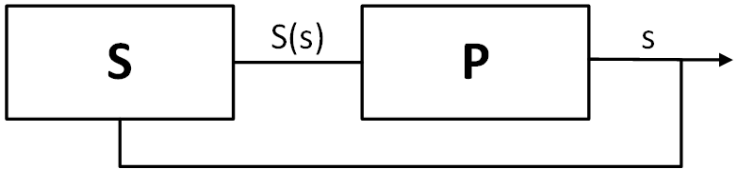


Figure 3.1: *Block diagram, representing the closed loop behaviour of a supervised DES, based on [1]*

Figure 3.1 shows this, where P refers to the plant automaton, which in DES is an FA representing the uncontrolled system. This system executes a sequence of events s, which is observed by the supervisor S. Note that all events are considered observable by the supervisor in this work. Based on the current location of P as a result of s, the supervisor communicates the set of enabled events $S(s)$ that P can execute [1]. Here, S is viewed as a feedback map. A more intuitive way to view the supervisor is as an automaton. According to [35] the supervised plant $P||S$ can then be computed, to describe the control of the supervisor over the plant behaviour.

In systems with multiple plant automata, the synchronous product $P_1||P_2||...P_n$ (Definition 2.4) is used to determine the global uncontrolled behaviour. The global controlled behaviour is then determined by computing the synchronous product of those plant automata with S.

For the purpose of control, a distinction is made between controllable and uncontrollable events. The reason for this is that some events simply cannot be controlled, e.g. the triggering of a simple position sensor in a cylinder. The finite set of events E is therefore split into two disjoint sets E_c and E_u, such that:

$$E = E_c \cup E_u. \tag{3.1}$$

Here,

- E_c is the set of controllable events: the events that can be disabled by the supervisor S.
- E_u is the set of uncontrollable events: the events that cannot be disabled by the supervisor S.

The supervisor is considered controllable if it does not disable any uncontrollable events that are possible from a given location in the plant [3].

Definition 3.1 – Controllability (FA): an automaton S is considered controllable with respect to an automaton P if whenever there exists a sequence of events $w \in \mathcal{L}(P||S)$ and $wu \in \mathcal{L}(P)$, with $u \in E_u$, then $wu \in \mathcal{L}(P||S)$.

In DES the main goal is to synthesize a supervisor S that is controllable with respect to a plant P, for which the supervised plant $P||S$ is non-blocking and that satisfies a set of given specifications. Here, specification language refers to the desired behaviour of the plant. These specifications are modelled as requirement automata R_i, which are added to the product of plant automata to model the global uncontrolled behaviour satisfying the specification language. It is assumed that such requirement automata have the same alphabet or a subset of the alphabet of the plant.

A problem that could result from computing the synchronous product $R||P$, is that the requirement automaton restricts uncontrollable behaviour. It is more difficult to consider both blocking and uncontrollability in synthesis than it is to only consider blocking, therefore it is desirable that the uncontrollability problem is converted to a blocking problem. To achieve this, the requirement automata must be converted to plant automata [36]. We shall refer to these automata as plantified requirements.

Definition 3.2 – Plantified Requirement: a plantified requirement R^p is a requirement that does not restrict uncontrollable events, by introducing a trash location $\perp$ to which all uncontrollable events $e \in E_u$ in the alphabet of R have an edge from each location (including the trash location) that does not have an outgoing edge with that event.

Plantification ensures controllability of plantified requirement R^p with respect to plant P. The following example will help to clarify the method.

Example 3.1 – Plantified Requirement: Consider a plant P that allows transitions to marked location 3, either via event $a \in E_c$ followed by event $b \in E_u$ or vice versa.

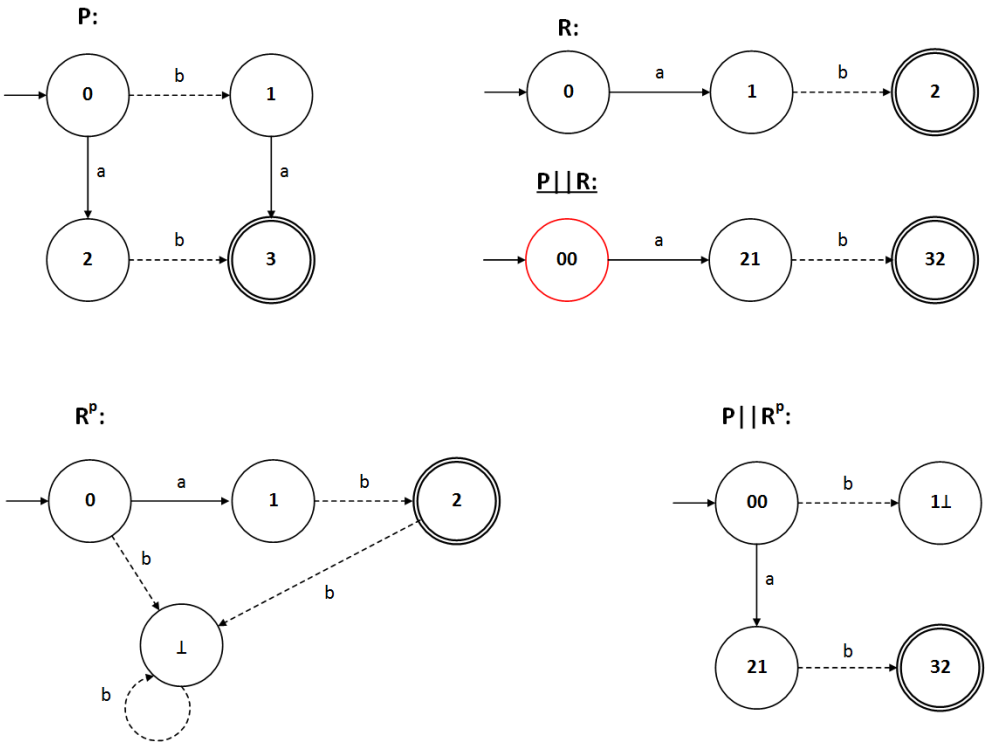


Figure 3.2: *Plantification of a requirement*

If a requirement is modelled, specifying that event a should be executed before b, then according to Definition 3.1, R is uncontrollable with respect to P. This can also be seen in Figure 3.2, where in $P||R$ an event b should be possible from location 00. However, this is no longer the case because location 0 of R does not allow this event to occur. The addition of the trash state in R^p converts the controllability problem to a blocking problem.

In addition to satisfying the requirements, assuring non-blockingness and controllability, the resulting supervisor should be maximally permissive.

Problem 3.1 – FA Supervisor Synthesis Problem: the FA supervisor synthesis problem is defined as follows:

For a given plant P and plantified requirements $R^p = R_1^p||R_2^p||...R_n^p$ modelled as FAs, find a maximally permissive supervisor S that is controllable with respect to $P||R^p$ and for which $S||P||R^p$ is non-blocking.

Note that controllability property is automatically satisfied by making sure that the requirement automata are plantified. The task is to make sure that the supervisor remains controllable with respect to the plant during and after synthesis.

3.2 SCT of TA

As the underlying transition system has an infinite state space, it is not possible to directly apply DES synthesis to a TA. Additionally, we are interested in finding TA supervisors using TDES SCT notions. Therefore, it is necessary to extend the SCT notions for the TA modelling framework. As in the DES case, the uncontrolled behaviour is modelled by the plant P^t, the specifications are modelled by plantified requirements $R^{p,t}$ and the supervised behaviour

is the result of $S^t||P^t||R^{p,t}$. Here, superscripts ^{t} are used to differentiate between timed and finite automata. The differences lie in the introduction of clocks in the requirement automata and in the introduction of new concepts in controllability. Both are discussed below, followed by the main TA supervisor synthesis problem.

Requirements are now modelled as TA, which can locally define their own clocks to impose new timing constraints. Due to the definition of $B(C)$ in 2.5, it is not possible to test or to reset a clock that is not local to the requirement. Additionally, it is not possible to model invariants in the requirements as the uncontrollable time delays Δ cannot be taken into account explicitly in the plantified requirement. As a result, the plantification of a TA requirement requires only one extra consideration compared to FA: uncontrollable edges with guards. This is the same as in the EFA plantification.

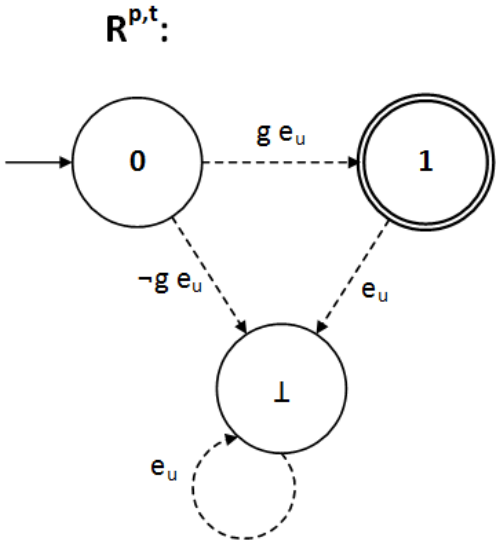


Figure 3.3: *Plantification of a TA requirement with guard g for uncontrollable edge* e_u

Figure 3.3 shows a plantified requirement automaton for which a guard g is specified for the edge with uncontrollable event e_u in the alphabet of the plant. To make sure that no uncontrollable behaviour is restricted, the negation of that guard must be added to a transition pointing to the trash location $\perp$.

The definition of controllability essentially remains the same as in Definition 3.1, i.e. uncontrollable events should not be disabled. The difference is that it now depends on the values of the clocks with respect to the guards of uncontrollable edges if an event is enabled [9].

Definition 3.3 – Controllability (TA): a timed automaton S^t is considered controllable with respect to a timed automaton P^t if it does not restrict any uncontrollable events that are enabled in any reachable state of the underlying transition system of P^t.

The Bus Pedestrian example, as seen in [5], is used to show a problem that arises in supervisory control of TA.

Example 3.2 – Bus Pedestrian Example: Consider a bus that is headed directly for a pedestrian, which will run over him at $x = 2$ time units if he does not move. The pedestrian needs $y = 1$ time unit to realise his fate, after which he has the chance to jump out of the bus's path.

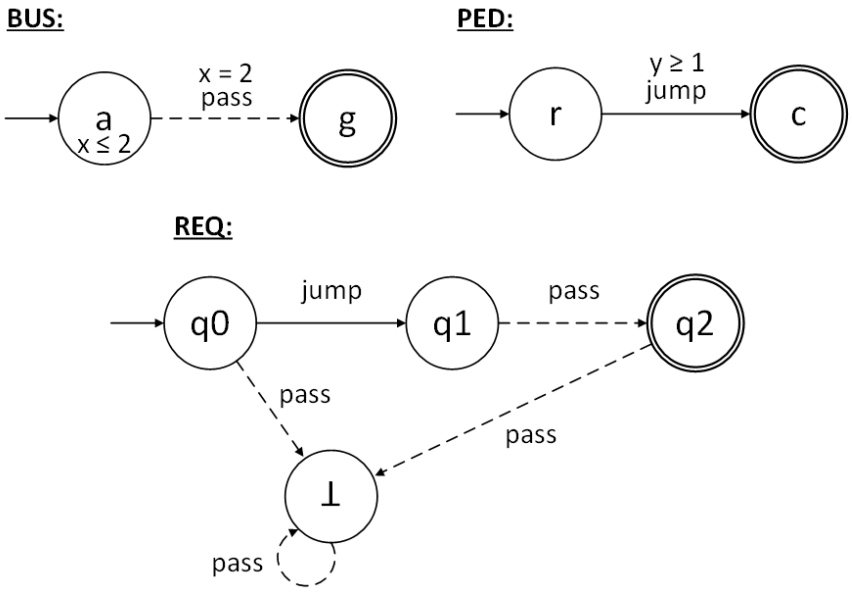


Figure 3.4: *Bus, passenger and plantified requirement automaton.*

Figure 3.4 shows the two plant automata representing the bus and the pedestrian. Additionally, a requirement is introduced. This requirement is meant to ensure the pedestrian jumps before the bus passes. Note however, that a "trash state" $\perp$ is added to the requirement shown as SPEC in [5] to plantify it.

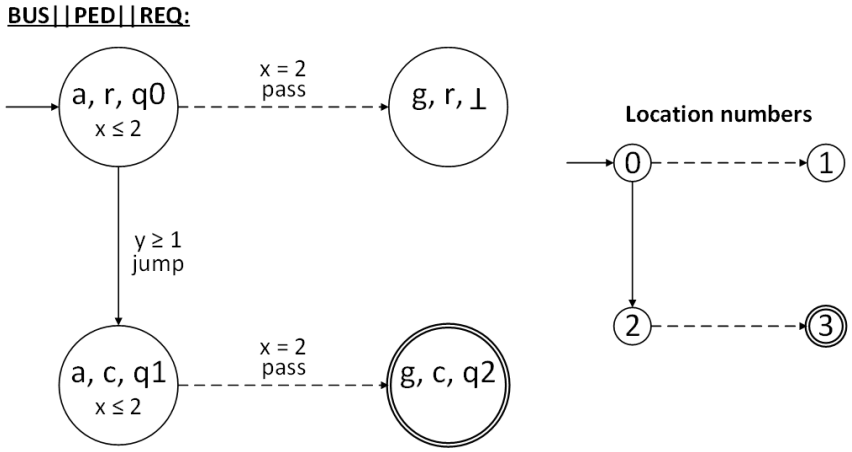


Figure 3.5: *Synchronous product of the three automata and the location numbers for the clarity of further reference.*

The synchronous product (Definition 2.11) of the bus, pedestrian and requirement automata is shown in Figure 3.5. Its underlying transition system is given in Figure 3.6, where the time delays Δ are depicted as uncontrollable events. Note that only the values of clock x are shown in the underlying transition system, as the values of y are the same in all states.

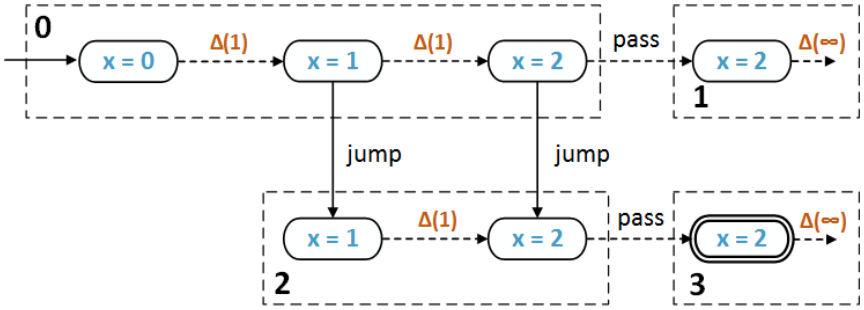


Figure 3.6: *Underlying transition system of* $BUS||PED||REQ$

In Figure 3.5 it can be seen that there is an uncontrollable edge from the initial location to deadlock location 1. The only way to prevent this location from being reached, is to make sure that the pedestrian jumps before $x = 2$. This would mean that the time delay Δ, depicted as an uncontrollable event between state $(0, x = 1)$ and $(0, x = 2)$ in Figure 3.6, would need to be disabled. It is known from DES synthesis of EFA that guards can be adapted as a means of supervisory control [3]. However, only guards of controllable edges can be adjusted to satisfy the controllability property. Because of this, the problem is unsolvable for DES SCT notions of controllability.

In TDES [37], a new category of events called forcible events is introduced as a solution. An example of a forcible event is the event *close*, to close a valve before a tank overflows if the water level sensor is defect. Forcible events can be used to preempt an uncontrollable *tick* of the global discrete clock. In terms of controllability this means that if a forcible event exists, leading to another state, the *tick* event becomes controllable and the supervisor may remove it from the set of possible events in that state. It is evident from figure 3.6, that the underlying transition system of the bus-pedestrian example TA, is very similar to the tick model of the same example in [37]. However, instead of modelling ticks, we use time delays Δ and we imagine an infinite number of possible time delays between the states of interest. Therefore, it is possible to introduce the notion of forcibility to the underlying transition system of a TA.

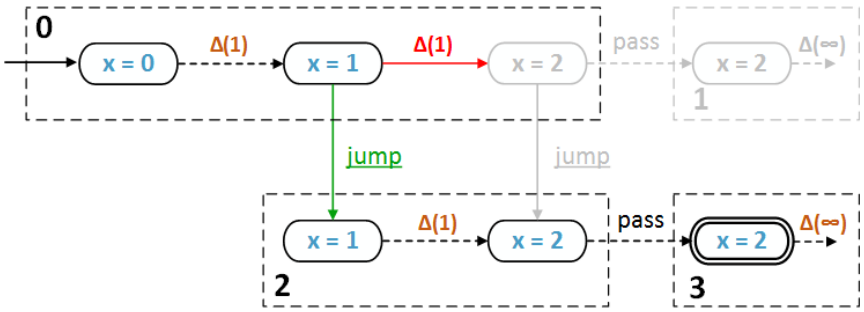


Figure 3.7: *Forcibility of location invariant*

As in [37], we assume that the event $jump$ is a forcible event. From now on forcible events will be underlined in figures for a clear distinction. Figure 3.7 shows that as event $jump$ (depicted green) can be executed before state $(0, x = 2)$, it can preempt any delay other than $\Delta(0)$. Because of this, the delay transition shown in red can be considered controllable. Therefore, it can be disabled in any state $(0, x < 2 \wedge x \geq 1)$. This does not mean that time can be prevented from passing, it simply guarantees that the supervisor can force the plant out of a location before an undesirable state is reached. This idea is clear for tick models as there is a finite number of states and *tick* events. If one of these states is removed by disabling an ingoing *tick* event, for which the source state has an enabled forcible event, it can be seen in the model that this state no longer exists when it is removed due to unreachability. However, when viewing a TA instead of its infinite underlying states, the underlying states cannot be viewed individually. Therefore we need a means to model the remaining states, after the effective removal of the states for which their incoming time delays are preempted via a forcible event. TA already have a solution to this problem, namely, invariants are naturally used to represent states that cannot be reached, or more specifically, simply do not exist in the model. Location invariants can enforce progress by forcing a transition to occur when the maximum clock value is reached. In the case of the bus-pedestrian example TA shown in Figure 3.5, the adaptation of invariant I_0 from $x \leq 2$

to $x < 2$ would represent the disablement of the smallest possible strictly positive time delay Δ in the underlying transition system. The reason for choosing the smallest possible time delay is that maximal permissiveness should be preserved. Strict positivity is mentioned explicitly, as according to the semantics $\Delta \in \mathbb{R}_{\geq}0$, and the zero-valued delay cannot be preempted.

This concept requires some further clarification. Firstly, we impose no restriction on when it is permitted to model invariants in a TA. For instance, there need not be a forcible event out of a location to model an invariant. This assumption is in line with the idea that states beyond the invariant simply do not exist, either because we are not interested in those states or because the plant is physically constrained to work within the given time limits. For example, consider the bus-pedestrian example, but now the event $jump$ is not considered forcible. The invariants in the bus-pedestrian example could still be modelled, as the bus will definitely hit the pedestrian if he does not jump before $x = 2$. The difference is that in this case, there is no way to force the pedestrian to jump before the bus hits him (e.g. no hero to force his jump by pushing him out of the way). Therefore, we do impose a restriction on adapting invariants. Namely, that a forcible event must at least be available before the time delay that must be preempted. This ensures that the adaptation of a location invariant does not introduce uncontrollability.

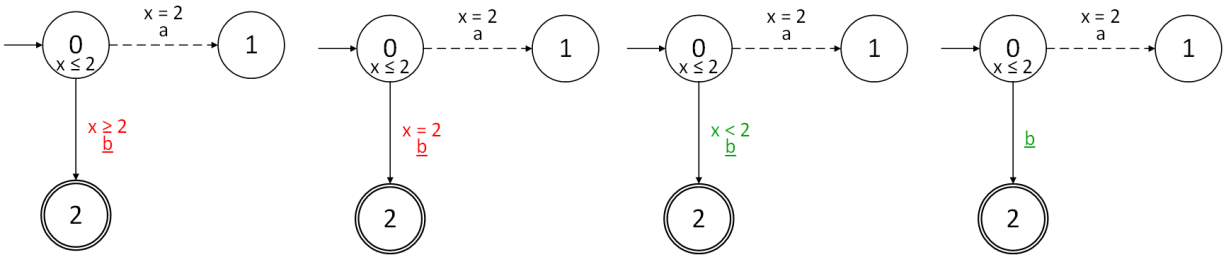


Figure 3.8: *Example TA with forcible event b with guards that allow invariant adaptation (green) and guards that do not (red)*

Figure 3.8 shows a TA with forcible event b. The goal here would be to preempt the smallest possible time delay that leads to state $(0, x = 2)$, by adapting the invariant $I_0 = x \leq 2$ to $x < 2$. The red guards represent two situations for which it is not possible to adapt the invariant, as in the underlying transitions system there is no outgoing event from any state preceding $(0, x = 2)$. In the two green cases on the right, it is possible. Note that the guard $x < 1$ also allows preemption, but only to $I_0 = x < 1$ at the most. It is also assumed that forcible events appear naturally in a system, such that the supervisor cannot introduce them to the plant or determine if an event in the plant is forcible or not [37]. Note that there is no direct relation between forcibility and controllability, such that forcible events can either be controllable or uncontrollable.

Another goal that could be set in TA synthesis, is to find a timelock-free supervisor. According to [38] "both absence of Zeno runs and deadlocks suffice to guarantee timelock-freedom". Whilst verification of timelock-freedom has been studied, synthesis of timelock-free supervisors does not seem to have been studied. Assuring timelock freedom through synthesis is not a trivial task. Therefore we choose not to take it into account in synthesis, as the problem of TA supervision and state space explosions should be tackled for well-known synthesis goals first. It is left up to the modeller to design a system without timelock. How to this, is explained in [28]. Therefore the TA supervisor synthesis problem is the same as the FA problem, but now with TAs that may include forcible events.

Problem 3.2 – TA Supervisor Synthesis Problem: the main synthesis problem for TA is defined as follows:

For a given plant P^t and plantified requirements $R^{p,t} = R_1^{p,t}||R_2^{p,t}||...R_n^{p,t}$ modelled as TAs, find a maximally permissive supervisor S^t that is controllable with respect to $P^t||R^{p,t}$ and for which $S^t||P^t||R^{p,t}$ is non-blocking.

To research the applicability, improvements and problems considering the introduction of invariant adaptation, Problem 3.2 is considered for an indirect and a direct synthesis approach in the following sections.

4 Indirect Synthesis

There are two main reasons why standard FA synthesis algorithms cannot be applied directly to TA. The first reason is that TA include variables, the clocks, and the second reason is that the dense nature of these clocks results in an infinite state space. For FA with discrete variables (EFA), algorithms such as SSEFA [3] exist. However, discrete variables in EFA can only increment or decrement on the update of an edge, whilst clocks in TA evolve continuously in each location. Therefore, SSEFA is not suited for TA. Instead, the infinite state TA should be transformed into an FA before applying FA synthesis. Here, this will be referred to as indirect synthesis as the algorithm will not use the original TA model directly as an input, but requires a conversion to a finite state representation of the TA first.

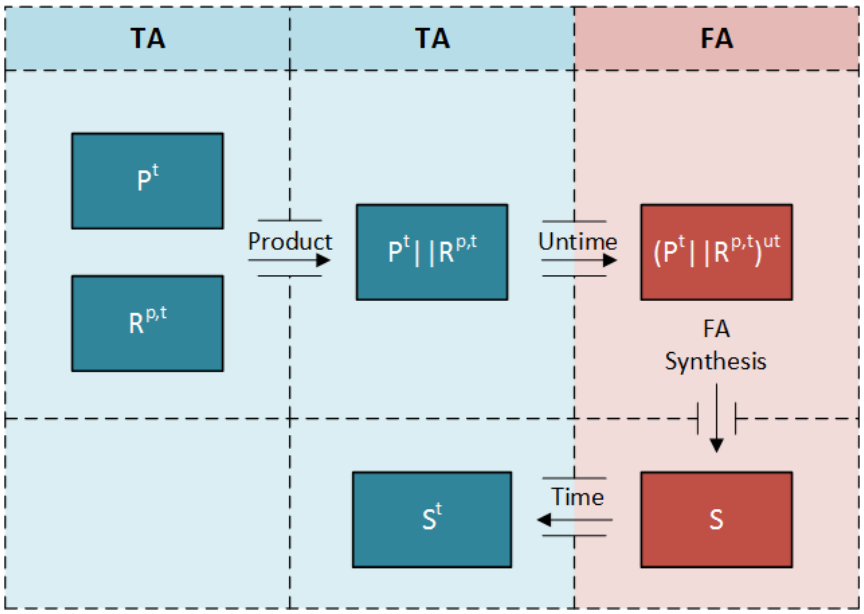


Figure 4.1: *Overview of indirect synthesis steps, where ^{ut} represents an untimed timed automaton.*

The goal is to compute a maximally permissive timed supervisor S^t which is controllable with respect to $P^t||R^{p,t}$ and for which $S^t||P^t||R^{p,t}$ is non-blocking (Definition 3.2). This was achieved in [17] via a language based approach that performs DES synthesis on the possible traces of the TA using a region-graph time abstraction[6]. Based on the steps as introduced by [17], we propose an approach that is more suited for implementation in CIF3 [27]. We consider two event-based FA synthesis approaches that are compatible with the existing synthesis algorithms in CIF3. This can prove to be useful for future research on the topic. Figure 4.1 gives an overview of the steps that are taken to find S^t:

1. compute the synchronous product of the timed plant P^t and plantified requirement $R^{p,t}$: $P^t||R^{p,t}$,
2. abstract explicit time, i.e. “Untime” the product TA such that it becomes an FA,
3. use the resulting FA: $(P^t||R^{p,t})^{ut}$ as an input for the FA synthesis algorithm to produce a finite state supervisor S and
4. extract the timed supervisor S^t, i.e. ”Time” the FA supervisor S such that it becomes a TA.

Step 1, the TA synchronous product for the plant and the plantified requirements remains unchanged with respect to Definition 2.11. Therefore, we shall not discuss this again in this section. In Section 4.1, the region graph abstraction is briefly explained and it is shown how this can be used to construct a Region Equivalent Quotient Graph (REQG) for the purpose of event-based synthesis (step 2). In Section 4.2, we distinguish between two types of FA synthesis. The first is based on standard DES SCT and the second is based on TDES SCT, which includes forcible events (step 3). We show that TDES synthesis can be applied to solve the untimed bus pedestrian example and that it is possible to solve an even larger set of problems by assuming that the grammar of the invariants and guards in the TA

[6]Note that in [17], they refer to maximally permissive as least restrictive.

supervisor includes the expressions that represent the regions in the states of the REQG. The "Timing" procedure is discussed in Section 4.3, where it is shown how the TA supervisor S^t can be constructed based on the region information of the FA supervisor S (step 4). Finally, a discussion on how these steps could be implemented in CIF is provided in Section 4.4.

4.1 Untime: Abstracting Explicit Time

There are a number of methods to abstract explicit time from the TA. For the purpose of SCT for systems modelled as TA, one of the methods is to construct a region graph [17] and the other is to transform the TA to an se-FSA [18]. Due to the introduction of a different type of events than those that are conventionally used in SCT, using the latter approach requires an adjustment of the definition of controllability. This makes it more difficult to compare results to the direct synthesis results in Section 5 and to make it suitable for implementation in CIF3. Additionally, the state space blow up of the region graph based abstraction will not be a problem for the compact theoretical examples that are used in this study. Therefore, a region graph based approach is used here. The construction of a region graph is based on the definition of region equivalence [29], [31].

Definition 4.1 – Region equivalence: For a real number $r \in \mathbb{R}$, the fractional part is given by $\{r\}$ and the integral part by $\lfloor r \rfloor$, such that $r = \lfloor r \rfloor + \{r\}$. Let there be a TA with clock set C. Each clock $x \in C$ is mapped to the largest integer n in the set of clock constraints of form $x \sim n$, by the ceiling function $k(x)$. Two clock assignments u and v are region equivalent, indicated by $u \dot{\sim}_k v$, iff:

1. for all clocks $x \in C$, either $\lfloor u(x) \rfloor = \lfloor v(x) \rfloor$ or both $u(x) > k(x)$ and $v(x) > k(x)$,
2. for all clocks $x \in C$, if $u(x) \leq k(x)$ then $\{u(x)\} = 0$ iff $\{v(x)\} = 0$ (i.e. the corner points, horizontal and vertical open line segments),
3. for all clocks $x, y \in C$, if $u(x) \leq k(x)$ and $u(y) \leq k(y)$ then $\{u(x)\} \leq \{u(y)\}$ iff $\{v(x)\} \leq \{v(y)\}$ (i.e. distinguish between $x > y$ and $x < y$ with diagonal open line segments).

Here the index k in $\dot{\sim}_k$ refers to the fact that a clock specific ceiling is used instead of using the maximum value of all clocks as in [29]. Not doing so can unnecessarily increase the state space of the untimed TA even further.

A region graph (RG) can be used to visualize these regions.

Example 4.1 – Clock Regions in Region Graph: Consider a TA with maximum clock constraint values $k(x) = 1$ and $k(y) = 1$. Based on these values, a region graph containing 18 regions can be constructed using Definition 4.1 and visualized by plotting clock y vs x.

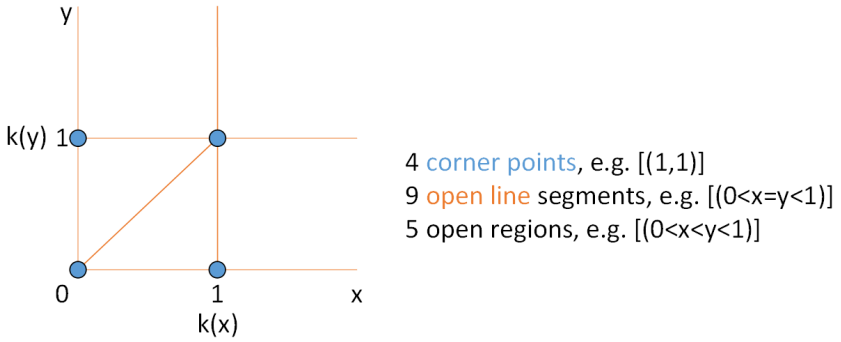


Figure 4.2: *Region graph example*

Figure 4.2 shows this region graph, where a distinction is made between three types of clock regions: corner points, open line segments and the open regions between them. Note that in accordance with Definition 4.1 there is no open line segment $y = x$, for $y > k(y)$ and $x > k(x)$. There are also no open line segments $y = x - 1$ for $x > k(x)$ and $y = x + 1$ for $y > k(y)$. Such segments would exist if diagonal constraints of form $x - y \leq n$ were allowed in the grammar of the input TA. The region graph construction can be extended to include diagonal constraints of this form by introducing the new open line segments. However, we choose not to include them, as according to [39] they

do not add to the expressiveness of TA and they can be removed systematically. Additionally, any other form such as $2x - y \leq n$, would contradict time abstracting bisimulation (TAB) as time successors from equivalent regions would lead to different regions [28]. This TAB is required for the correct representation of the TA as an FA for synthesis.

The resulting region graph can then be used to derive what we will refer to as a region equivalent quotient graph (REQG). The latter is necessary to include location information in the graph and to model it as an FA. This allows to extract the guards and invariants from the region expressions in the REQG states and links them to the input TA by means of the location information instead of using a language based timing approach as in [17]. Additionally, the REQG differs from the region automaton (RA) in [29] in that time jumps τ between the regions are left in the graph. Therefore, it becomes a strong time-abstracting bisimulation of the TA, which is necessary for applying both DES and TDES synthesis [28]. Each clock-equivalence region in the region graph is considered to be a symbolic state. Example 4.2 shows what this would look like for a simple two location TA.

Example 4.2 – TA to REQG: Given a simple TA, with two clocks x and y with initial assignments $u(x) = u(y) = 0$ and maximum value $k(x) = k(y) = 1$, the following RG and REQG result.[7]

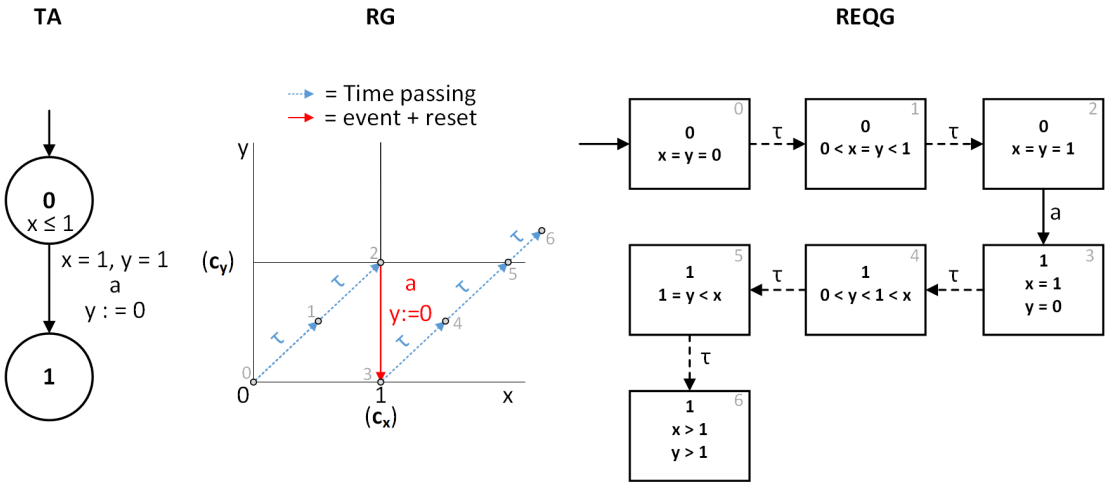


Figure 4.3: *RG to REQG example*

The RG in Figure 4.3 shows a nice property of TA, namely that clocks evolve at the same rate. Therefore, a time delay will always follow a line with slope 1 in the RG of a TA with two clocks. Time jumps between regions are indicated by event τ shown as a blue dotted line in the RG. Events without resets do not influence the region graph, because it does not take into account the locations of the states. Only if a reset of one of the clocks occurs after an event transition, it will be visible in the RG as shown by the red arrow. Each state in the RG is numbered in grey, to show its relation to the states of the REQG. There are two types of transitions in the REQG:

1. an event transition $S_1 \xrightarrow{e} S_2$ such that S_2 is an event successor of S_1,
2. a time jump transition $S_1 \xrightarrow{\tau} S_2$ such that S_2 is an immediate time successor of S_1.

Here, an event transition only influences the region if a clock reset occurs on the edge to which that event belongs. Therefore, each symbolic state of the REQG is a combination of the location of the TA due to event transitions and the region of the RG based on time delays and resets. As the REQG has a finite number of clocks, for which the values are partitioned into a finite number of clock equivalence regions with a finite number of time jumps τ between them, it can be concluded that the resulting REQG is an FA. To preserve the finiteness of the REQG and the compatibility with the clock evolution rate, it is required to use integer constants in the constraints of the TA. Alternatively, rational constants can be used, but in this case the least common multiple of denominators of all TA constraints should be multiplied with all constants [29]. Note that the time jump transitions τ are considered as uncontrollable events as in [17]. This is exactly what is needed to apply FA synthesis. Note that the REQG does not have self-looping τ events, as this event indicates a time jump between regions and not a tick of the global clock

[7] Note that to keep the example simple, the open regions are not used here. However, if guards of form $x \leq n$ are used instead of $x = n$, these open regions definitely play a role. Certainly, if self-loops with resets are introduced.

as in [5], [37]. In line with what is proposed for tick events in [5], the REQG could be minimized before applying synthesis. The reason is that subsequent time jumps between states without the possibility to take events $e \in E$ are not interesting for the purpose of synthesis as they represent delays of time that do not influence the possibility of taking discrete transitions between locations. For example, the REQG in Figure 4.3 could be minimized to a graph where grey states $0-2$ and $3-6$ can be seen as one state. The result is a graph with two states. We choose not to minimize the REQG in our examples, as they are compact enough to explain the synthesis approach.

Finally, it should be noted that the choice to apply the synchronous product of the plant and requirement TA before abstracting time is not arbitrary. The reason for this is that due to abstraction, the relative value of the distinct clocks in one TA is lost with respect to those of the other TA. If the synchronous product is applied before untiming, the third rule of the region equivalence relation makes sure that a distinction is made between relative values of clocks in the regions. This is explained in more detail in Appendix B by means of Example B.1.

4.2 Supervisor Synthesis

In [17], DES synthesis was applied to the untimed plant model to find a maximally permissive, non-blocking and controllable supervisor. However, the results are only maximally permissive with respect to DES SCT. The DES synthesis concepts as introduced in Section 3.1 limit the capabilities of finding a supervisor that is suited for new situations that arise in TA which require forcing. As the REQG structure is quite similar to that of the tick-models in [37], the DES synthesis approach can be extended to take into account the notion of forcibility of TDES synthesis. Both the results of DES and TDES synthesis will compared to show which approach suits which problems.

4.2.1 DES Synthesis

For ease of explanation the synthesis algorithm that will be used here, is a simplified version of the algorithm in CIF3. It differs in the fact that it is not optimized for efficiency and networks of automata, but the basic principles are the same. It is referred to as event-based synthesis in [4] and it is built up out of four main steps starting from the uncontrolled system $(P^t||R^{p,t})^{ut}$:

1. compute the set of blocking states. If there are none, the resulting FA is the supervisor, otherwise continue,
2. compute the set of bad states,
3. remove all controllable events that point towards a bad state,
4. remove all unreachable states and the edges between them. Return to step 1.

Here, bad states are introduced to take into account the fact that uncontrollable events may not be removed. A small example will help to clarify what bad states are.

Example 4.3 – Bad states: Consider an REQG, with blocking state 3 and marked state 4 as in Figure 4.4.

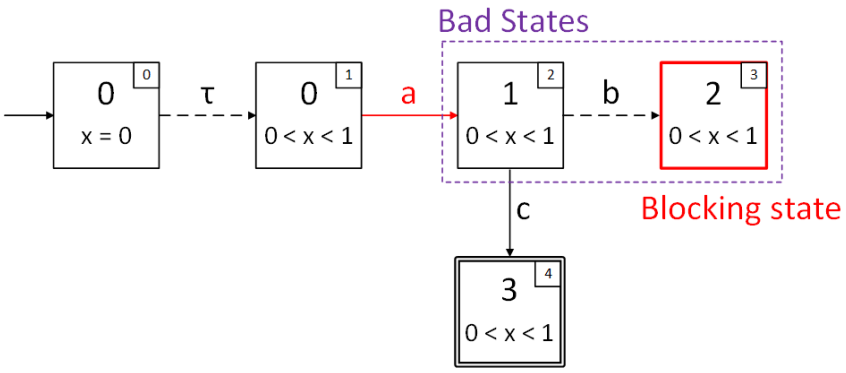


Figure 4.4: *Part of a possible REQG with blocking state* 3 *and bad states* 2 *and* 3

From non-blocking state 2, it is possible to reach blocking state 3 via an uncontrollable event b. This means that even though state 2 is non-blocking, there is no means to prevent state 3 from being reached as event b cannot be

removed. Therefore, the only solution is to prevent the REQG from ending up in state 2, by removing controllable event a. To make sure this happens during synthesis, another type of state is considered, which is the bad state. Bad states are used to indicate which controllable events must be removed to make sure that no blocking state can be reached. This means that any state from which a bad state can be reached, is also a bad state. Therefore, after removing event a, state 1 becomes blocking and state 0 also identifies as a bad state. An empty supervisor results.

The goal of the four step algorithm is to remove blocking, without introducing uncontrollability of S with respect to $(P^t||R^{p,t})^{ut}$. By taking the uncontrolled system as a starting point and only removing the necessary controllable edges, maximal permissiveness is preserved. Controllability of the product $P^t||R^{p,t}$ with respect to the plant is assured through the plantification of the requirements.

In DES synthesis, forcible events are not considered. As a result, the time jump event τ is always modelled as an uncontrollable event. Therefore, τ events may not be removed and all states that are time predecessors of a bad state, are also considered bad states. An example of such a problem is given below.

Example 4.4 – DES synthesis example: Consider a TA A^t with deadlock location 2 as shown in Figure 4.5.

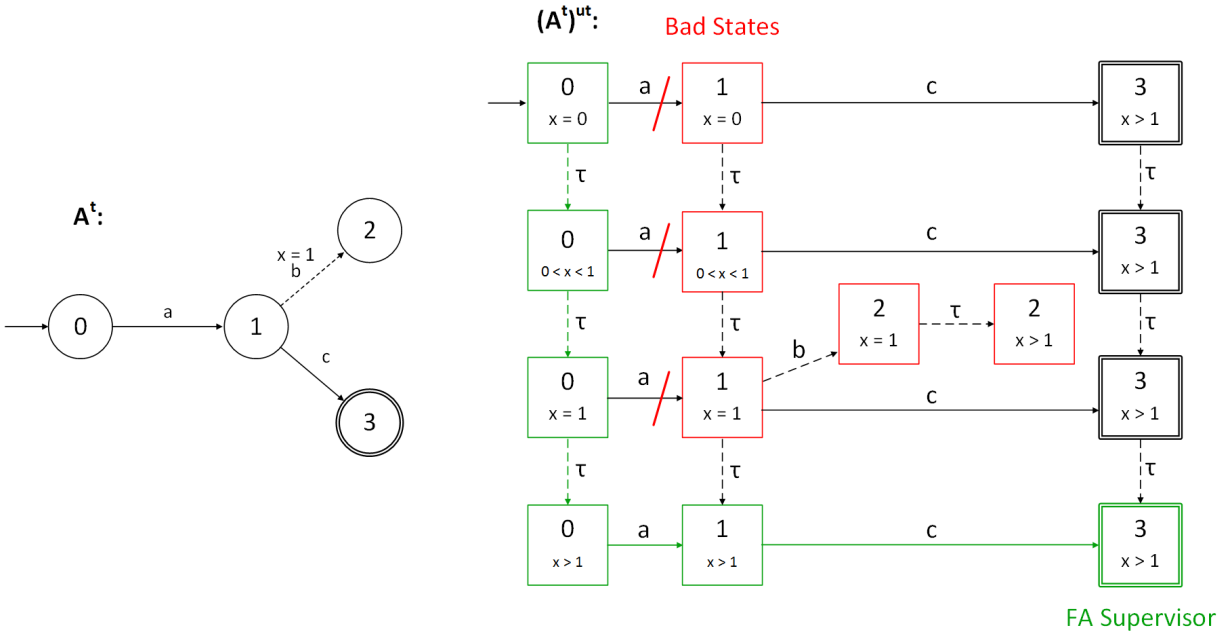


Figure 4.5: *Example TA, deadlock problem*

Clearly, state $(2, x = 1)$ and $(2, x > 1)$ are blocking and therefore bad states. As uncontrollable event b points to state $(2, x = 1)$, state $(1, x = 1)$ is also a bad state. Finally, states $(1, x = 0)$ and $(1, 0 < x < 1)$ are also bad states as they are time predecessors of bad state $(1, x = 1)$. Using the event-based DES algorithm, it is possible to prevent these states from being reached by disabling event a from states $(0, 0 \leq x \leq 1)$. Step 4 of the algorithm is to remove the unreachable states that result from removing the controllable events. As the green-highlighted states are all non-blocking, these represent the FA supervisor that results from synthesis. Note that even without knowing the timing procedure, it can be seen that the TA equivalent of this supervisor would be A^t with a guard adaptation such that $0 \xrightarrow{x>1} 1$. Therefore, the TA would delay past the bad state.

However, it is known from Section 3.2 that for TA, problems exist that can only be solved by introducing forcible events. In Example 4.5, we show what happens when applying DES synthesis to the REQG of the bus pedestrian example (Example 3.2).

Example 4.5 – Bus Pedestrian example, DES synthesis: Untiming the product TA in Figure 3.5, results in the REQG in Figure 4.6.

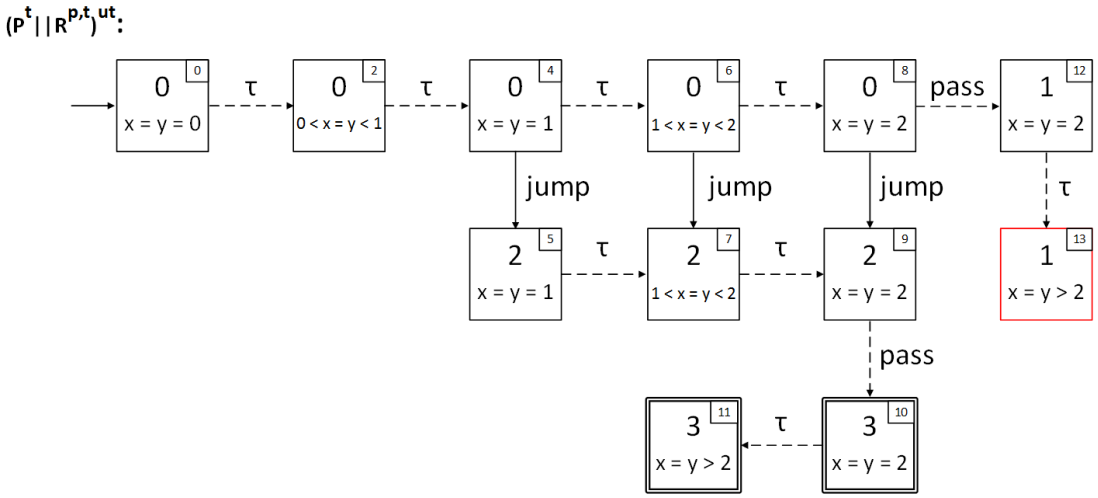


Figure 4.6: *REQG of Bus Pedestrian example plant*

It can be seen that symbolic state 13 is identified as blocking in the first step of the event-based algorithm, as there are no outgoing transitions.

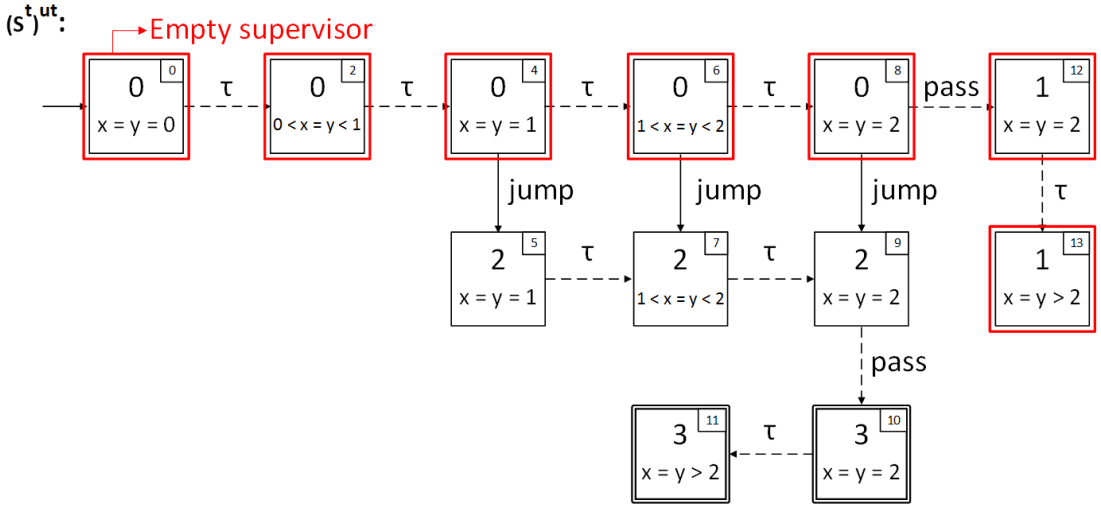


Figure 4.7: *REQG of Bus Pedestrian example, untimed supervisor after TDES synthesis*

Because events $pass$ and τ are uncontrollable, all states preceding state 13 are identified as bad states. This means that the initial transition must be disabled, resulting in an empty supervisor for this problem. The empty supervisor is the correct maximally permissive result for this problem in DES synthesis, but it is preferred to compute a supervisor that can save the pedestrian.

The tick-model in [37] is coarser but very similar to the REQG above. There, a supervisor that saves the pedestrian is found. To understand how, TDES synthesis must be considered.

4.2.2 TDES synthesis

Including the notion of forcibility allows for a much broader range of solutions to TA synthesis problems. We propose an algorithm that consists of similar steps as in the DES synthesis approach, but separately takes into account the τ events that become controllable due to the presence of a forcible event. The new steps are as follows:

1. compute the set of reachable blocking states, if there are none, remove all unreachable states and the edges between them, the resulting FA is the supervisor, otherwise continue,

2. compute the set of bad states,
3. remove all controllable events that point towards a bad state and that are not τ jump events. If no events can be removed, continue, otherwise return to step 1,
4. remove all controllable τ events that point to a bad state,
5. for all states that do not have an outgoing forcible event and also no τ event, excluding the states that did not have τ events in the input REQG due to invariants or the maximum clock region, add the τ event again, but now as an uncontrollable event and return to step 1.

There are two key differences with the DES algorithm. The first one is that the controllability of time jump events τ depends upon the availability of a forcible event in a state. This means that time predecessors of a bad state are not necessarily bad states as a τ event pointing to a bad state may be controllable. The second key difference is that an extra iteration loop is added, in which only controllable events that are not τ events are removed. This is repeated until no controllable events pointing at bad states remain. This is done to make sure that τ events are not removed prematurely, as it could be the case that a controllable forcible event is removed at some point during synthesis. Due to new bad states that appear from the iterative removal of controllable events in step 3, the latter definitely results. However, it could also possibly occur due to the removal of τ events in step 4. As we have no proof that the problem also exists for the removal of τ events or that it does not exist, we include step 5 to return τ events that have been removed prematurely as uncontrollable events. It is expected that the algorithm will terminate, as the return of the τ event as an uncontrollable event, makes sure that the source state of that τ event will become a bad state. The reason for this is that the controllable τ event can only have been removed initially if the target state was a bad state.

In the case that no τ events are ever adapted prematurely, the algorithm effectively reduces to the first four steps. However, it is inefficient to hold onto the reachable states if this is the case. Therefore, if a proof is found that step 5 is not necessary, an alternative five step algorithm is provided in Appendix D.1. This allows the removal of unreachable states in step 3 and as step 5, such that it is no longer necessary to determine the reachable blocking states in step 1.

To show how step 3 prevents premature removal of τ events, Example 4.6 is considered. Note that in the other examples that are provided in this work, the problem of premature τ removal does not occur. Therefore, we remove the unreachable states intermediately for clarity.

Example 4.6 – Premature τ removal: Consider a possible REQG of a TA, shown in Figure 4.8. Here, event b is considered forcible such that the τ event from state 2 is initially controllable.

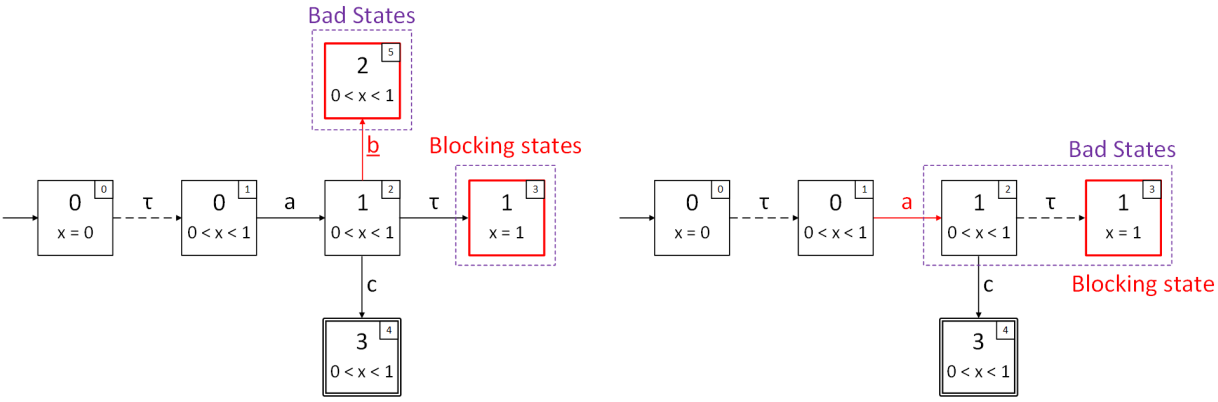


Figure 4.8: *Example showing a forcible event b, which points to a blocking state.*

The left figure shows the first three steps of the TDES indirect synthesis algorithm. The blocking states and the bad states are both the same, states 3 and 5. However in step 3, controllable forcible event $\underline{b}$ is removed, as it points to blocking state 5. Because of this, the algorithm returns to step 1, shown in the right figure. As state 5 has been removed, only state 3 is blocking. However, as the only forcible event from state 2 has been removed, the τ event between state 2 and blocking state 3 is now uncontrollable. Therefore, state 2 also becomes a bad state. The only

option that remains is to remove event a and return to step 1 again. The final result is an empty supervisor, as state 1 becomes blocking and state 0 is identified as a bad state.

To show the advantage that TDES synthesis has over DES synthesis, the bus pedestrian example is used as a comparison.

Example 4.7 – Bus Pedestrian example, TDES synthesis: As for the DES synthesis example, the REQG of the bus pedestrian example is computed first. However, in this case the event $jump$ is considered forcible as in [5], [37]. The result is that the controllability of event τ now depends on the existence of a forcible edge in a state.

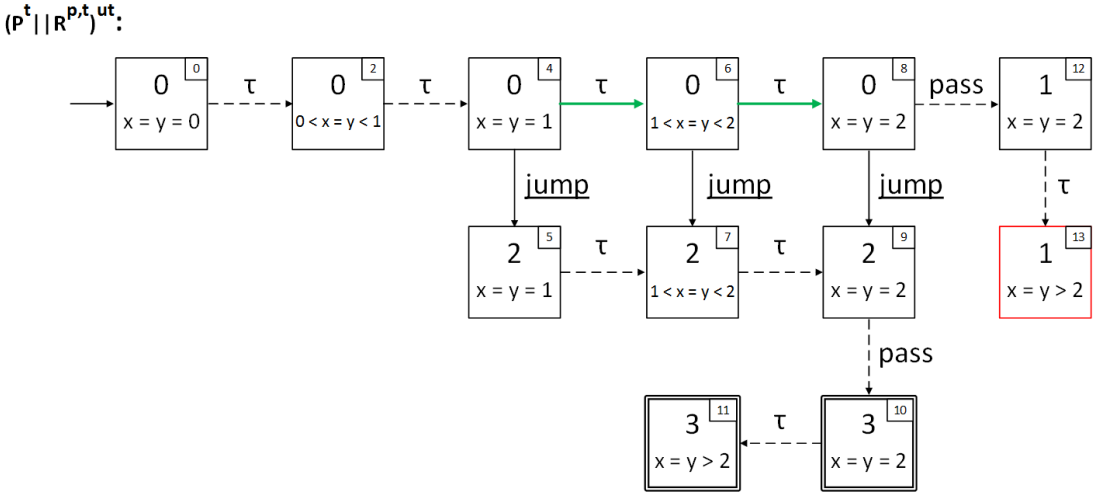


Figure 4.9: *REQG of unconrolled system, with forcible event jump*

As shown in Figure 4.9, the states of the REQG remain the same. However, the green arrows now indicate the edges with τ events that have changed from uncontrollable to controllable. It can be seen that state 13 is found to be a blocking state and that all states where the event jump is possible now have controllable edges τ.

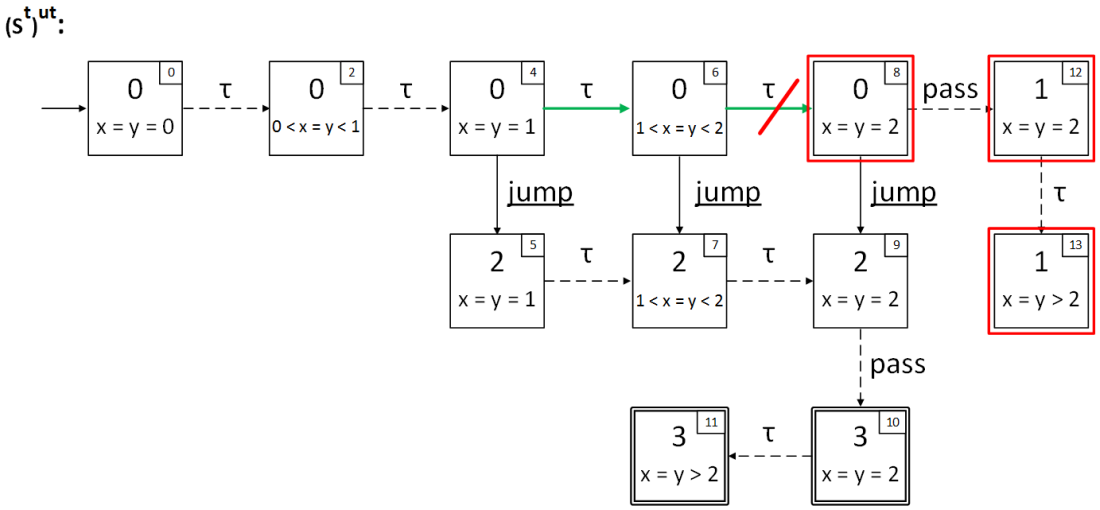


Figure 4.10: *REQG of unconrolled system, with forcible event jump*

In Figure 4.10, it can be seen that the now controllable event τ can be eliminated from the REQG. This allows to solve the blocking problem whilst satisfying the requirement that the pedestrian should jump before the bus passes at $x = 2$. The result is an REQG that is very similar to the BPSave supervisor in [37], which only considers the integer values of x and y.

However, there is one potential problem that turns up when using guards that represent intervals, e.g. $n_1 < x < n_2$ or $x = n$. Namely, that allowing the removal of τ events, may result in behaviour that would not be permitted by

the grammar of the input TA invariants and guards. We show this by applying TDES indirect synthesis to Example 4.4 and comparing the resulting FA supervisor with the one that results from DES synthesis.

Example 4.8 – TDES interval problem: Consider the same TA as in Example 4.4, but now we assume that event c is forcible as shown in Figure 4.11.

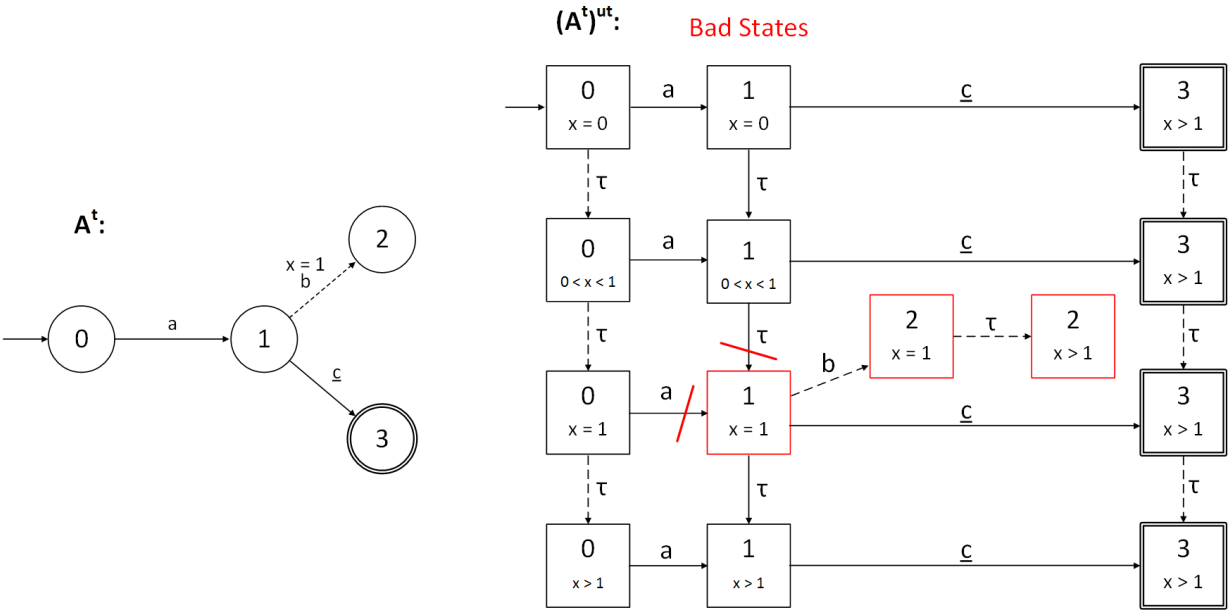


Figure 4.11: *TA and REQG showing identification of bad states when event c is forcible.*

As event c is forcible and available in all states with location 1, all τ events leaving these states are controllable. Therefore, only the two blocking states with location 2 and state $(1, x = 1)$ are considered as bad states. We show both the removal of controllable event a and event τ pointing to state $(1, x = 1)$ in one figure for compactness. The result of removing these events is shown in Figure 4.12.

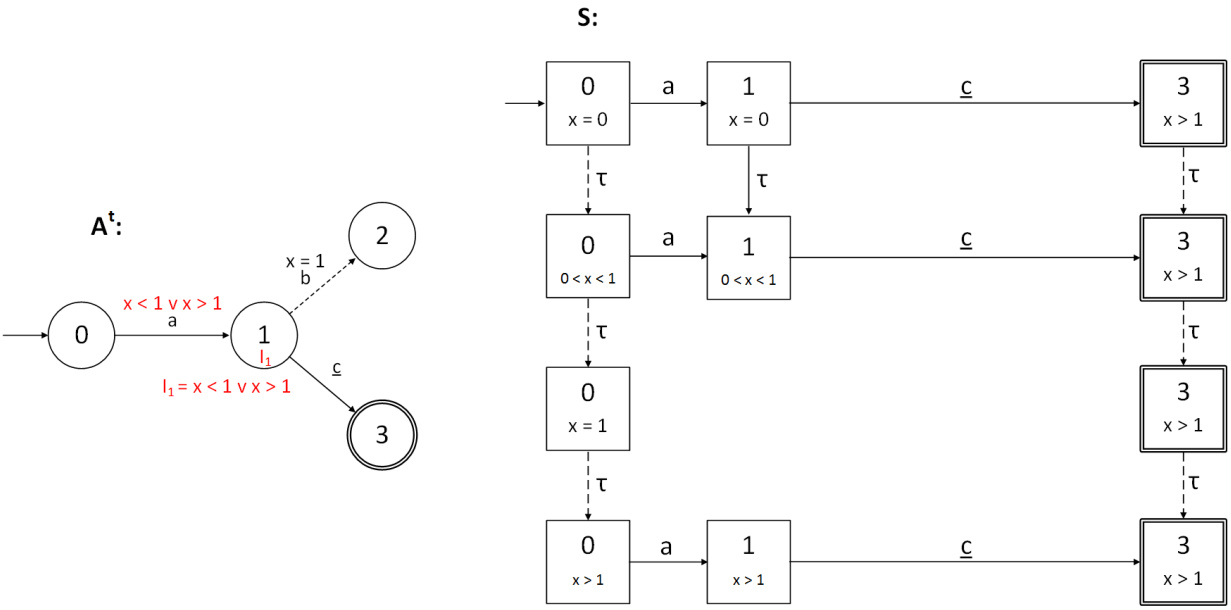


Figure 4.12: *Resulting REQG and equivalent TA after TDES synthesis*

The problem may not directly be evident from only viewing the REQG. Therefore, the equivalent TA is shown in

the left of Figure 4.12. Evidently, the supervisor can either force a transition out of location 1 before $x = 1$, or delay in location 0 until $x > 1$. However, for this to be possible, the disjunction in $I_1 = x < 1 \vee x > 1$ is required. This type of expression is not considered in the grammar of the clocks constraints of the input TA. Therefore, if we choose to follow these restrictions when reconstructing the TA, we are left with choosing either to delay past $x = 1$ with a guard $x > 1$ or use invariant $x < 1$ to leave location 1 before $x = 1$. Therefore, the resulting supervisor would not be unique in this case.

Instead of viewing this as a problem, we see it as a solution to find the most permissive FA supervisor, independent of restrictions on guards and invariants. This is the supervisor given in Figure 4.12. More specifically, it can be seen that it is no longer a problem if a state is reached, that precedes a bad state, as long as there is a forcible event to preempt a τ jump to that bad state. Next to that, by allowing the TA in location 0 after $x = 1$, it is also possible to delay past the bad state. This leaves the choice open, what to do with that information in the FA supervisor. The most intuitive choice is to allow the most permissive behaviour, by omitting any restriction on the grammar of the guards and invariants in the supervisor. This greatly simplifies the timing construction in Section 4.3.

4.3 Timing

Given the resulting supervisor S from synthesis, there is sufficient information to supervise the uncontrolled plant. However, this would require a function that continuously maps each clock assignment u together with the current location l to the current state of the REQG of the supervisor. The supervisor can then decide and communicate which events are disabled in that state. This is similar to coming up with a new control structure as in [16].

However, it is much easier to analyse and more compact to have the global controlled behaviour, the supervised plant, as a TA. Therefore, a timing procedure is proposed, which uses the region information in the states and the original uncontrolled plant $P^t||R^{p,t}$ to construct a timed supervisor S^t. To solve the problem that the REQG allows more behaviour, the alternative to creating a new control structure is simply to allow more types of guards and invariants in the TA that represents the supervisor. A unique, maximally permissive result follows.

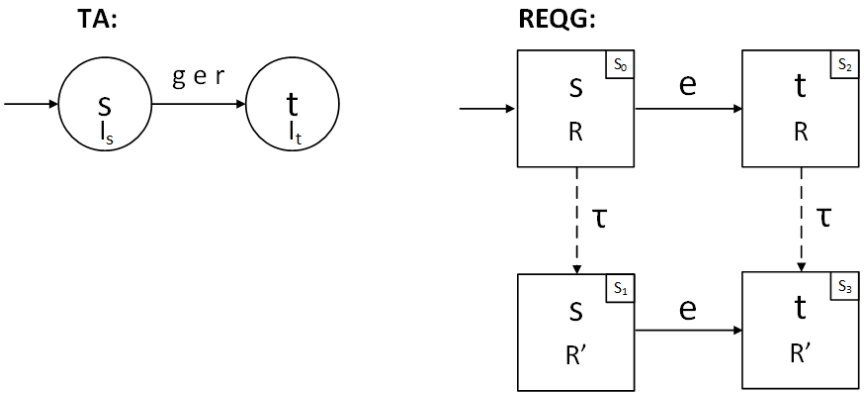


Figure 4.13: *Generalized example of TA and a possible REQG*

Figure 4.13, gives a generalized example of a TA and a possible REQG as a reference to explain the timing procedure, which works as follows.

Starting with the initial location, for each edge $s \xrightarrow{g\ e\ r} t$ of each location s of $P^t||R^{p,t}$ determine which symbolic states of FA supervisor S have an edge $(s, R) \xrightarrow{e} (t, R)$ or $(s, R') \xrightarrow{e} (t, R')$, etcetera. List $J_{s,i}$ stores the indices of these states S_j for each edge $\rightarrow_i$. List K_s stores the indices of all states S_k with the same location s. Now perform the following two steps:

1. Determine the new guard of each controllable edge $\rightarrow_i$ under supervision via:

$$g := g \wedge (\bigvee_{j \in J_{s,i}} S_j.R), \tag{4.1}$$

where g is the guard of the edge in $P^t||R^{p,t}$ and $S_j.R$ refers to the region of state S_j. Using the $REQG$ of Figure 4.13, the equation above would result in: $g := g \wedge (S_0.R \vee S_1.R')$.

2. Determine the new invariant of location s via:

$$I_s := I_s \wedge (\bigvee_{k \in K_s} S_k.R), \tag{4.2}$$

where I_s is the invariant of location s in $P^t||R^{p,t}$ and $S_k.R$ refers to the region R in state S_k. Using the $REQG$ of Figure 4.13, the equation above would result in: $I_s := I_s \wedge (S_0.R \vee S_1.R')$.

No additional restriction to when the invariant can be adapted is needed, as synthesis provides a final result that has already taken into account forcibility. Note that it is independent of the type of synthesis, DES or TDES, if the invariant should be computed. However, in the DES case, the invariant will not have changed as the uncontrollable τ events cannot be removed. The bus pedestrian example is used again, to illustrate the method:

Example 4.9 – Bus Pedestrian example, timing the supervisor: For the resulting supervisor as shown in Example 4.7, the timed supervisor is computed as follows:

First, list $J_{0,0}$ and K_0 are determined for location 0 and controllable edge 0 of that location.

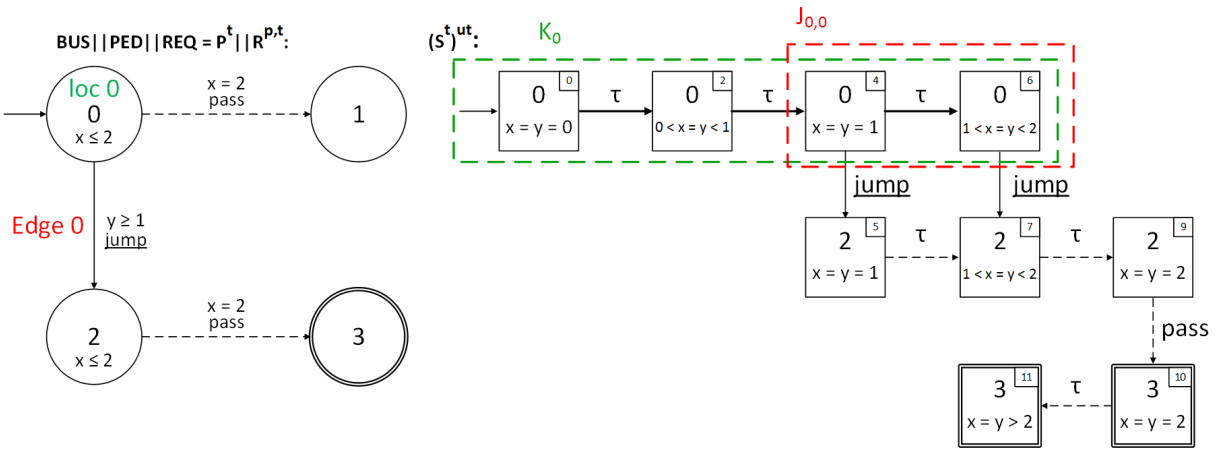


Figure 4.14: *Illustration of first step in timing of Bus Pedestrian example supervisor*

From Figure 4.14, it can be concluded that list $J_{0,0} = [4,\ 6]$ and $K_0 = [0,\ 2,\ 4,\ 6]$. Based on this the guard of edge 0 can be adapted such that:

$$0 \xrightarrow{y \geq 1 \wedge (x = y = 1 \vee 1 < x = y < 2)} 1 = 0 \xrightarrow{1 \leq x = y < 2} 1, \tag{4.3}$$

This type of guard represents a conjunction of multiple guards. In this case this would be $x = y$, $x < 2$, $y < 2$, $x \geq 1$ and $y \geq 1$. However, in line with the assumption of no restrictions on clock constraints, we choose to allow $1 \leq x = y < 2$ as it poses no additional constrictions. The invariant of location 0 can be adjusted using:

$$I_0 := x \leq 2 \wedge (x = y = 0 \vee 0 < x = y < 1 \vee x = y = 1 \vee 1 < x = y < 2) = 0 \geq x = y < 2. \tag{4.4}$$

Which makes sure that the delay to event $pass$ is preempted.

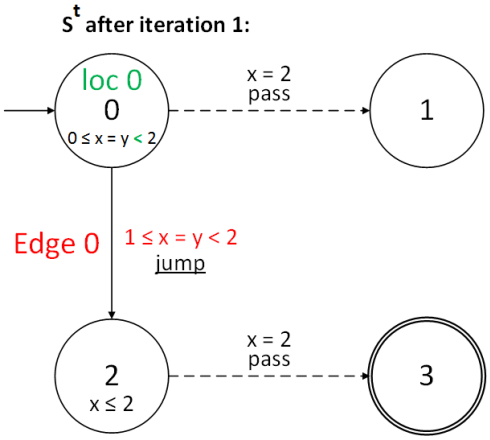


Figure 4.15: *Result after first iteration of timing operation*

The result is shown in Figure 4.15, where the guard and location invariant are adapted. The same steps are taken for the other locations, where nothing changes with respect to Figure 4.15. It may seem as though the supervisor restricts the plant more than necessary after applying the timing procedure. For example, in the plant there was only a guard $0 \xrightarrow{y \geq 1} 2$, and now $0 \xrightarrow{1 \leq x = y < 2} 2$. However, this seemingly more restrictive guard does not limit the behaviour more than the original guard. The reason for this is that the new guard explicitly takes into account the possible behaviour: $x = y$ always holds in this TA and $0 \leq x = y < 2$ is always the case in location 0 due to the invariant. Therefore, the result can still be considered maximally permissive as it does not restrict the behaviour of the uncontrolled plant further than necessary.

For TA with resets, the same timing procedure can be used. As the uncontrolled timed system $P^t||R^{p,t}$ is used as a starting point and the supervisor is not allowed to introduce new resets, all resets of the edges are preserved. Also note that by taking the target location into account in this set of states, a distinction can be made between locations that have multiple edges with the same event, with guards that make sure they remain deterministic.

Example 4.10 – Same outgoing events with different guards: Consider the TA in Figure 4.16 with two edges $\rightarrow_0$ between location 0 and 1 and $\rightarrow_1$ between location 0 and 2:

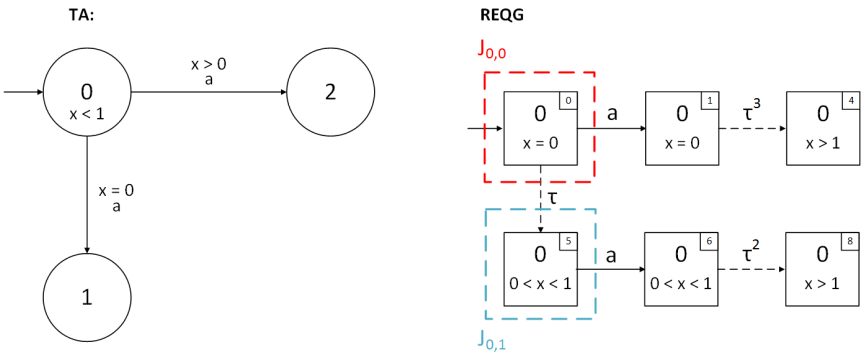


Figure 4.16: *Deterministic TA with two edges that have the same events, but different guards and its REQG*

The REQG of this TA is shown on the right side of Figure 4.16. Determining the lists $J_{0,0}$ for the guard of edge $\rightarrow_0$ from location 0 and $J_{0,1}$ for the guard of edge $\rightarrow_1$ from location 0 shows that a distinction is made between the two edges. Note that synthesis is not considered here as it would result in an empty supervisor.

Even if the target locations were not taken into account, the result would probably be the same. If multiple edges with the same events, but different guards exist, then all states with that event in their outgoing edges will be considered in $J_{s,i}$. As a less strict guard from the disjunction of all of the regions is compared to the original guard via conjunction, the original guard will remain the same or become stricter. However, it is more intuitive to make the distinction between different edges. Therefore, taking into account the target locations when determining $J_{s,i}$ is still

proposed in the timing operation.

All in all, a conceptual method to the problem of indirect synthesis has been found. It is suitable for the purpose of researching application of both DES and TDES SCT and comparing their results for small examples. It can be concluded that TDES SCT allows to compute supervisors for a broader range of problems than DES SCT, by using the concept of forcibility and applying this by means of location invariants in the reconstruction of the TA. To find a unique TA supervisor, we omit the restrictions that are present in the grammar of the invariants and guards of the input TA. This allows to include disjunctions of expressions such that a preemption before the bad state interval is possible or a delay past it.

4.4 Implementation in CIF

Because the size of an REQG is very sensitive to the maximum clock values in the constraints, its practical uses for synthesis are limited [18]. Therefore, only the synchronous product of TA has been implemented in the source code of CIF as this is also useful for direct synthesis. As it is only interesting for small theoretical examples, the possible implementation approaches of the rest of the steps is briefly explained such that the approach can be compared to existing approaches and to the direct synthesis approach in Section 5.

The algorithm that we use is based on what is currently used in CIF for the (E)FA synchronous product of multiple automata simultaneously. This only considers the reachable locations of the product automaton. As our algorithm is not built for efficiency, but for proof of concept, synchronous products of multiple TA are computed by looping over the product of two subsequent TA. The product of two TA A_1 and A_2 is computed in line with Definition 2.11. However, we do not simply consider the cartesian products of all possible locations. Instead, we start with the initial location pair (l_1^0, l_2^0) and work our way through the outgoing edges of both locations to determine the new target location pairs. The new outgoing edge of source location pair $(l_{s,1}, l_{s,2})$ and new location pair is determined by distinguishing three cases:

$$\begin{cases} \text{New edge } \langle (l_{s,1}, l_{s,2}), e, g_1 \wedge g_2, r_1 \cup r_2, (l_{t,1}, l_{t,2}) \rangle, & \text{if } e \in E_1 \cap E_2, \\ \text{New edge } \langle (l_{s,1}, l_{s,2}), e, g_1, r_1, (l_{t,1}, l_{s,2}) \rangle, & \text{if } e_{A1} \in E_1 \setminus E_2, \\ \text{New edge } \langle (l_{s,1}, l_{s,2}), e, g_2, r_2, (l_{s,1}, l_{t,2}) \rangle, & \text{if } e_{A2} \in E_2 \setminus E_1. \end{cases} \tag{4.5}$$

Based on the locations in these new location pairs, the new target location pairs are determined again by walking through their outgoing edges. This continues until no new location pairs are created, after which the product automaton A_p is the result. It consists of locations l_p that represent the resulting location pairs (l_1, l_2) with the combined edges with guards $g_1 \wedge g_2$ and resets $r_1 \cup r_2$. Additionally, marked locations are determined according to the rule that only if both locations in a location pair are marked, the new location l_p is considered marked.

Invariants have not been implemented yet, because at the time of building the tool TA without invariants were still considered for the sake of simplicity. However, these can be included quite easily via $I(l_p) = I(l_1) \wedge I(l_2)$, for each location pair (l_1, l_2), represented by location l_p. Additionally, only guards as defined for TA in Equation 2.1 are supported. Therefore, it is not possible to include location references.

An example of two input TA and a resulting product TA is given in Example 4.11.

Example 4.11 – CIF3 synchronous product example: Consider the two TA A_1 and A_2 in Figure 4.17 and the resulting synchronous product A_p.

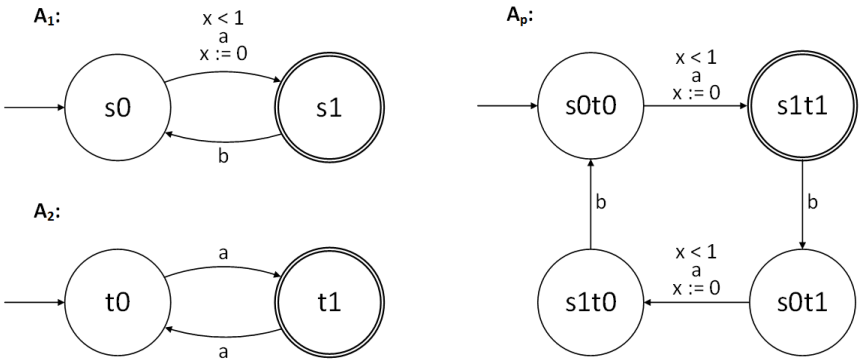


Figure 4.17: *Example TA A_1 and A_2 that are input into the synchronous product for TA tool in CIF3, and should result in A_p shown on the right*

The input TA, A_1 and A_2 are modelled as follows in Listing 1.

```
1  event a,b;
2
3  automaton A1:
4  cont x=0 der 1;
5      location s0:
6          initial;
7          edge a when x<1 do x:=0 goto s1;
8      location s1:
9          marked;
10         edge b goto s0;
11 end
12
13 automaton A2:
14     location t0:
15         initial;
16         edge a goto t1;
17     location t1:
18         marked;
19         edge a goto t0;
20 end
```

Listing 1: *Input TA A_1 and A_2*

It can be seen that clock x is defined as a continuous variable with a derivative of value 1. Note that events a and b are declared outside of the automata, such that they can be used globally. In this case event a is shared by A_1 and A_2. The output of the synchronous product for TA tool is shown in Listing 2.

```
1  event a;
2  event b;
3  automaton product:
4  alphabet a, b;
5  cont x der 1;
6      location A1_s0__A2_t0:
7          initial;
8          edge a when x < 1 do x := 0 goto A1_s1__A2_t1;
9      location A1_s1__A2_t1:
10         marked;
11         edge b goto A1_s0__A2_t1;
12     location A1_s0__A2_t1:
13         edge a when x < 1 do x := 0 goto A1_s1__A2_t0;
14     location A1_s1__A2_t0:
```

```
15          edge b goto A1_s0__A2_t0;
16  end
```

Listing 2: *Product TA* $A_p = A_1||A_2$

The resulting CIF-file shows the automaton "product", which represents the automaton A_p in Figure 4.17. Note that the location names are prefixed with the automaton from which they originated. This is done to give a clear distinction for smaller examples.

The result is a working tool that allows the user to input a file, or multiple files with multiple TA, after which the product automaton of all of those TA is returned. Note that in CIF3 it is possible to distinguish between, supervisor, plant and requirement automata. This distinction is supported in the tool.

How to construct a region graph in the source code, has not been considered in this study. As the partitioning of the clockspace is predefined by the maximum clock constraint values $k(x)$, this function k will have to be constructed first. Next, a map that links clock values to a region is required to construct the graph. Then, a link between the timed automaton and the region graph must be made to create the REQG. Implementation was not discussed in [17], but more information can be found on constructing region graphs for the purpose of model checking in [40].

After having constructed the REQG, the standard DES algorithm that is used in CIF3 cannot directly be applied. This DES algorithm can only apply synthesis to FA without any information other than locations and events, or in the EFA case including discrete variables. Therefore, the algorithm should be adapted such that an REQG can be input and DES synthesis is applied to the symbolic states which can be viewed as locations and the events between those states. To include forcible events, they must be introduced as a separate type of event that can be controllable or uncontrollable. Then either in the REQG construction or in the synthesis algorithm, it should be determined which τ events become controllable. Note that a name other than τ must be used in CIF3, as this is already an existing event. Additionally, a computation of the states that initially have no outgoing τ events in the input REQG should be made, for step 5 in the proposed algorithm. Note that this is only the case if it is proven that τ event removal can result in as situation where a τ event that was removed earlier in synthesis, must be returned as an uncontrollable τ event.

Finally, the new timing procedure could be implemented quite similarly to how it was explained in the previous subsection. First identify the states of interest for each edge, to reconstruct the guards from the disjunction of the regions from which the event is possible. Next, identify the states of interest for the reconstruction of the location invariants. Then use the original TA $P^t||R^{p,t}$ to compare the reconstructed guards and invariants via conjunction with the original ones, and to obtain the reset information of the edges.

We propose that all of the steps, excluding the synchronous product, should be considered as one tool in CIF3. Namely, only the synchronous product can be used for other purposes. The REQG contains symbolic state information that cannot be converted into a CIF automaton without losing that information and the DES synthesis algorithm only considers FA without symbolic state information. It is much easier to adapt the DES synthesis algorithm to work with REQGs and to introduce the REQG as a type of FA with extra information in the source code. Note that the implementation of such a tool should be for research purposes only, as the region graph is too sensitive to state space explosion to use it for practical applications.

5 Direct Synthesis

The indirect synthesis approach may seem like a straightforward solution, as it works with a finite number of discrete states. However, it can quickly become very extensive for practical applications, as the number of states increases rapidly with the number of clocks and their range of values. Therefore, an alternative method is sought, that does not require such a time abstraction of the complete TA behaviour. This type of approach will be referred to as direct synthesis, as it is directly applied to the TA that are used to model the plant and requirements. In this study, we consider monolithic supervisors and aim to adapt the uncontrolled plant $P^t||R^{p,t}$ such that a supervisor $S^t = S^t||P^t||R^{p,t}$ is obtained.

First, we distinguish between two different approaches. As in indirect synthesis, the first approach only considers DES SCT notions in the underlying transition system. This means that forcibility of events is not taken into account and therefore only the guards of the input TA may be adapted to compute the supervisor. This means that all time delays in the underlying transition system remain uncontrollable. We refer to this as Time Uncontrollable Direct Synthesis (TUDS). The second approach includes TDES SCT notions in the underlying transition system, such that it allows the adaptation of invariants if a forcible event is available. This means that some delays in the underlying transition system may be preempted by a forcible event. Therefore, we refer to this as Time Preemptable Direct Synthesis (TPDS). Both direct synthesis approaches are based on the existing supervisory synthesis approach for extended finite automata (SSEFA) [3]. However, it is extended, such that it becomes suitable for TA, by taking into account the possible time delays that exist in the underlying transition system.

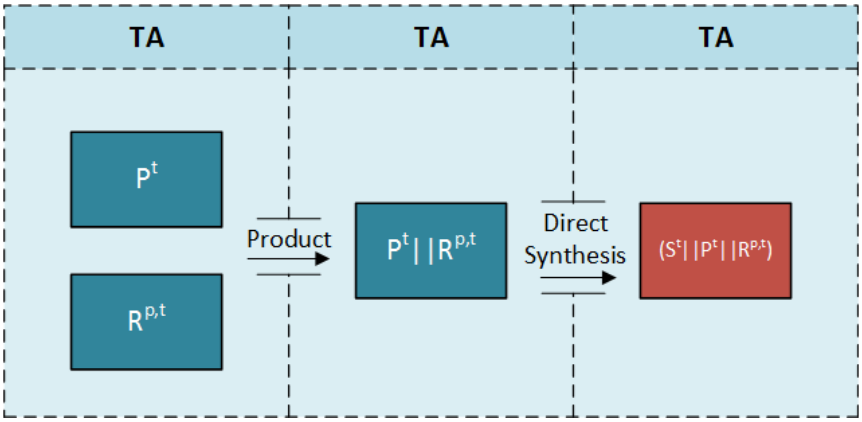


Figure 5.1: *Overview of direct synthesis approach steps*

Figure 5.1 gives an overview of the steps that are made in the direct synthesis approach. It is evident that less steps are required than in indirect synthesis. More importantly, the untiming step is no longer considered, which was the largest problem in indirect synthesis.

As the non-blocking condition computation is the same for both the TUDS and TPDS approach, it is discussed once in Section 5.1. The bad state condition and guard adaptation for TUDS are then discussed in Section 5.2. This is followed by the explanation of the bad state computation, guard adaptation and invariant adaptation for TPDS in Section 5.3. We conclude the direct synthesis section with a comparison with the indirect synthesis approaches and related work and a short discussion on the topic of implementation.

5.1 Compute Non-Blocking Condition

The first step is to compute the non-blocking conditions N, i.e. the clock values for which each location of the plant is non-blocking.

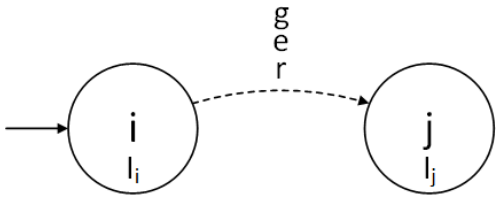


Figure 5.2: *Generic view of TA with source location i and target location j*

Figure 5.2 gives a generic view of a TA, from the point of view of a source location i with an edge $i \xrightarrow{g,\, e,\, r} j$. Computing N_i can be done via the same main steps as in SSEFA, with a few adjustments in accordance with the definition of blocking states for TA (Definition 2.7):

1. Initially, N_i is set to the source locations predicate I_i for all marked locations and to $false$ for all other locations.

$$N_i^0 := \begin{cases} I_i, & \text{if } i \text{ is a marked location} \\ false, & \text{otherwise} \end{cases} \tag{5.1}$$

2. Next, the final non-blocking condition is computed by repeating the following iteration until the predicate for all locations no longer changes:

$$N_i^{m+1} := N_i^m \vee \bigvee_{i \xrightarrow{g\,e\,r} j} (g \wedge N_j^m[r] \wedge I_i) \vee \exists_\Delta (I_i[x+\Delta] \wedge N_i^m[x+\Delta]). \tag{5.2}$$

   Here, N_i^{m+1} is the new non-blocking condition of source location i, N_i^m is the previous non-blocking condition of that location, g is the guard of the edge between i and j, N_j^m is the previous non-blocking condition of target location j with r referring to a possible reset of one or more of the clocks and I_i is the invariant of the source location. Note that $\bigvee_{i \xrightarrow{g\,e\,r} j}$ refers to a disjunction of $(g \wedge N_j^m[r] \wedge I_i)$ for all outgoing edges of location i pointing to target locations j.

   Three adjustments can be seen with respect to SSEFA. First, the initial non-blocking condition for marked locations is the source invariant I_i of location i instead of $true$. This is to take into account marked locations with invariants, which may not always have an outgoing transition. Secondly, a third term is added to take into account that staying in the location can also be a means to reach a non-blocking state. For instance, when a guard bounds a clock from below (i.e. $x \geq n$) or requires a clock to be equal to a value (i.e. $x = n$). Finally, the invariant I_i of the source location is added as a conjunct between the brackets to determine if the edge with guard g actually exists in the underlying transition system. Intuitively one would expect a conjunct with $I_j[r]$ instead of I_i, as we are interesting in finding out if an edge to location j exists from any of the possible states in location i. However, including I_i also includes I_j via the $N_j[r]$ term, such that in this case both invariants are taken into account. This gives more consistent results when comparing edges with guards $x > n$ to edges without guards and makes sure that guards are adapted to explicitly show under which conditions edges exist. The latter is nice for visually working out examples. An example is worked out in Appendix C, to show why results are more consistent for I_i. Note that the resulting behaviour of the supervised system does not change based on this choice.

   Equation 5.2 should be interpreted as follows. The new non-blocking condition is either the existing condition or a condition for which the location can be left via an edge that exists in the underlying transition system, or the condition for which there exists a delay Δ such that the current non-blocking condition is satisfied whilst satisfying the location invariant. Example 5.1 visualizes this by means of the states of the underlying transition system of a TA.

   Example 5.1 – Non-blocking condition algorithm: Consider a timed automaton and its underlying transition system as given in Figure 5.3.

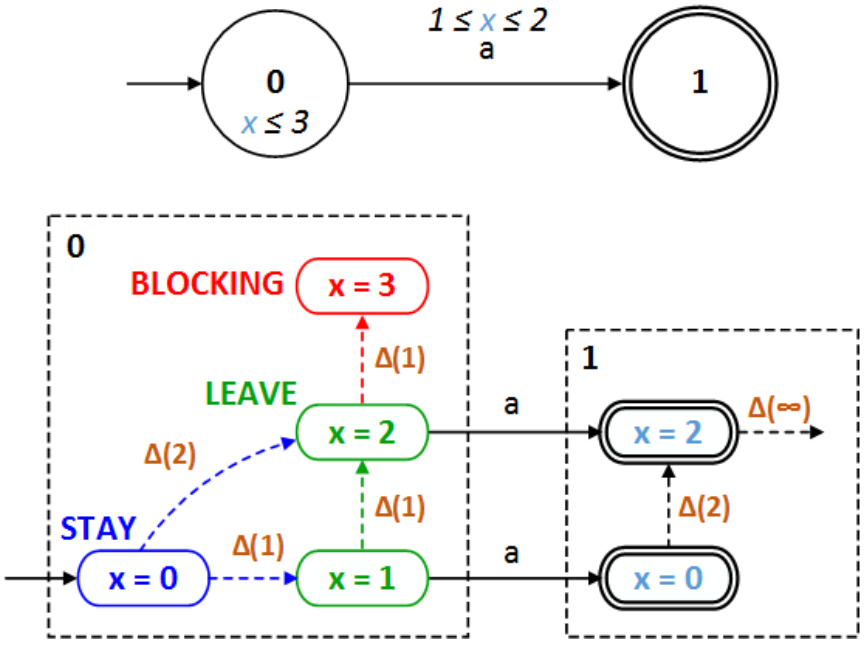


Figure 5.3: *Non-blocking condition example*

The red edge and state in Figure 5.3 indicate all the blocking states $(0, 2 < x \leq 3)$. The green edge and states indicate the states that are non-blocking because they allow a transition to states with marked location l_1 via event a. Finally, the blue state and edges indicate the states that are non-blocking because it is possible to stay in location l_0 (i.e. delay time) until another non-blocking state is reached (the green states). In Equation 5.2, the second term determines the green states and the third term determines the blue ones. To show how the algorithm would work for this example, the intermediate steps are worked out in Table 5.1.

Table 5.1: *Non-blocking algorithm worked out for Example 5.1, note that $t = true$ and $f = false$*

	N	
Iteration	**Location 0**	**Location 1**
0	f	t
1	$f \vee (1 \leq x \leq 2 \wedge t \wedge x \leq 3) \vee \exists_\Delta(x + \Delta \leq 3 \wedge f) =$ $f \vee 1 \leq x \leq 2 \vee f =$ $1 \leq x \leq 2$	t
2	$1 \leq x \leq 2 \vee \exists_\Delta(x + \Delta \leq 3 \wedge 1 \leq x + \Delta \leq 2) =$ $1 \leq x \leq 2 \vee 0 \leq x \leq 2 =$ $0 \leq x \leq 2 = x \leq 2$	t
3	$x \leq 2$	t

In accordance with Equation 5.1, $N_0^0 = false$ and $N_1^0 = true$, here we use f and t respectively to keep the notation more compact. Also note that in iteration 2, for location 0, the term $(g \wedge N_1^2 \wedge I_0) = (1 \leq x \leq 2 \wedge t \wedge x \leq 3)$, is left out as it gives the same result as before: $(1 \leq x \leq 2)$. In accordance with Figure 5.3, the final result is $N_0 = x \leq 2$.

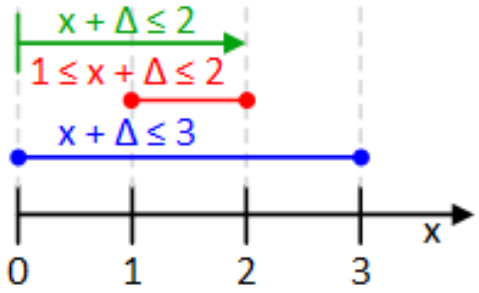


Figure 5.4: *Clock graph of clock x for "there exists" visualization*

The $\exists_\Delta$ term should be read as:

$$\{x \in \mathbb{R}_{\geq 0} | \exists_\Delta (I_i[x + \Delta] \wedge N_i^m[x + \Delta])\} \tag{5.3}$$

The interpretation is as follows; the values of x, such that there exists a delay Δ that satisfies the expressions between the brackets. The term $\exists_\Delta(x + \Delta \leq 3 \wedge 1 \leq x + \Delta \leq 2) = \exists_\Delta(1 \leq x + \Delta \leq 2) = x \leq 2$ from Table 5.1 is worked out graphically in Figure 5.4. It can be seen that as long as $x \leq 2$, there exists a delay $\Delta \in \mathbb{R}_{\geq 0}$ such that the condition $1 \leq x \leq 2$ is satisfied.

To show why it is necessary to include the source (or target) location invariant as a conjunct in the second term of Equation 5.2, another example is used.

Example 5.2 – Why I_i is necessary: Consider the TA and its underlying transition system in Figure 5.5.

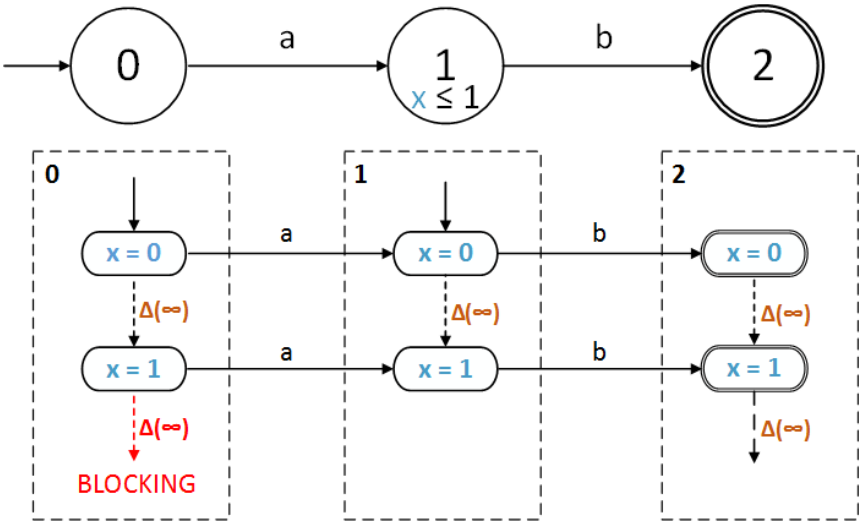


Figure 5.5: *Example showing why the conjunction with I_i is necessary in N*

Table 5.2: *Worked out non-blocking algorithm for Example 5.2*

	N (No I_i)		**N (With I_i)**	
It	**Loc 0**	**Loc 1**	**Loc 0**	**Loc 1**
0	f	f	f	f
1	$f \vee (t \wedge f)$ $\vee \exists_{\Delta}(f \wedge t) =$ $f \vee f \vee f =$ f	$f \vee (t \wedge t)$ $\vee \exists_{\Delta}(f \wedge x + \Delta \leq 1) =$ $f \vee t \vee f =$ $\boldsymbol{t}$	$f \vee (t \wedge f \wedge t)$ $\vee \exists_{\Delta}(f \wedge t) =$ $f \vee f \vee f =$ f	$f \vee (t \wedge t \wedge \boldsymbol{x \leq 1})$ $\vee \exists_{\Delta}(f \wedge x + \Delta \leq 1) =$ $f \vee x \leq 1 \vee f =$ $\boldsymbol{x \leq 1}$
2	$f \vee (t \wedge \boldsymbol{t})$ $\vee \exists_{\Delta}(f \wedge t) =$ $f \vee t \vee f =$ t	t	$f \vee (t \wedge \boldsymbol{x \leq 1} \wedge t)$ $\vee \exists_{\Delta}(f \wedge t) =$ $f \vee x \leq 1 \vee f =$ $x \leq 1$	$x \leq 1$

The problem becomes evident when working out the algorithm without I_i vs the one that includes it. It can be seen in Table 5.2 that the wrong information about location 1 is passed on to N_0. This is shown by the red t that shows up in the $N_j[r]$ term of location 0 in iteration 2. Including the source invariant of location 1, $I_1 = x \leq 1$ solves this. The blue highlighted predicates show how $N_1^1 = x \leq 1$ shows up in the $(g \wedge N_1^2 \wedge I_i)$ term of location N_0^2. The result is a correct conclusion that $x \leq 1$ is the non-blocking condition for location 0.

5.2 Time Uncontrollable Direct Synthesis

The TUDS approach consists of the following steps:

1. compute the non-blocking conditions for each location of the plant,
2. compute the bad state conditions,
3. adapt the guards of the supervised plant based on these conditions,
4. if all guards are the same after a subsequent iteration, this is the resulting supervisor, otherwise return to step 1.

This is similar to the steps in the DES indirect synthesis approach, but in this case non-blocking and bad state conditions are computed instead of the individual states. Based on these conditions, the guards are adapted to provide a non-blocking and controllable supervisor in each iteration with respect to the previous guards. As the non-blocking and bad state conditions can change as a result of these adaptations, all four steps are repeated until the guards no longer change.

The bad state condition computation is discussed first. This uses the non-blocking condition as a starting point and from there it determines for which conditions each location is already bad or there is an uncontrollable event pointing to another bad state or there exists a time delay such that a bad state can be reached. As the bad state conditions follow the same rules as those in DES indirect synthesis and the guard adaptation is equivalent to removing the controllable events that point to bad states in the underlying state transition system, we expect that the supervisor that results from TUDS has the same properties as the supervisor from DES indirect synthesis.

5.2.1 Compute Bad State Condition

With the introduction of uncontrollable events, a new problem arises as guards of edges with uncontrollable events cannot be adjusted to prevent blocking. In EFA, bad state conditions are used to tackle this problem, by propagating the problem backward to an edge with a controllable event. However, in the case of EFA, it is only interesting to

consider uncontrollable edges that are possible from the current state and pointing to a bad state. In TA we are also interested in the evolution of the clocks in each location. More specifically, if there exists a delay such that a state is reached, from which a bad state can be reached by means of an uncontrollable event. Example 5.3, will help to visualize what this means, by viewing the underlying transition system of an example TA and showing why it is necessary to compute the bad state conditions.

Example 5.3 – TUDS Bad state visualization: Consider the TA and its underlying transition system, shown in Figure 5.6. It is assumed that event b is a forcible event.

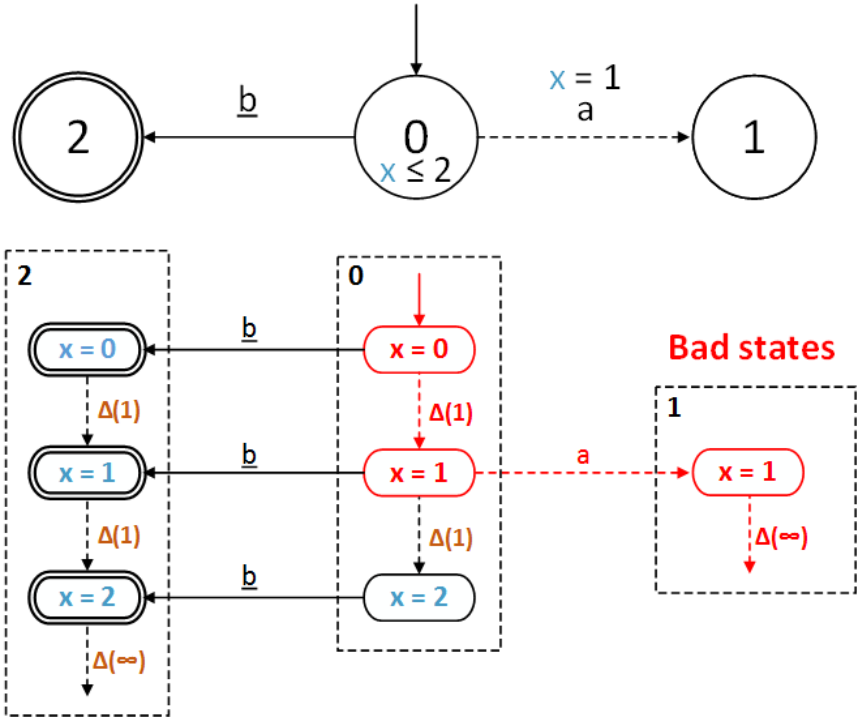


Figure 5.6: *Example TA and its underlying transition system, where the red states are considered bad states*

Because location 1 is not marked and it has deadlock, all of its states are blocking and therefore bad states. As there is an uncontrollable event a pointing from state $(0, x = 1)$ to state $(1, x = 1)$, the former is also considered a bad state. Finally, as forcible events are not considered in this approach, all time delays Δ are considered uncontrollable. Therefore, all states that are time predecessors of a bad state are also considered bad states. For the TA in Figure 5.6, this means that all states $(0, x < 1)$ are also bad states, more specifically, time uncontrollable states. Because the initial conditions for the clocks are assumed to be zero in TA, this means that an empty supervisor should be returned.

The bad state conditions are computed, based on the information that is provided by the current TA supervised plant and the non-blocking conditions of each location. This can be done as follows:

1. Initially, B_i is set to the logical negation of N_i for each location i:

$$B_i^0 := \neg N_i \tag{5.4}$$

2. Then, B_i is computed by repeating the following iteration until the predicate for all locations no longer changes:

$$B_i^{n+1} = B_i^n \vee \bigvee_{\substack{i \xrightarrow{g\ e\ r} j, \\ e \in E_u}} \Big(g \wedge B_j^n[r] \wedge I_j[r] \Big) \vee \exists_\Delta \Big(B_i^n[x + \Delta] \wedge I_i[x + \Delta] \Big). \tag{5.5}$$

Here, B_i^{n+1} is the new bad state condition of source location i, B_i^n is the previous bad state condition of that location, g is the guard of the uncontrollable edge between i and j, $B_j^n[r]$ is the previous bad state condition

of target location j with r referring to a possible reset of one or more of the clocks and $I_j[r]$ is the target invariant with reset r. Note that $\bigvee_{i \xrightarrow{g\ e\ r} j, e \in E_u}$ refers to a disjunction of $(g \wedge B_j^n[r] \wedge I_j[r])$ for all outgoing edges with uncontrollable events E_u of location i and target locations j.

With respect to the SSEFA algorithm, there are two adjustments. First, the target invariant term $I_j[r]$ is included to check the existence of the transition. Here, I_i is not required, as the initial condition B_i^0 already takes into account the source invariant, which is present in N_i. As the source invariant is always the weakest possible condition in N_i, it becomes the strongest possible condition in $B_i^0 = \neg N_i$. Therefore, if the guard does not satisfy the source invariant condition, it does not enlarge the set of bad states. The second adjustment is the addition of the term $\exists_\Delta\Big(B_i^n[x+\Delta] \wedge I_i[x+\Delta]\Big)$, which is added to take into account the possibility that a bad state can be reached via a delay of time. To prevent false indication of a bad state, the delay in the last term must satisfy the location invariant of that state. Also note that the extra disjunction does not depend on the controllability of outgoing events, such that it is always checked.

To illustrate how this algorithm works, the bad state conditions of Example 5.3 are worked out. First, the non-blocking conditions are worked out in Table 5.3 to provide the initial conditions B_i^0.

Table 5.3: *Non-blocking condition N for the TA in Figure 5.6 worked out*

	N		
It	**Loc 0**	**Loc 1**	**Loc 2**
0	f	$\boldsymbol{f}$	t
1	$f \vee (x = 1 \wedge \boldsymbol{f} \wedge x \leq 2) \vee (t \wedge t \wedge x \leq 2) \vee \exists_\Delta(x + \Delta \leq 2 \wedge f) =$ $x \leq 2$	$\boldsymbol{f}$	t
2	$x \leq 2 \vee (x = 1 \wedge \boldsymbol{f} \wedge x \leq 2) \vee (t \wedge t \wedge x \leq 2) \vee \exists_\Delta(x + \Delta \leq 2 \wedge x + \Delta \leq 2) =$ $x \leq 2$	$\boldsymbol{f}$	t

It can be seen that the uncontrollable edge to location 1 is not taken into account in the non-blocking algorithm because the term $N_j^n[r] = false$, as shown in red. The bad state condition computations are worked out in Table 5.4.

Table 5.4: *Bad state conditions for the TA in Figure 5.6 worked out*

	B		
It	**Loc 0**	**Loc 1**	**Loc 2**
0	$x > 2$	t	f
1	$x > 2 \vee (x = 1 \wedge t \wedge t) \vee \exists_\Delta(x + \Delta > 2 \wedge x + \Delta \leq 2)$ $x > 2 \vee (x = 1 \wedge t \wedge t) \vee f =$ $x > 2 \vee x = 1$	t	f
2	$x > 2 \vee x = 1 \vee \exists_\Delta((x + \Delta > 2 \vee x + \Delta = 1) \wedge x + \Delta \leq 2)$ $x > 2 \vee x = 1 \vee x \leq 1 =$ $x > 2 \vee x \leq 1$	t	f

The result coincides with the red states in Figure 5.6. This example shows why the $\exists_\Delta$ term is required, the result after the first iteration: $x > 2 \vee x = 1$ is extended with $x < 1$ as from all $x < 1$ it is possible to delay up to $x = 1$. The result is an empty supervisor, because $(0, x = 0)$ is considered a bad state. From iteration 1 for location 0, it also becomes evident why the source invariant must be satisfied. The $\exists_\Delta$ term returns false, because it is not

possible to delay to condition $x > 2$ because of the invariant $I_0 = x \leq 2$. If this was not added to the equation, the resulting bad state condition would be $true$ after the first iteration. In this example, this would not be a problem, but if the edge with event a to location 1 would not exist, it would still return $true$ event though there is no problem.

5.2.2 Adapt Supervised Plant

In line with the DES indirect synthesis approach, the bad states of the underlying transition system must be removed. In TUDS, it is only possible to adapt the guards based on the bad state conditions. Therefore, the guards must be adapted such that they become stricter, to make sure that no bad state can be reached. The guards of the controllable edges are adjusted according to:

$$i \xrightarrow{g \wedge \neg B_j[r]} j. \tag{5.6}$$

Here, $\neg B_j[r]$ is the logical negation of the bad state condition of the target location, considering a possible reset r. This is the same as in SSEFA, as the source invariant I_i is directly taken into account in the non-blocking condition and the target invariant indirectly via the $N_j^n[r]$ term of the non-blocking condition equation. The bus pedestrian example is used to show how the complete algorithm works.

Example 5.4 – Direct Synthesis on Bus Pedestrian Example: First, the non-blocking conditions are computed in Table 5.5. Note that the conditions for location 1 and 3 are left out, as they are $false$ and $true$ for all iterations.

Table 5.5: *Bus pedestrian example, non-blocking condition*

	N	
It	**Loc 0**	**Loc 2**
0	f	f
1	$f \vee (y \geq 1 \wedge f \wedge x \leq 2) \vee \exists_\Delta(x + \Delta \leq 2 \wedge f) =$ f	$f \vee (x = 2 \wedge t \wedge x \leq 2) \vee$ $\exists_\Delta(x + \Delta \leq 2 \wedge f) =$ $x = 2$
2	$f \vee (y \geq 1 \wedge x = 2 \wedge x \leq 2) \vee \exists_\Delta(x + \Delta \leq 2 \wedge f) =$ $y \geq 1 \wedge x = 2$	$x = 2 \vee \exists_\Delta(x + \Delta \leq 2 \wedge x + \Delta = 2) =$ $x \leq 2$
3	$y \geq 1 \wedge x = 2 \vee (y \geq 1 \wedge x \leq 2 \wedge x \leq 2) \vee$ $\exists_\Delta(x + \Delta \leq 2 \wedge x + \Delta = 2 \wedge y + \Delta \geq 1) =$ $(y \geq 1 \wedge x = 2) \vee (y \geq 1 \wedge x \leq 2) \vee y \geq 0 \wedge x \leq 2 = x \leq 2$	$x \leq 2 \vee \exists_\Delta(x + \Delta \leq 2 \wedge x + \Delta \leq 2) =$ $x \leq 2$
4	$x \leq 2$	$x \leq 2$

This result makes sense with respect to the TA, as it is possible to reach marked location 3 for $x \leq 2$ from both location 0 and 2. Next, the bad state condition is computed in Table 5.6. Note that forcible event $jump$, is not considered in TUDS.

Table 5.6: *Type 2 Bus pedestrian example, bad state condition*

	B	
It	**Loc 0**	**Loc 2**
0	$x > 2$	$x > 2$
1	$x > 2 \vee (x = 2 \wedge t \wedge t) \vee \exists_\Delta(x + \Delta > 2 \wedge x + \Delta \leq 2) =$ $x > 2 \vee x = 2 \vee f =$ $x \geq 2$	$x > 2 \vee (x = 2 \wedge f \wedge t) \vee$ $\exists_\Delta(x + \Delta > 2 \wedge x + \Delta \leq 2) =$ $x > 2 \vee f \vee f =$ $x > 2$
2	$x \geq 2 \vee$ $\exists_\Delta(x + \Delta \geq 2 \wedge x + \Delta \leq 2) =$ $x \geq 2 \vee x \leq 2 =$ t	$x > 2$

Based on the fact that $B_0 = true$, it can be concluded that an empty supervisor is returned.

Finally, an algorithm is proposed to give an overview of how TUDS should work. We refer to this algorithm as Time Uncontrollable Direct Supervisor Synthesis (TUDSSTA).

Algorithm 1 Time Uncontrollable Direct Supervisory Synthesis for TA (TUDSSTA)

procedure TUDSSTA($P^t || R^{p,t}$)
  Initialize iterators: $o := 0$
  repeat
    Initialize iterators: $m := 0,\ n := 0$
    Initialize non-blocking predicate of every location $l \in L$:

$$N_i^{o,0} = \begin{cases} I_i, & \text{if } l_i \in L_m \\ false, & \text{if } l_i \notin L_m \end{cases} \tag{5.7}$$

    repeat
      Update N_i^o of every location $l_i \in L$:

$$N_i^{o,m+1} := N_i^{o,m} \vee \bigvee_{i \xrightarrow{g^o\ e\ r} j} (g^o \wedge N_j^{o,m}[r] \wedge I_i^p) \vee \exists_\Delta (I_i^p[x+\Delta] \wedge N_i^{o,m}[x+\Delta]) \tag{5.8}$$

      $m := m + 1$
    until $\forall l_i \in L, N_i^{o,m} = N_i^{o,m-1}$
    $\forall l_i \in L : N_i^o := N_i^{o,m}$
    Initialize bad state condition predicate for every location $l \in L$:

$$B_i^{o,0} := \neg N_i^o \tag{5.9}$$

    repeat
      Update B_i^o of every location $l_i \in L$:

$$B_i^{o,n+1} := B_i^{0,n} \vee \bigvee_{\substack{i \xrightarrow{g^o\ e\ r} j, \\ e \in E_u}} \Big(g^o \wedge B_j^{0,n}[r] \wedge I_j[r]\Big) \vee \exists_\Delta \Big(B_i^{0,n}[x+\Delta] \wedge I_i[x+\Delta]\Big) \tag{5.10}$$

      $n := n + 1$
    until $\forall l_i \in L, B_i^{o,n} = B_i^{o,-1n}$
    $\forall l_i \in L : B_i^o := B_i^{o,n}$
    Update guard of every edge in $\rightarrow$ with controllable event $e \in E_c$ via:

$$i \xrightarrow{g^{o+1},\ e,\ r} j := i \xrightarrow{g^o \wedge \neg B_j^o[r]} j. \tag{5.11}$$

    $o := o + 1$
  until For all edges of all locations $l_i \in L, g^o = g^{o-1}$
end procedure

Because of the $\exists_\Delta$ term in the bad state condition, the resulting bad state condition is always of the form $x \leq n$ or $n_1 \leq x \leq n_2$. Therefore no extra restrictions or extensions are required for the guards of the supervisor when it is computed with TUDSSTA. Note that when only considering invariants of the form $x < n$ or $x \leq n$, it automatically holds that for all values of x from which there exists a delay that satisfies the invariant, the invariant is satisfied for all possible delays $\Delta' \leq \Delta$. Therefore, the term $\exists_\Delta(I_i[x + \Delta'])$ automatically holds for all $\Delta' \leq \Delta$. For more complex invariants, which are also bounded from below, this is not the case as there are values of x that precede the invariant, but for which a delay exists that satisfies the invariant. Therefore, a generalization to include more complex invariants and guards can be made, by requiring that the source invariant is satisfied for all possible delays. This is written as $(\exists_\Delta(B_i^n[x + \Delta]) \wedge \forall_{\Delta' \leq \Delta}(I_i^p[x + \Delta])$.

5.3 Time Preemptable Direct Synthesis

The TUDS approach is limited, as it only allows delays past bad states as a means to compute a non-blocking, controllable and maximally permissive supervisor[8]. To solve a broader range of problems, the TPDS approach is introduced, which includes forcible events. This allows the supervisor to preempt positive delays of time as long as a forcible event is enabled in the current state of the underlying transition system. As in TUDS, the first step after computing the non-blocking condition, is to determine the bad state condition B, which will be used to adapt the supervised plant until it no longer changes.

The TPDS approach will require a number of extra steps, to take into account invariant adaptation. Firstly, it is necessary to make sure that the guards no longer change as a result of synthesis, before allowing the adaptation of the invariants. The idea is that this prevents premature adaptation of invariants, such that at a later point during synthesis a forcible event is disabled such that its source invariant should never have been adapted. However, as in the TDES indirect synthesis approach, it has not been proven that this is sufficient to prevent all cases of premature invariant adaptation, for instance as a result of the adaptation of other invariants. The problem in TPDS, is that the adaptation of invariants and guards, is constructed such that they can only become more restrictive. Therefore, reintroducing old invariants is not as simple as the reintroduction of τ events. It will also not turn up in the bad state conditions, as the problem no longer exists after invariant adaptation. As we have not thoroughly studied this problem for TPDS, we choose not to provide an algorithm such as in Algorithm 1 in this section. We have added such an algorithm in Appendix D.2, in case it is proven that the problem that is sketched here does not exist. For the sake of clarity we propose a six step algorithm, in which we implicitly take into account the problem of premature invariant adaptation due to invariant adaptation, either by proposing a different adaptation equation for step 5, or introducing a new step that can reintroduce the old invariant or hold on to the old invariant in the bad state computation. This is sufficient for all examples that have been studies, as no example that includes the aforementioned problem has been found. The six step algorithm is as follows:

1. compute the non-blocking conditions for each location of the plant,
2. compute the bad state conditions,
3. adapt the guards of the supervised plant based on these conditions,
4. if all guards are the same after a subsequent iteration continue to step 5, otherwise return to step 1,
5. adapt the location invariants of the supervised plant, according to the bad state conditions,
6. if the resulting invariants of all locations are the same after a subsequent iteration, this is the resulting supervisor, otherwise return to step 1.

Here, the non-blocking conditions are computed in the same fashion as for TUDS, and the bad states are extended with the notion of forcible events. At first, only the guards of $S^t||P^t$ are adapted until they no longer change. This is done to make sure that invariants are not adapted prematurely as a result of forcible event guard adaptation, e.g. when a controllable forcible event initially points towards a location that becomes blocking at some point, such that it is disabled. When all of the guards remain the same with respect to the previous iteration, the bad state conditions of the locations are used to adapt the location invariants. We implicitly assume that prematurely adapted invariants are reintroduced in step 5. Finally, if the invariants are unchanged with respect to the previous iteration, the remaining TA is the supervisor.

5.3.1 Compute Bad State Condition

With the introduction of forcible events, not all delays are considered uncontrollable. Therefore, only the states that point to a bad state via an uncontrollable delay in the underlying transition system, should also be considered bad states. Example 5.5 helps to visualize this.

Example 5.5 – TPDS Bad state visualization: Consider the TA and its underlying transition system, shown in Figure 5.7. It is assumed that event b is a forcible event.

[8] It is maximally permissive in the same way that DES indirect synthesis is. Namely, in accordance with DES SCT based on the underlying transition system.

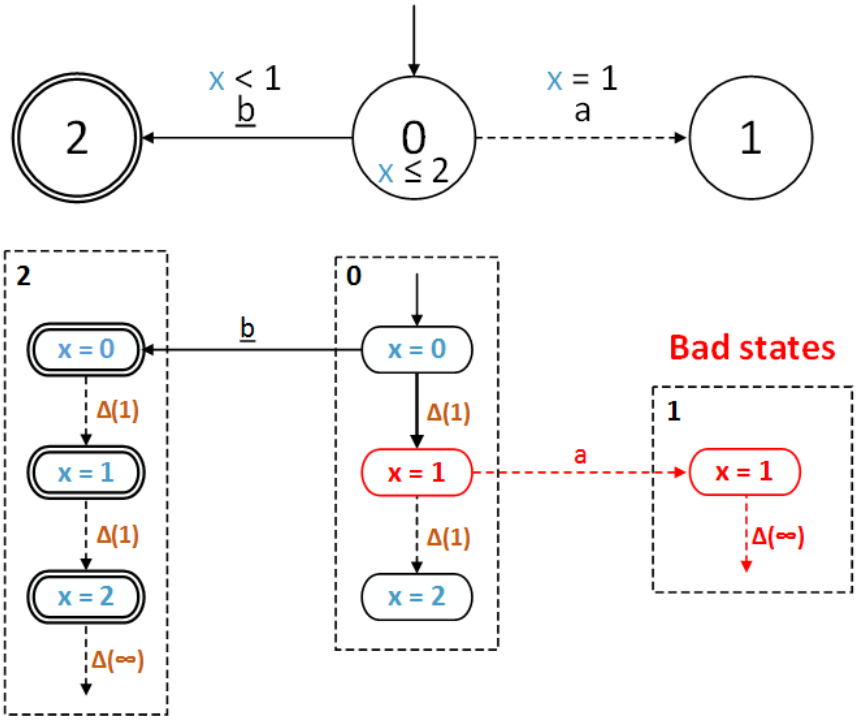


Figure 5.7: *Example TA and its underlying transition system, where the red states are bad states and event b is forcible*

Because location 1 has deadlock, all of its states are bad states. As there is an uncontrollable event a pointing from state $(0, x = 1)$ to state $(1, x = 1)$, the former is also considered a bad state. Because the guard $x < 1$ on the b-labelled transitions allows event b to be taken before $x = 1$, all states $(0, x < 1)$ have preemptable delays Δ. Therefore, they are not considered as bad states.

Based on this example, it is evident that the bad state condition in TPDS requires an extension with respect to the bad state condition in TUDS. Initially, it is expected that the bad states in the underlying transition system are those states that can reach another bad state via an uncontrollable event or via a time delay from a state that does not have an outgoing forcible event. This is similar to the TDES indirect synthesis notion of bad states. However, in TDES indirect synthesis, the τ events represent a fixed range of possible delays based on the region to which it transitions. In direct synthesis, up until now, we have not imposed any restrictions on the time delay Δ other than that it should be non-negative. Therefore, the notion of bad states as in TDES indirect synthesis is not specific enough. It has to take into account that each delay Δ can be split into any m number of subsequent time transitions $\Delta_1 + \Delta_2 ... + \Delta_m$ [28], and that one or more of these time transitions may be controllable due to the availability of a forcible event. This becomes evident when imagining Example 5.3.1, but now with the guard $x = 0.5$ for the b-labelled transition. Figure 5.8 shows what the underlying states with location 0 would look like in this case. Note that the states are rotated with respect to Figure 5.7 and that a c-labelled edge without a guard is added such that the non-blocking condition is always initially $N_0 = x \leq 2$.

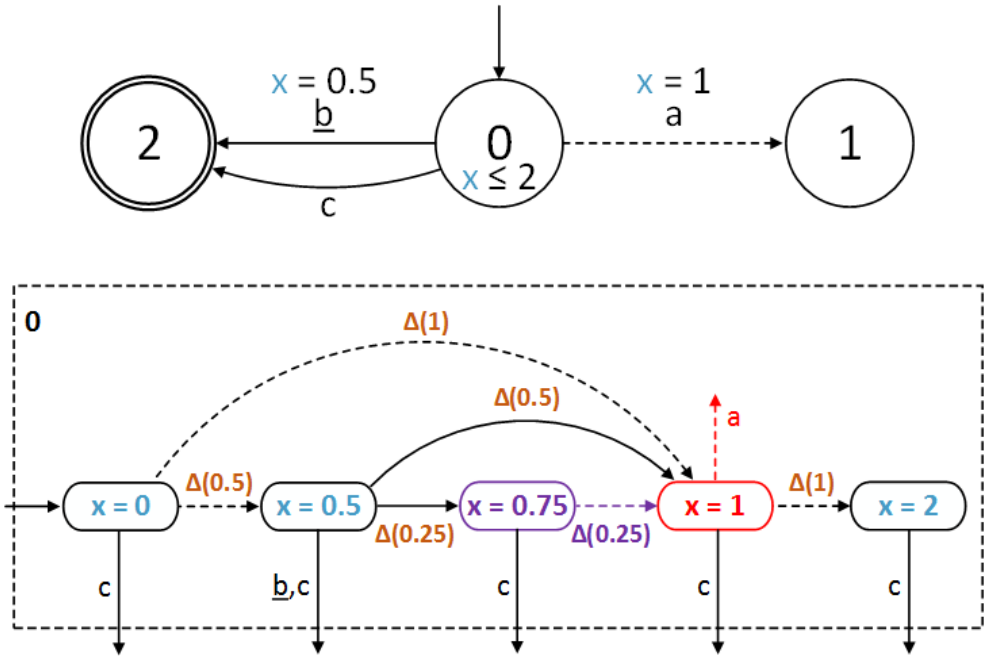


Figure 5.8: *Underlying states space of Example 5.3.1, now with guard guard $x = 0.5$ for the b-labelled transition*

A possible delay from state $(0, x = 0)$ is $\Delta(1)$, which leads directly to the bad state $(0, x = 1)$. If we were to consider the TDES synthesis notion of the bad state, the fact that there exists a time-transition to the bad state and state $(0, x = 0)$ does not have any outgoing forcible events, would lead to the conclusion that state $(0, x = 0)$ is a bad state. This also holds for all states $(0, x < 0.5)$ and $(0.5 < x \leq 1)$, i.e. all states with uncontrollable outgoing Δ-labelled transitions, preceding $(0, x = 1)$. However, this is not the correct conclusion, as for all states $(0, x \leq 0.5)$ it is guaranteed that when state $(0, x = 0.5)$ is reached, forcible event b can be used to force a transition out of location 0 (i.e., the controllable time delays can be removed during synthesis). The same problem appears when considering interval guards such as $0.25 < x < 0.75$ and lower bounded guards such as $x \geq 0.5$.

The correct conclusion is highlighted purple in Figure 5.8. As there is no controllable Δ within the purple time-transition $\Delta(0.25)$, none of the intermediate delays can be preempted by a forcible event. Therefore, state $(0, x = 0.75)$ must be considered as a bad state. Figure 5.9 gives a more specific view of the situation.

Figure 5.9: *Smallest possible delay $\Delta_{>0}$ from $x = 0.5$, showing that all states $0.5 < x \leq 1$ should be considered as bad states*

When considering the smallest possible delay $\Delta_{>0}$ from state $(0, x = 0.5)$, it can be seen that not only state $(0, x = 0.75)$ must be considered as a bad state, but all states $(0, 0.5 < x \leq 1)$ as well. Based on this, the improved condition for determining the bad state condition as a result of time delays, i.e. the time uncontrollable states, is defined as follows: there exists a delay Δ to an existing bad state and for all possible delays $\Delta' \leq \Delta$ to that existing bad state, there does not exist a delay $\Delta'' < \Delta'$ to any state in which a forcible event exists that points to a not bad state. This time uncontrollable state condition TU can be described for one outgoing forcible edge with guard g_f from location i to location $k \neq i$, as follows:

$$TU := \Big(\exists_{\Delta} \big(B_i^n[x + \Delta] \wedge I_i^p[x + \Delta] \big) \wedge \forall_{\Delta' \leq \Delta} \big(B_i^n[x + \Delta'] \wedge I_i^p[x + \Delta'] \big) \nexists_{\Delta'' < \Delta'} \Big(g_f[x + \Delta''] \wedge I_k^p[r][x + \Delta''] \wedge \neg B_k^n[r][x + \Delta''] \Big) \Big). \tag{5.12}$$

In the following, we generalize to multiple forcible edges and explain in more depth what the additions are, with respect to TUDS.

As in TUDS, the bad state conditions are computed based on the information that is provided by the current TA supervised plant and the non-blocking conditions of each location. This can be computed a follows:

1. Initially, B_i is set to the logical negation of N_i for each location i:

$$B_i^0 := \neg N_i \tag{5.13}$$

2. Then, bad state B_i is computed for each location i, for all edges $i \xrightarrow{g\ e\ r} j$ with $e \in E_u$ and $i \xrightarrow{g\ e\ r} k$ with $e \in E_f$ and $k \neq i$, by repeating the following iteration until the predicate for all locations no longer changes:

$$\begin{aligned} B_i^{n+1} = B_i^n \vee & \bigvee_{i \xrightarrow[e \in E_u]{g\ e\ r} j,} \left(g \wedge B_j^n[r] \wedge I_j[r]\right) \\ \vee & \Big(\exists_\Delta \left(B_i^n[x+\Delta] \wedge I_i^p[x+\Delta]\right) \wedge \forall_{\Delta' \leq \Delta} (B_i^n[x+\Delta'] \\ & \wedge I_i^p[x+\Delta']) \nexists_{\Delta'' < \Delta'} \bigvee_{i \xrightarrow[e \in E_f,\ k \neq i]{g\ e\ r} k,} \left(g_f[x+\Delta''] \wedge I_k^p[r][x+\Delta''] \wedge \neg B_k^n[r][x+\Delta'']\right)\Big). \end{aligned} \tag{5.14}$$

   With respect to the TUDS algorithm (Equation 5.5), there is one new addition. Namely, the term that represents all possible delay to the bad state such that there does not exist an intermediate delay to any state with an existing forcible event pointing to a not bad state, i.e. the term with $\forall_{\Delta' \leq \Delta}$ and all that lies behind it. The additional constraint that $\Delta'' < \Delta'$ is required to make sure that only those forcible events that are enabled before the bad state is reached are considered. The $\bigvee_{i \xrightarrow{g\ e\ r} k,\ e \in E_f,\ k \neq i}$ refers to a disjunction of $\left(g_f[x+\Delta''] \wedge I_k^p[r][x+\Delta''] \wedge \neg B_k^n[r][x+\Delta'']\right)$ for all edges from source location i, with event $e \in E_f$ and target location k, which is not the source location i, i.e. $k \neq i$. This makes sure that the conditions under which any of the forcible events are enabled are taken into account and also that the term is not checked if no such edge with a forcible event exists from the source location. The latter effectively means that the TPDS bad state condition algorithm reduces to the TUDS algorithm in Equation 5.5. Note that I_k^p is the invariant of target location k, which is used to check under which conditions other than imposed by the guard, the edge with the forcible event exists. Additionally, $\neg B_k^n$ is used to make sure only those conditions under which the forcible edge points to the states of location k that are not bad, are considered.

Figure 5.10 shows how the conclusion of the new term is determined for state $(0, x = 0)$ and $(0, x = 0.5)$ of the TA in Figure 5.8 with $x = 0.5$ the guard of the forcible b-labelled transition. Note that only bad state $(0, x = 1)$ is considered here, as the term $\exists_\Delta \left(B_i^n[x+\Delta] \wedge I_i^p[x+\Delta]\right)) = \exists_\Delta \left((x = 1 \vee x > 2)[x+\Delta] \wedge x \leq 2[x+\Delta])\right)$ reduces to $\exists_\Delta (x + \Delta = 1)$.

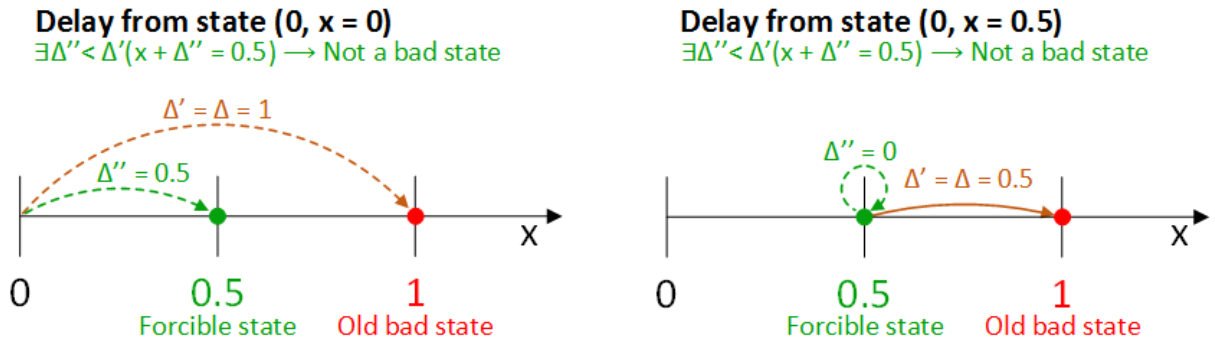


Figure 5.10: *How to determine if a state is a new bad state, here both state* $(0, x = 0)$ *and* $(0, x = 0.5)$ *are not bad states.*

From state $(0, x = 0)$, there is only one delay $\Delta = 1$ to bad state $(0, x = 1)$, therefore $\Delta' = \Delta$ in this example. It can be seen in the left clock graph in Figure 5.10, that it is also possible to delay from $(0, x = 0)$ to $(0, x = 0.5)$ with a delay $\Delta'' = 0.5$ such that there exists a $\Delta'' < \Delta'$ to state $(0, x = 0.5)$ with outgoing forcible event b. Therefore, state $(0, x = 0)$ is not a new bad state, as there is a controllable delay between it and the old bad state $(0, x = 1)$. The same conclusion results when checking state $(0, x = 0.5)$, as the only delay to the bad state is $\Delta' = \Delta = 0.5$ and state $(0, x = 0.5)$ can be reached with $0 = \Delta'' < \Delta'$. In Figure 5.11, state $(0, x = 0.75)$ is checked, which turns out to be a bad state, as was concluded from Figure 5.8. Additionally, in the right figure, the result is given after considering all states of location 0.

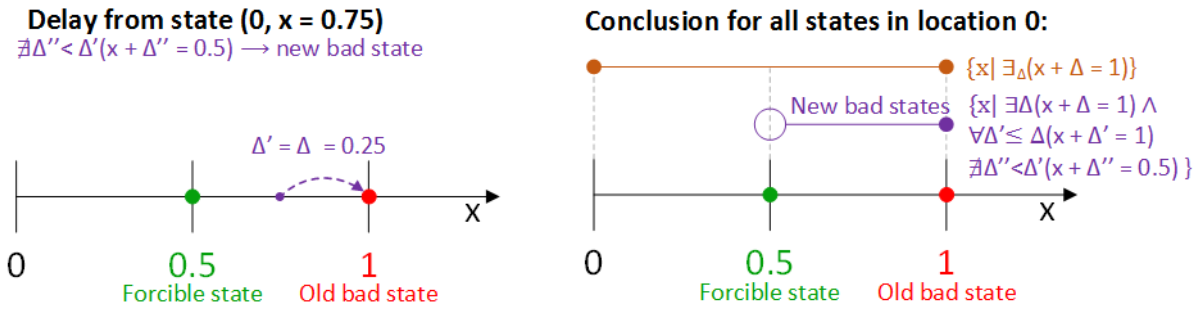


Figure 5.11: *How to determine if a state is a new bad state, here both state* $(0, x = 0.75)$ *is a bad state and in general all states* $(0, 0.5 < x \leq 1)$ *are bad states due to time uncontrollability too*

It can be seen in the left figure why $(0, x = 0.75)$ is a new bad state, there is no non-negative delay $\Delta'' < \Delta' = 0.25$ such that state $(0, x = 0.5)$ can be reached. Therefore, it is not possible to reach any state from which a preemption is possible via forcible event b. The condition that results after checking all states, is shown in the right figure. It is concluded that for $0.5 < x \leq 1$ in location 0, no controllable delays exist before reaching state $(0, x = 1)$. The new bad state condition is the disjunction of the old and the new, recall that condition $x > 2$ was the initial bad state condition and $x = 1$ is a bad state due to uncontrollable event a pointing to deadlock location 1 in Figure 5.8. Taking the disjunction of these bad states with the new ones that result from uncontrollable time delays, results in a bad state condition: $0.5 < x \leq 1 \vee x > 2$. To show how the bad states are iteratively determined in synthesis, the bad state computation is worked out in Table 5.7.

Table 5.7: *Bad state condition for TA in Figure 5.8 worked out.*

	B		
It	**Loc 0**	**Loc 1**	**Loc 2**
0	$x > 2$	t	f
1	$x > 2 \vee (x = 1 \wedge t \wedge t) \vee (f \wedge ...)$	t	f
2	$(x = 1 \vee x > 2) \vee \Big(\exists_\Delta (x + \Delta = 1)$ $\wedge \forall_{\Delta' \leq \Delta} (x + \Delta' = 1) \nexists_{\Delta'' < \Delta'} (x + \Delta'' = 0.5)\Big) =$ $x = 1 \vee x > 2 \vee 0.5 < x \leq 1 =$ $0.5 < x \leq 1 \vee x > 2$	t	f
2	$0.5 < x \leq 1 \vee x > 2$	t	f

The expected bad state condition follows from working out the algorithm. Note that it is also interesting to consider those cases for which the forcible event is only enabled from or after the bad states. For instance, the case where the guard of the b-labelled transition in Figure 5.8 is $x = 1$, the case for which the guard is $x > 1$ and the case for which both the guard and the bad state condition are $x > 1$. All of these cases result in $B_0 = true$, as the term that checks for uncontrollable time delays, returns $true$. This makes sense, as it is not possible to preempt a delay Δ to the bad state if the forcible event is not enabled before the bad state or if it already is in a bad state. All of these cases are covered by $\forall_{\Delta' \leq \Delta}(B_i^n[x + \Delta'] \wedge I_i^p[x + \Delta'])\nexists_{\Delta'' < \Delta'}$, as one of the possible delays $\Delta' \leq \Delta$ to the bad state, if you are already in the bad state, is $\Delta' = 0$ such that no $\Delta'' < \Delta'$ can exist as all delays must be non-negative.

To extend the bad state condition equation, to take into account all types of invariants, it can be adapted as follows:

$$\begin{aligned} B_i^{n+1} = B_i^n \vee & \bigvee_{i \xrightarrow[e \in E_u]{g\ e\ r} j,} \left(g \wedge B_j^n[r] \wedge I_j[r]\right) \\ \vee & \Bigg(\exists_\Delta (B_i^n[x+\Delta]) \wedge \forall_{\Delta' \leq \Delta} (I_i^p[x+\Delta']) \wedge \forall_{\Delta' \leq \Delta} (B_i^n[x+\Delta'] \\ \wedge & I_i^p[x+\Delta']) \nexists_{\Delta'' < \Delta'} \bigvee_{\substack{i \xrightarrow{g\ e\ r} k, \\ e \in E_f, \\ k \neq i}} \Big(g_f[x+\Delta''] \wedge I_k^p[r][x+\Delta''] \wedge \neg B_k^n[r][x+\Delta''] \Big) \Bigg). \end{aligned} \tag{5.15}$$

Here, instead of checking if there exists a delay to the bad state that lies within the invariant, it is checked if there exists a delay such that the bad state is reached and for all possible $\Delta' \leq \Delta$ the invariant must be satisfied. This is only the case if x satisfies the invariant. This check is done with the term $\forall_{\Delta' \leq \Delta}(I_i^p[x+\Delta'])$ in conjunction with the existence of a delay to the bad state.

5.3.2 Adapt Supervised Plant

In EFA, it is assumed that the supervisor may only enable or disable events and it cannot force them. This is also the assumption that was made in the indirect synthesis approach in [17], DES indirect synthesis and TUDS. The only means of control is then to adapt the guards of controllable edges such that they become stricter. In addition to this, we introduced a concept of forcibility through invariants. This provides a second means of control, namely by adapting the invariants to allow the TA to enter a location before a bad state that can be reached by time delays.

Based on the aforementioned six step algorithm, the guards of the controllable edges are adjusted according to:

$$i \xrightarrow{g \wedge \neg B_j[r]} j. \tag{5.16}$$

This is the same as in SSEFA, as the source invariant I_i is directly taken into account in the non-blocking condition and the target invariant indirectly via the $N_j^n[r]$ term of the non-blocking condition equation. We assume that invariants can be adapted via:

$$I_i := I_i \wedge \neg B_i. \tag{5.17}$$

No additional restriction in terms of forcible event availability is required, as this is already taken into account in the bad state condition. Note that as in the TDES indirect synthesis approach, expressions that differ from the originally permitted guards and invariants may be present in the bad state conditions. To take this into account, we assume that the input TAs contain the restricted guards and invariants, but the guards and invariants may be adapted such they are not restricted. The bus pedestrian example is used to show how the complete algorithm would work.

Example 5.6 – Direct Synthesis on Bus Pedestrian Example: First, the non-blocking conditions are computed in Table 5.8. Note that the conditions for location 1 and 3 are left out, as they are $false$ and $true$ for all iterations.

Table 5.8: *Bus pedestrian example, non-blocking condition*

	N	
It	**Loc 0**	**Loc 2**
0	f	f
1	$f \vee (y \geq 1 \wedge f \wedge x \leq 2) \vee$ $\exists_{\Delta}(x + \Delta \leq 2 \wedge f) =$ f	$f \vee (x = 2 \wedge t \wedge x \leq 2) \vee$ $\exists_{\Delta}(x + \Delta \leq 2 \wedge f) =$ $x = 2$
2	$f \vee (y \geq 1 \wedge x = 2 \wedge x \leq 2) \vee$ $\exists_{\Delta}(x + \Delta \leq 2 \wedge f) =$ $y \geq 1 \wedge x = 2$	$x = 2 \vee \exists_{\Delta}(x + \Delta \leq 2 \wedge x + \Delta = 2) =$ $x \leq 2$
3	$y \geq 1 \wedge x = 2 \vee (y \geq 1 \wedge x \leq 2 \wedge x \leq 2) \vee$ $\exists_{\Delta}(x + \Delta \leq 2 \wedge x + \Delta = 2 \wedge y + \Delta \geq 1) =$ $(y \geq 1 \wedge x = 2) \vee (y \geq 1 \wedge x \leq 2) \vee y \geq 0 \wedge x \leq 2 = x \leq 2$	$x \leq 2 \vee \exists_{\Delta}(x + \Delta \leq 2 \wedge x + \Delta \leq 2) =$ $x \leq 2$
4	$x \leq 2$	$x \leq 2$

This result makes sense with respect to the TA, as it is possible to reach marked location 3 for $x \leq 2$ from both location 0 and 2. Next, the Bad state condition is computed in Table 5.9. Here, there exists a forcible event $jump$ from location 0, with guard $y \geq 1$ to represent the response time of the pedestrian. Note that when including multiple clocks, relative values become of importance. For instance if we assume that it is allowed to start with initial conditions $x = 1.5$ and $y = 0$, then the pedestrian cannot jump in time. Therefore, for this example, in addition to the expected bad state condition $B_0 = x \geq 2$, the condition $B_0 = y + 1 \leq x$ would also be a bad state condition. This would require a constraint $y + 1 > x$ on the initial conditions. However, as we assume that all clocks start from zero initially, we simplify the computation by using $x \geq 1$ as the guard of the forcible edge, as in this case $x = y$ for all underlying states of the bus pedestrian TA. Note that the term $\nexists_{\Delta'' < \Delta'}(x + \Delta'' \geq 1 \wedge x + \Delta'' \leq 2 \wedge x + \Delta'' \leq 2)$ is represented as $\nexists_{\Delta'' < \Delta'}(x + \Delta'' \geq 1 \wedge x + \Delta'' \leq 2)$ in iteration 2 of location 0, to save space as it is a duplicate predicate due to the fact that $I_2 = \neg B_2$ in all iterations. The same holds for $\exists_{\Delta}(x + \Delta > 2 \wedge x + \Delta \leq 2) = \exists_{\Delta}(f[x + \Delta])$ in iteration 0 of location 0 and 2. For location 0 the conjunction with the rest of the term ($\wedge$...) is left out as it will inevitably return $false$. For location 2, this conjunction term does not exist as there is no outgoing forcible event $jump$ from that location.

Table 5.9: *Bus pedestrian example, worked out bad state conditions for TPDS*

	B	
It	**Loc 0**	**Loc 2**
0	$x > 2$	$x > 2$
1	$x > 2 \vee (x = 2 \wedge t \wedge t) \vee$ $(\exists_{\Delta}(f[x + \Delta]) \wedge ...) =$ $x \geq 2$	$x > 2 \vee (x = 2 \wedge f \wedge t) \vee \exists_{\Delta}(f[x + \Delta]) =$ $x > 2$
2	$x \geq 2 \vee (\exists_{\Delta}(x + \Delta = 2) \wedge$ $\forall_{\Delta' \leq \Delta}(x + \Delta' = 2) \nexists_{\Delta'' < \Delta'}(x + \Delta'' \geq 1 \wedge x + \Delta'' \leq 2)) =$ $x \geq 2 \vee (x \geq 2) =$ $x \geq 2$	$x > 2$

Based on the results from Table 5.9, the guard of edge $i \xrightarrow{y \geq 1} j$ is adapted to $i \xrightarrow{y \geq 1 \wedge x \leq 2} j$, in accordance with $B_2 = x > 2$ in the lower right cell of Table 5.9. The other two edges have uncontrollable events such that they cannot be adapted. As nothing changes in terms of disablement of forcible event $jump$, a subsequent iteration gives the same result after adapting the guards. Now, based on $B_0 = x \geq 2$, the invariant of location 0 is adapted to $I_0 = x \leq 2 \wedge x < 2 = x < 2$. After adapting both the guard and the invariant, the result is as follows:

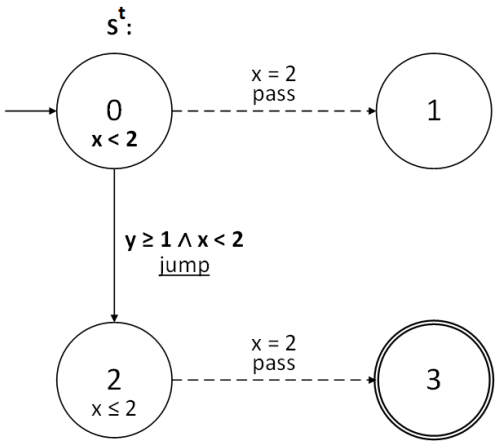


Figure 5.12: *Result after direct synthesis of bus pedestrian example*

It can be seen that a similar result to that of the TDES indirect synthesis approach results, with much less effort. However, the bus pedestrian example does not take into account the possibility that a forcible event can be disabled at some point. Adapting the guards until they do not change, before adapting the invariants solves this. We visualize this by means of Example 5.7.

Example 5.7 – Forcible event to bad state: consider the TA in Figure 5.13 where event a is a forcible event on an edge that is eventually disabled.

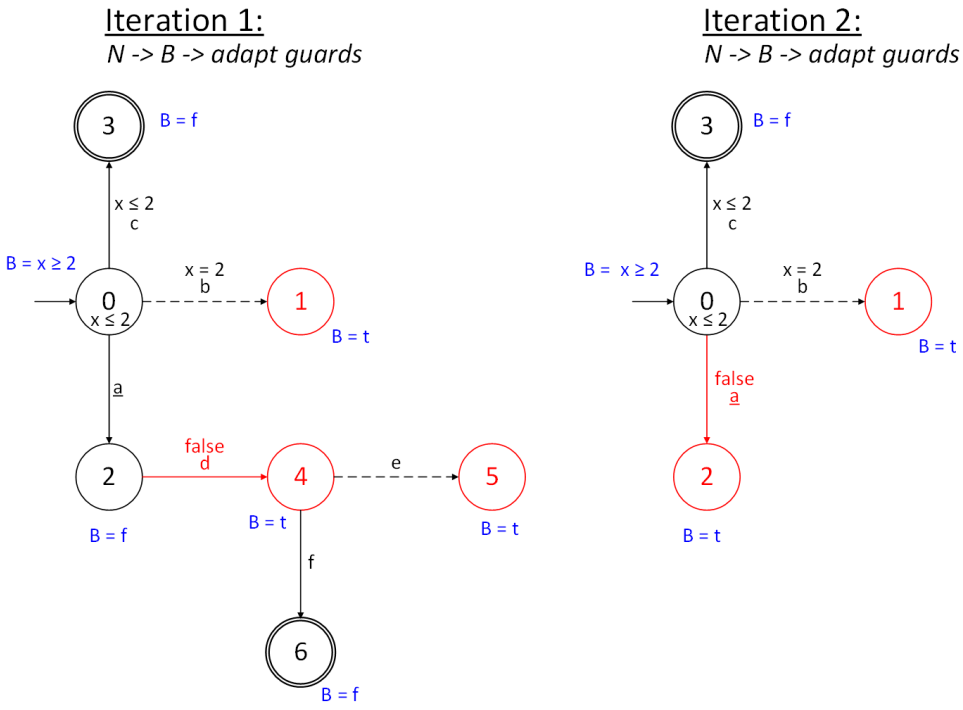


Figure 5.13: *Example with forcible event pointing to bad state, here it is shown what the result is of adapting the guards first until they do not change and then the invariants, iteration 1 and 2*

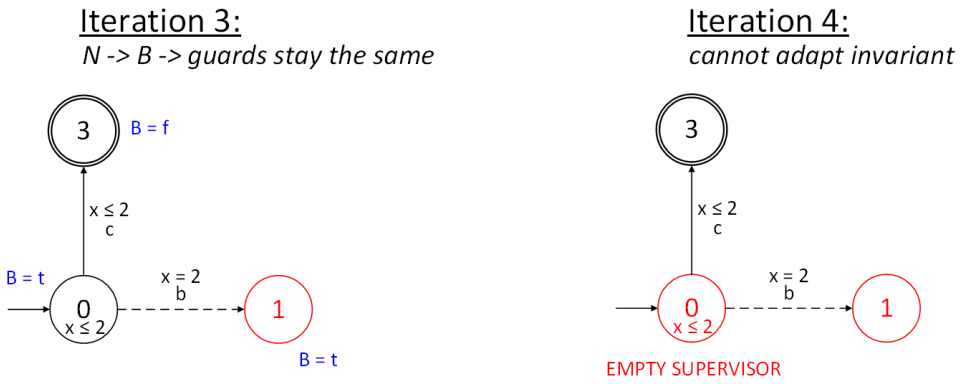


Figure 5.14: *Example with forcible event pointing to bad state, here it is shown what the result is of adapting the guards first until they do not change and then the invariants, iteration 3 and 4*

After the first iteration, it is determined that $N_5 = false$ and $N_1 = false$, then using B_2 for location 4, it is determined that $B_4 = true$. As event b is uncontrollable, only the edge with event d can be disabled by setting the guard to $false$. Because of this, location 2 becomes a deadlock location. This is computed in iteration 2, after which the guard is adapted to $false$ to disable the edge with forcible event a. In iteration 3 it is determined that the guards no longer change. As forcible event a has been disabled, bad state condition $B_0 = x \leq 2 \vee x \geq 2 = true$ results. Therefore, an empty supervisor results. This is seen as the correct result for this problem, as there is no forcible event that can preempt the delay to state $(0, x = 2)$ from occurring.

To show what happens when both the guards and invariants are adapted directly after each other, Figure 5.15 and 5.16 give an overview of the three iterations that would result.

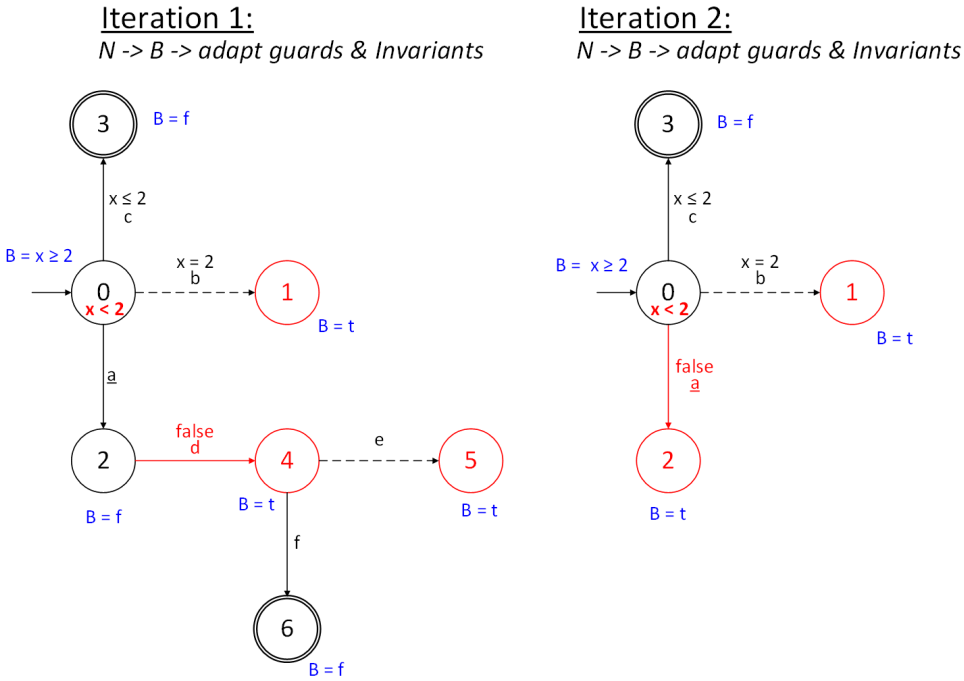


Figure 5.15: *Example with forcible event pointing to bad state, here it is shown what happens if both guards and invariants are adapted simultaneously*

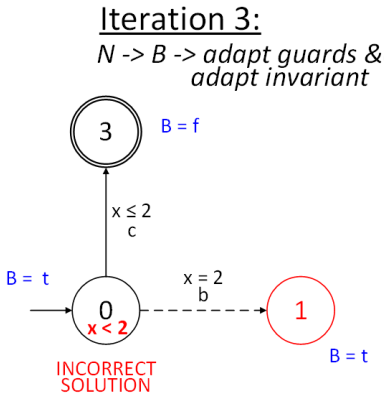


Figure 5.16: *Example with forcible event pointing to bad state, here it is shown what happens if both guards and invariants are adapted simultaneously*

The problem that arises already shows in iteration 1, as I_0 is adapted to $x < 2$ prematurely. This is possible, because a is available from location 0 at this point. Therefore $B_0 = x \geq 2$. However, after the second and third iteration, a is disabled, such that the delay to $x = 2$ cannot be preempted. Therefore, the result is considered incorrect.

As mentioned in the introduction of this section, we have no proof to show that such premature adaptations of the invariants cannot result from adapting other invariants. Therefore, we choose to not introduce an algorithm here, as we do not have a method to reintroduce the original constraints of prematurely adapted invariants. However, we have not yet found an example in which such a case occurs. Therefore, the algorithm that can be used if it is proven that this problem does not exist, is shown in Appendix D.2. Even if the problem does exist the non-blocking and bad state computations will remain the same. Possibly changes to the adaptation will be required to take into account that the constraints must effectively be weakened to their initial forms. Alternatively a new step that can overrule the adaptation could prove to be a solution. Either way, we consider this as one of the main challenges that remains to be solved.

5.4 Comparison of approaches

In this subsection, we first compare the indirect synthesis approaches with the direct ones by means of a simple example. Then, we conclude the section with a brief discussion on the topic of implementability of the direct synthesis approaches.

To compare the indirect and direct synthesis approaches, we revisit Example 4.4 from Section 4.2. First, DES indirect synthesis is compared to TUDS in Example 5.8.

Example 5.8 – Unreachable bad states: consider the Input TA in Figure 5.17, with no forcible events and an uncontrollable event b pointing to location 2 for which all states are blocking as a result of deadlock.

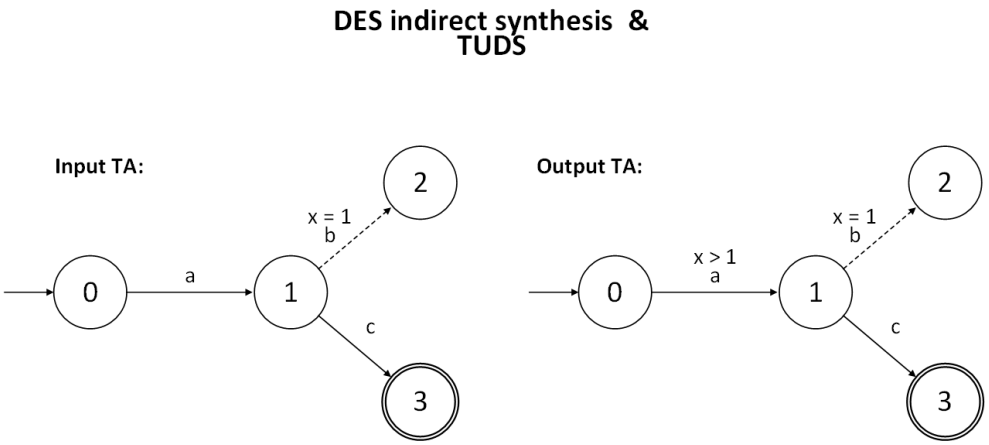


Figure 5.17: *Unreachable bad state example, no forcible events*

In the DES indirect synthesis approach, the REQG is constructed as shown in Figure 5.18.

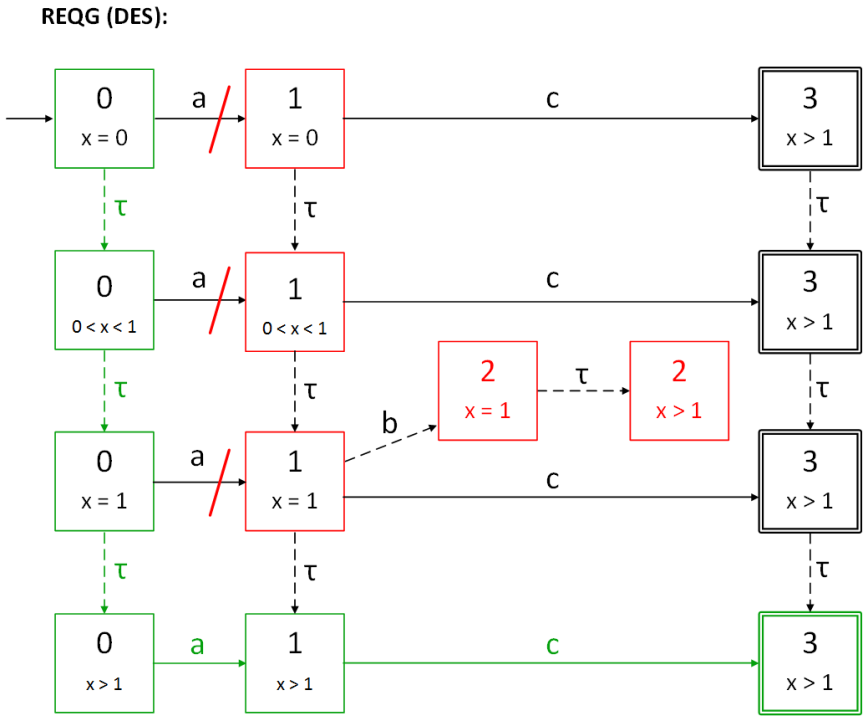


Figure 5.18: *REQG showing DES indirect synthesis procedure, the green path is the resulting untimed supervisor*

It can be seen that the blocking states are prevented by disabling the edges with event a for all states $(0, 0 \leq x \leq 1)$. The resulting FA supervisor is shown in green. The TA result is the Output TA as shown in Figure 5.17. The same output results when applying the direct synthesis algorithm. In this case, the non-blocking conditions are $true$ for all locations other than location 2 and the bad state conditions are worked out in Table 5.10.

Table 5.10: *DSSA bad state condition worked out for the input TA in Figure 5.17*

	B			
It	**Loc 0**	**Loc 1**	**Loc 2**	**Loc 3**
0	f	f	t	f
1	f	$f \vee (x = 1 \wedge t \wedge t) \vee \exists_{\Delta}(f \wedge x + \Delta \leq 2)$ $f \vee (x = 1 \wedge t \wedge t) \vee f =$ $x = 1$	t	f
2	f	$x = 1 \vee \exists_{\Delta}(x + \Delta = 1 \wedge x + \Delta \leq 2)$ $x = 1 \vee x \leq 1 =$ $x \leq 1$	t	f

In this table, it can be seen that $B_1 = x \leq 1$ is the result such that the guard is adapted according to $0 \xrightarrow{t \wedge x > 1,\ a,\ -} 1$, which is the result in Figure 5.17. Note that TPDS takes into account the availability of forcible events, such that if there are none, the bad state computation equation reduces to that of TUDS. This is also necessary for TPDS as there may be some locations without forcible events.

Now, we apply TDES indirect synthesis and TPDS to the Input TA in Figure 5.17, but we assume that event c is forcible.

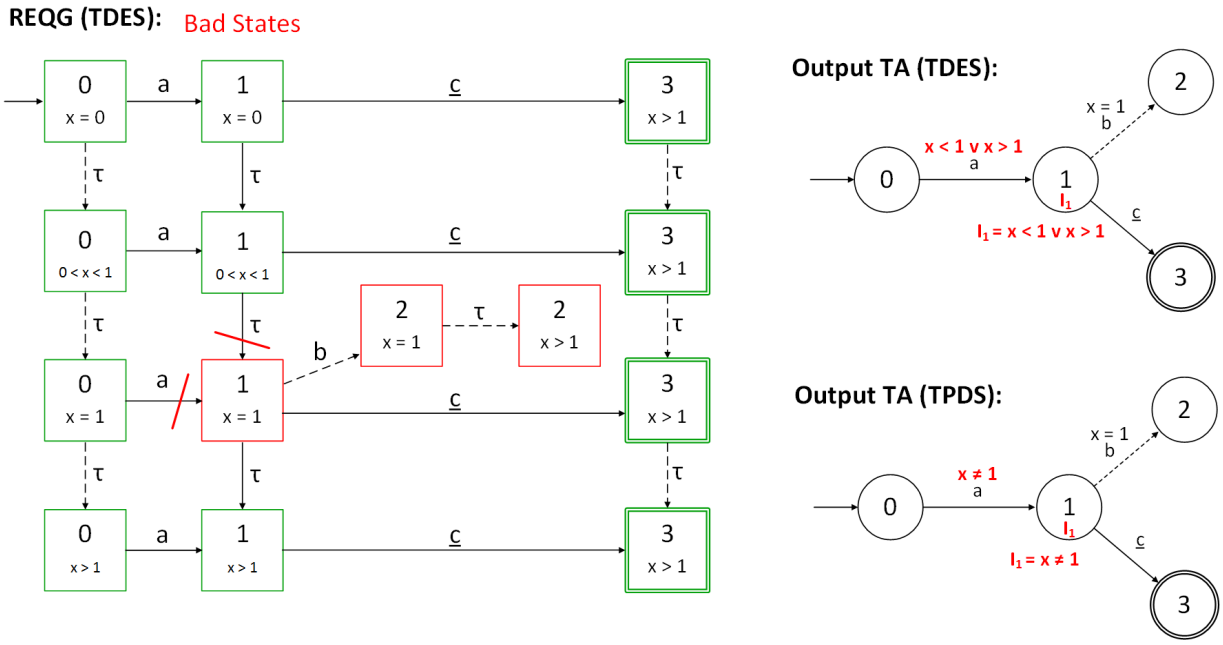


Figure 5.19: *REQG showing TDES indirect synthesis procedure, with the FA supervisor in green, the resulting output TA that is reconstructed from this FA supervisor: Output TA(TDES), and the similar supervisor that results from TPDS*

The REQG in Figure 5.19 shows a number of steps at once. First the blocking states are determined, then the other bad states. The controllable event a is then removed as it points to bad state $(0, x = 1)$. The bad state computation does not result in any new bad states, the same are still reachable via controllable time jump event τ. As no other controllable events can be removed, τ is removed. The resulting guard and invariant adaptation is also shown in Figure 5.19 in the Output TA (TDES), it can be seen that these represent the states of the resulting FA supervisor as highlighted green in the REQG. The bad state condition computation for direct synthesis is worked out in Table 5.11, the resulting Output TA (TPDS) is shown in Figure 5.19.

Table 5.11: *DSSA bad state condition worked out for the Iteration 2 TA in Figure 5.19*

	B			
It	**Loc 0**	**Loc 1**	**Loc 2**	**Loc 3**
0	f	f	t	f
1	f	$f \vee (x = 1 \wedge t \wedge t) \vee (\exists_{\Delta}(f))$ $x = 1$	t	f
1	f	$x = 1 \vee (\exists_{\Delta}(x + \Delta = 1) \wedge \forall_{\Delta' \leq \Delta}(x + \Delta' = 1) \nexists_{\Delta'' < \Delta'}(t)) =$ $x = 1 \vee (x = 1) =$ $x = 1$	t	f

It can be seen that both approaches provide an equivalent result. The difference lies in the adaptation of the invariants versus the construction of the invariants. The TPDS approach uses the negation of the bad state conditions, whilst the TA reconstruction after TDES indirect synthesis determines the invariants based on the remaining existing states of each location. If the invariants were constructed by taking the conjunction of the original guards with the negation of the disjunction of the removed states, the same result would follow. This makes sense, as this is exactly what we intend to model with the invariants in TA, the states that still exist after synthesis. The comparison in this example does not guarantee that this equivalence will be the case for all problems, this remains to be proven. A good starting point for a proof is to determine if the derivation of the bad states is equivalent as the TDES case is known.

We conclude this section with a brief discussion on the topic of implementation. Adapting guards and invariants should not be a problem. Simple conjunctions and disjunctions of predicates have already been considered in SSEFA, but it is the extension with terms of the form $\exists_{\Delta}$ in the non-blocking and bad state conditions, that differentiates TUDS and TPDS from the other approaches. Although it might be quite simple to compute the result of this term by hand for small examples, the extension to multiple dimensions and the means to find and answer automatically may prove to be a larger challenge. The intermediate step is to draw a clock graph and then to determine for which values of the clock there exists a delay such that the predicates in the term are satisfied. What happens when multiple clocks are introduced, has not been thoroughly studied yet. It may be necessary to construct mini zone automata with which the existence of a delta such that the predicates are satisfied can be checked. If this is the case, it is expected that this would still be a large improvement with respect to applying the region graph abstraction to convert the entire input TA to an REQG.

6 Conclusions & Recommendations

The two goals of this study were to: 1) find a means to link the TDES SCT notion of forcibility to TA, and to apply these notions to an indirect synthesis approach to construct a TA supervisor using the FA supervisor that results, and 2) research the possibility to directly apply synthesis to TA, without the need to transform the entire TA to an FA. First of all, we have found a means to link the TDES SCT notion of forcibility to TA via invariants. Based on this, we define two types of SCT problems for TA; problems without forcible events and problems with forcible events. To solve these problems, we first considered two supervisor synthesis approaches that use an abstraction of exact time by means of finite automata that are constructed using region graphs. We refer to these approaches as indirect approaches as they are not directly applied to the TA plant model. They consist of four steps; 1) compute the synchronous product of the plant and requirement TA $P^t||R^{p,t}$, 2) "untime" the resulting TA by constructing the REQG, a graph with symbolic states that represent a finite number of regions from the region graph combined with the locations of the TA, 3) apply either DES or TDES synthesis to the resulting FA REQG to find an untimed supervisor $(S^t)^{ut}$, 4) reconstruct the timed supervisor S^t, by adapting the guards and invariants of the input TA $P^t||R^{p,t}$ based on the regions in the REQG $(S^t)^{ut}$. The DES indirect synthesis approach computes a set of bad states that are blocking or can potentially lead to blocking via uncontrollable events or τ jumps. The controllable events that point to these states in the REQG are removed. No τ events can be removed in this approach as they are always considered uncontrollable. In TDES indrect synthesis the set of bad states is restricted to those states for which there exists an uncontrollable time jump τ to an existing bad state. Here, we first remove all controllable guards that are not τ, and if no controllable guards can be removed any more, the τ events are removed. This prevents premature removal of τ events as a result of controllable forcible event removal, as they may point to bad states eventually. In this approach, the removal of τ events effectively removes the bad states from existence for a given location, the location invariant is used to model that these states no longer exist in the reconstruction of the TA supervisor. However, removal of τ events, may result in new bad states, at least, the opposite has not been proven although no examples were found in this study. Therefore in TDES indirect synthesis, if at some point a forcible event is removed from a state that does not belong to the states that initially did not have outgoing τ events, the τ events will be added again, but as uncontrollable events.

The first of the four steps, the synchronous product, has been implemented in the CIF3 tool. Although it does not consider all of the necessary features, such as invariants, it shows that it is possible to model TA in CIF3 and it is possible to use these models as an input for a tool such as the synchronous product. After more in depth research on how to implement the methods that have been proposed in our work, this can prove to be useful, if in the future there is a need to implement a direct synthesis approach for testing use cases. Finally, we conclude that in the timing procedure the most natural choice of permitted guard and invariant grammar is to allow the region expressions as constraints. This is the most permissive solution by allowing the supervisor to enter a location, force it out of the location before an intermediate upperbound is reached and return at a later point in time when it is safe to do so.

However, indirect synthesis approaches require abstractions that are very sensitive to the number of clocks that are used and the largest integer constants that are used for each clock. Therefore, we propose two synthesis approaches that can be applied directly to a TA. For the first approach, Time Uncontrollable Direct Synthesis (TUDS), forcible events are not considered. Therefore only guards may be adapted. The second approach allows the adaptation of invariants as long as a forcible event is present to guarantee preemption of a time delay. We refer to this approach as Time Preemptable Direct synthesis (TPDS). Both are based on an existing superisory synthesis approach for EFA, referred to as SSEFA [3] or data based synthesis in [4] and adapt it to suite TA. TPDS consists of six steps, of which the first four steps describe the TUDS approach but in step four, the alternative of returning to step 1 is that the resulting TA is the supervisor:

1. compute the non-blocking conditions for each location of the plant,
2. compute the bad state conditions,
3. adapt the guards of the supervised plant based on these conditions,
4. if all guards are the same after a subsequent iteration continue to step 5, otherwise return to step 1,
5. adapt the location invariants of the supervised plant, according to the bad state conditions,
6. if the resulting invariants of all locations are the same after a subsequent iteration, this is the resulting supervisor, otherwise return to step 1.

Based on these steps, an algorithm for TUDS is provided. We choose not to include an algorithm for TPDS, as we implicitly assume that if invariants that can be adapted in such a way that forcible events are no longer available for any of the previously adapted invariants, the original invariants of the latter can be reintroduced in one of the above steps. However, we have not yet found a method to do so and use the steps to conceptually explain the method. Note that if it is proven that this problem does not exist, the six steps remain unchanged. The input of the algorithms is the product $P^t||R^{p,t}$ and the output is a TA supervisor S^t, that is controllable with respect to $P^t||R^{p,t}$ and for which the supervised plant $S^t||P^t||R^{p,t}$ is non-blocking and maximally permissive. However, a formal proof is still required to confirm this. To account for the possibility that a valid forcible event is possible in TPDS, the TUDS bad state condition equation is extended such that only the conditions on clock x for which there exists a delay to the bad state, and there does not exist an intermediate controllable delay due to the enablement of a valid forcible event, are considered bad state conditions due to time uncontrollability. It can also be concluded that the problem with using the adaptation of upper bounded invariants as a means of control, is that they do not only restrict conditions for which there exists a bad state. They also make sure that it is no longer possible to return to the adapted location with a clock value that exceeds the invariant. To account for this, we propose a less limited view of invariants, such that any expression that is derived from the bad state conditions can be used. We have also briefly discussed the possibility to implement such a direct synthesis approach. The major challenge in this is to find a way to determine the outcome of the newly added time delay terms.

All in all it can be concluded that firstly, we have found a possible means to apply TDES SCT notions to TA, by using invariants to represent the states that no longer exist due to the availability of one or more enabled forcible events. Secondly, we have applied these notions to an indirect synthesis approach and used them to reconstruct the TA supervisor from the FA supervisor and the input TA. Next to that we introduce two possible direct synthesis approaches one that allows forcible events (TPDS) and one that does not (TUDS) and compare them to the indirect synthesis approaches by means of the well known bus pedestrian example and a theoretic example with bad state intervals. Based on these comparisons we conclude that TPDS and TDES indirect synthesis provide the most permissive solutions, by allowing the TA to enter a location before and after a bad state, instead of only allowing delays past it as in the other approaches. Additionally, we expect that both TUDS and DES indirect synthesis and TPDS and TDES indirect synthesis are equivalent, also due to the fact that the bad state computations are based on the same principles for these approaches. However, formal proof is required to confirm this. The main challenge that remains for direct synthesis in general, is to find a way to determine the outcome of the time delay terms in the non-blocking and bad state condition computations. The main challenge that remains for TPDS, is to prove that to prevent premature adaptation of invariants, it is sufficient to make sure that the guards of the controllable edges no longer change, before adapting the invariants. Equivalently, in TDES that first only removing controllable events that are not τ events until there are none left pointing to bad states. If the opposite is proven, the termination property of the algorithms should be proven. In addition to these challenges, a number of interesting research possibilities still remain.

Even though the algorithms of direct synthesis are based on a simplified version of the EFA synthesis algorithm in CIF3, we have not considered exactly how to implement them. The main challenge is to find a means to implement the $\exists_\Delta$ term, which looks at the time successors with respect to the input conditions. Implementing the approaches would be useful for the purpose of applying them to larger test cases, such that the computational gain can be compared to the indirect approaches. We have only considered simple examples in this work, with at most two clocks, but real problems are much larger. Therefore to automatically determine the outcome of the time delay existence terms, it could be necessary to locally use a form of abstraction, region or zone automaton abstraction. It is expected that even if this is the case, the result will be more efficient than when it is required to transform the entire TA to an FA.

It would also be interesting to research the possibilities of applying synthesis approaches to TA that are extended with the possibility to mark clock values, or discrete variables as in the TEFA in [5]. This would even further extend the expressiveness of TA. In the case of direct synthesis, it would also be possible to consider a broader set of constraints in the input TA.

In future work, it may also be of interest to consider prevention of timelock and Zeno behaviour by means of synthesis. In this work we did not consider these notions as one of the synthesis problems. However, as timelock and Zeno behaviour are a result of bad modelling, it is more appealing to not have to take it into account.

Finally, we have not formally proven the satisfaction of the SCT properties extended to TA. We have assumed that

the approaches work, based on the comparison of results with existing approaches for the well-known Bus Pedestrian example and a search for examples to prove the contrary. Therefore, the task remains to find a formal proof to show that a maximally permissive, non-blocking, controllable supervisor results for all approaches excluding DES indirect synthesis, which is known.

Bibliography

[1] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[2] J. E. Hopcroft, *Introduction to automata theory, languages, and computation*. Pearson Education India, 2008.

[3] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, 2011.

[4] M. Reniers and J. van de Mortel-Fronczak, "Lecture notes supervisory control," March 2018.

[5] S. Miremadi, Z. Fei, K. Åkesson, and B. Lennartson, "Symbolic supervisory control of timed discrete event systems," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 584–597, 2015.

[6] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[7] A. Khoumsi, "Supervisory control of dense real-time discrete-event systems with partial observation," in *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pp. 105–112, IEEE, 2002.

[8] J. S. Ostroff and W. M. Wonham, "A framework for real-time discrete event control," *IEEE Transactions on Automatic control*, vol. 35, no. 4, pp. 386–397, 1990.

[9] A. Dubey, "A discussion on supervisory control theory in real-time discrete event systems," *ISIS*, vol. 9, p. 112, 2009.

[10] J. S. Ostroff, "Deciding properties of timed transition models," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 170–183, 1990.

[11] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[12] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM journal on control and optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[13] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi, "Supervisory control and reactive synthesis: a comparative introduction," *Discrete Event Dynamic Systems*, vol. 27, no. 2, pp. 209–260, 2017.

[14] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[15] P. Gohari and W. M. Wonham, "Reduced supervisors for timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 7, pp. 1187–1198, 2003.

[16] L. Ouedraogo, A. Khoumsi, and M. Nourelfath, "Setexp: a method of transformation of timed automata into finite state automata," *Real-Time Systems*, vol. 46, no. 2, pp. 189–250, 2010.

[17] H. Wong-Toi and G. Hoffmann, "The control of dense real-time discrete event systems," in *Decision and Control, 1991., Proceedings of the 30th IEEE Conference on*, pp. 1527–1528, IEEE, 1991.

[18] A. Khoumsi and M. Nourelfath, "An efficient method for the supervisory control of dense real-time discrete event systems," in *Proc. 8th Intern. Conf. on Real-Time Computing Systems (RTCSA)*, 2002.

[19] L. Ouedraogo, M. N. El Fath, and A. Khoumsi, *Setexp: A method of transformation of timed automata into finite state automata*. CIRRELT, 2008.

[20] O. Maler, A. Pnueli, and J. Sifakis, "On the synthesis of discrete controllers for timed systems," in *Annual Symposium on Theoretical Aspects of Computer Science*, pp. 229–242, Springer, 1995.

[21] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Uppaal-tiga: Time for playing games!," in *International Conference on Computer Aided Verification*, pp. 121–125, Springer, 2007.

[22] W. W. et. al., "Xpttct." [Online]. Available: http://www.control.toronto.edu/people/profs/wonham/wonham.html. Accessed: 21-01-2019.

[23] D. D. Cofer and V. K. Garg, "Supervisory control of real-time discrete-event systems using lattice theory," *IEEE Transactions on Automatic Control*, vol. 41, no. 2, pp. 199–209, 1996.

[24] K. Akesson, M. Fabian, H. Flordal, and R. Malik, "Supremica-an integrated environment for verification, synthesis and simulation of discrete event systems," in *Discrete Event Systems, 2006 8th International Workshop on*, pp. 384–385, IEEE, 2006.

[25] K. Altisen and S. Tripakis, "Tools for controller synthesis of timed systems," in *Proc. 2nd Workshop on Real-Time Tools (RT-TOOLS02)*, pp. 2002–025, 2002.

[26] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, "Controller synthesis for timed automata," *IFAC Proceedings Volumes*, vol. 31, no. 18, pp. 447–452, 1998.

[27] D. A. van Beek, W. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. Van De Mortel-Fronczak, and M. A. Reniers, "Cif 3: Model-based engineering of supervisory controllers," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 575–580, Springer, 2014.

[28] S. Tripakis and S. Yovine, "Analysis of timed systems using time-abstracting bisimulations," *Formal Methods in System Design*, vol. 18, no. 1, pp. 25–68, 2001.

[29] R. Alur, "Timed automata," in *International Conference on Computer Aided Verification*, pp. 8–22, Springer, 1999.

[30] R. Alur, "Timed automata [powerpoint presentation]," September 2004.

[31] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Advanced Course on Petri Nets*, pp. 87–124, Springer, 2003.

[32] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit, "Updatable timed automata," *Theoretical Computer Science*, vol. 321, no. 2-3, pp. 291–345, 2004.

[33] S. Tripakis, "Verifying progress in timed systems," in *International AMAST Workshop on Aspects of Real-Time Systems and Concurrent and Distributed Software*, pp. 299–314, Springer, 1999.

[34] S. Tripakis, "Folk theorems on the determinization and minimization of timed automata," in *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 182–188, Springer, 2003.

[35] R. Kumar, V. Garg, and S. I. Marcus, "On controllability and normality of discrete event dynamical systems," *Systems & Control Letters*, vol. 17, no. 3, pp. 157–168, 1991.

[36] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, 2007.

[37] W. M. Wonham, "Supervisory control of discrete-event systems," *Encyclopedia of systems and control*, pp. 1396–1404, 2015.

[38] R. Gómez and H. Bowman, "Efficient detection of zeno runs in timed automata," in *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 195–210, Springer, 2007.

[39] R. Alur, *Techniques for automatic verification of real-time systems*. PhD thesis, stanford university, 1991.

[40] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Information and computation*, vol. 104, no. 1, pp. 2–34, 1993.

Appendix

A Formal model example - simple light example

For the extended simple light control example (Example 2.1) in Section 2.2, the formal representation of the timed safety automaton is as follows:

- $C = \{x\}$;
- $E = \{press\}$
- $B(C) = \{x > 3, x \leq 3, x \leq 10\}$
- $L = \{OFF, LIGHT, BRIGHT\}$
- $l_0 = \{OFF\}$
- $\rightarrow = \{(OFF, -, press, x := 0, LIGHT), (LIGHT, x > 3, press, -, OFF),$
  $(LIGHT, x \leq 3, press, -, BRIGHT), (BRIGHT, -, press, -, OFF)\}$
- $L_m = \{OFF\}$
- $I(LIGHT) = x =\leq 10$

Note that we only include $I(LIGHT)$ here, as the other locations do not have invariants.

B Region Abstraction after Synchronous Product

Here, we explain in more detail why synchronization of clocks is lost through abstraction.

Example B.1 – Synchronous product before time abstraction:
consider two TA TA_1 and TA_2 and their synchronous product as shown in Figure B.1, each having a unique clock x and y respectively.

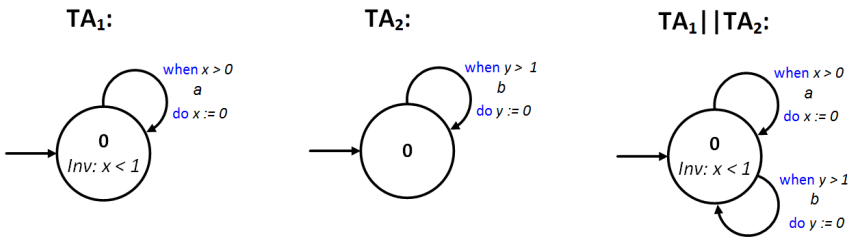


Figure B.1: *Two example TA and their synchronous product*

If we were to abstract TA_1 and TA_2, before applying the synchronous product, a different result follows than when the synchronous product is computed first. In the former case, two possible region graphs for clock x of TA_1 and clock Ty follow, as shown in Figure B.2.

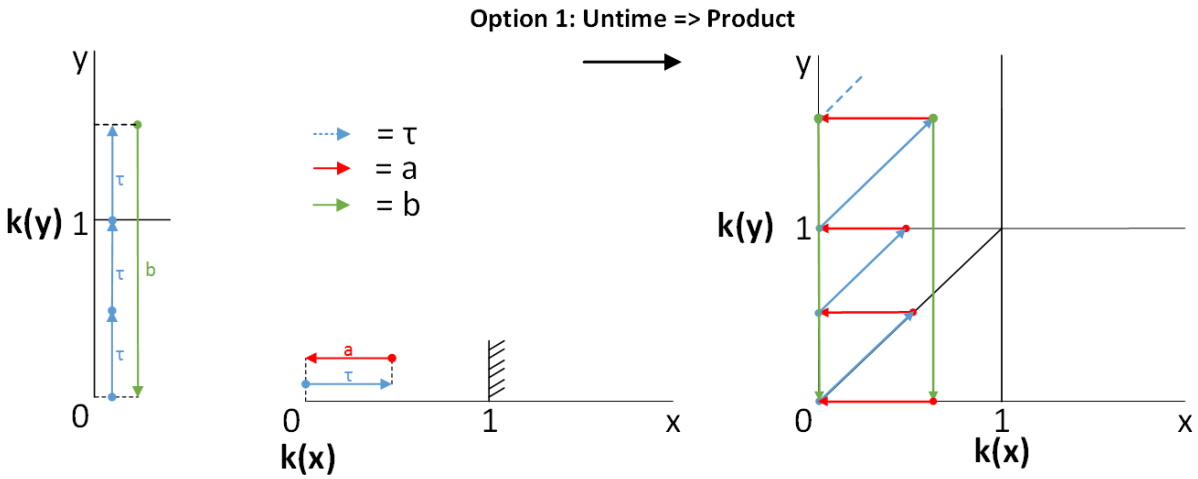


Figure B.2: *Quotient graphs showning why the synchronous product must be computed before time abstraction - option 1: untime before applying synchronous product*

It can be seen that the relative values of x with respect to y are not taken into account due to the abstraction of the exact values and due to the fact that the separate region graphs do not have a distinction between the relative values as they only consider one clock. When composing the two region graphs, it can indeed be seen that the result does not take into account the open regions.

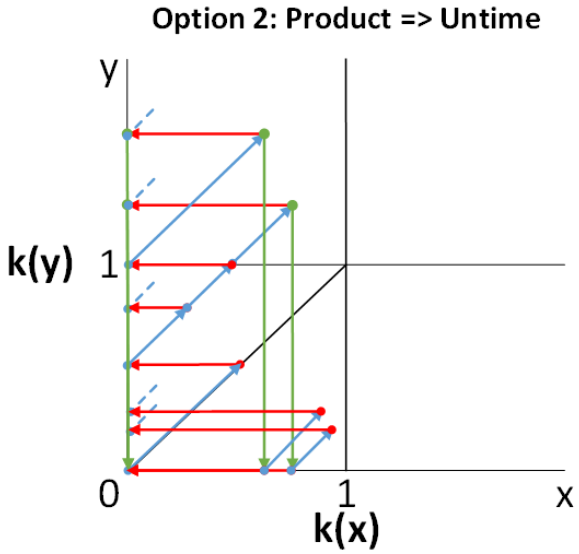


Figure B.3: *Quotient graphs showning why the synchronous product must be computed before time abstraction - option 3: apply synchronous product before untiming*

In Figure B.3, it can be seen that when applying the synchronous product of the TA first, a lot more regions are considered. It becomes clearer from viewing the resulting REQGs as shown in Figure B.4 and B.5.

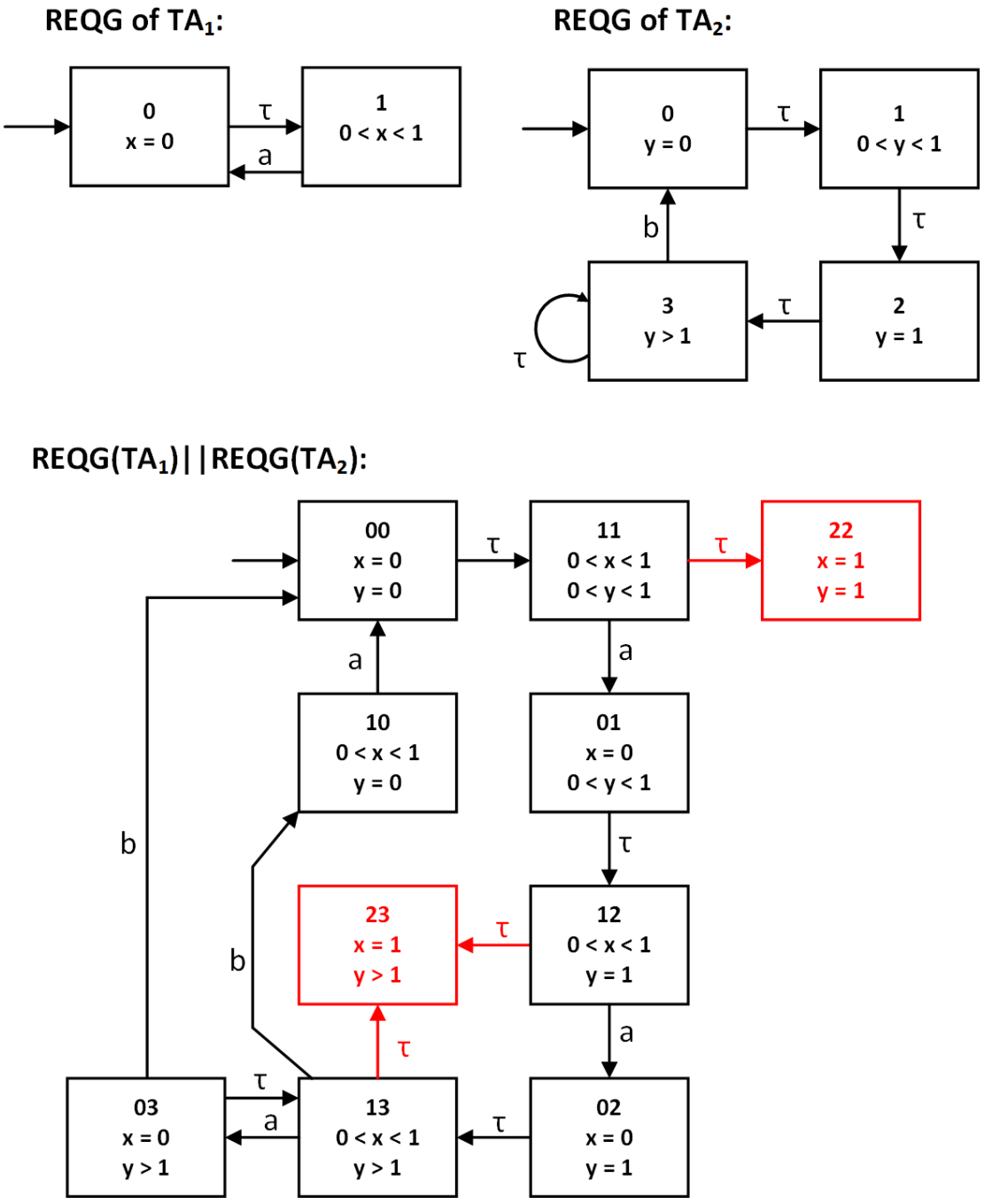


Figure B.4: *REQG that results for option 1: untime -¿ product*

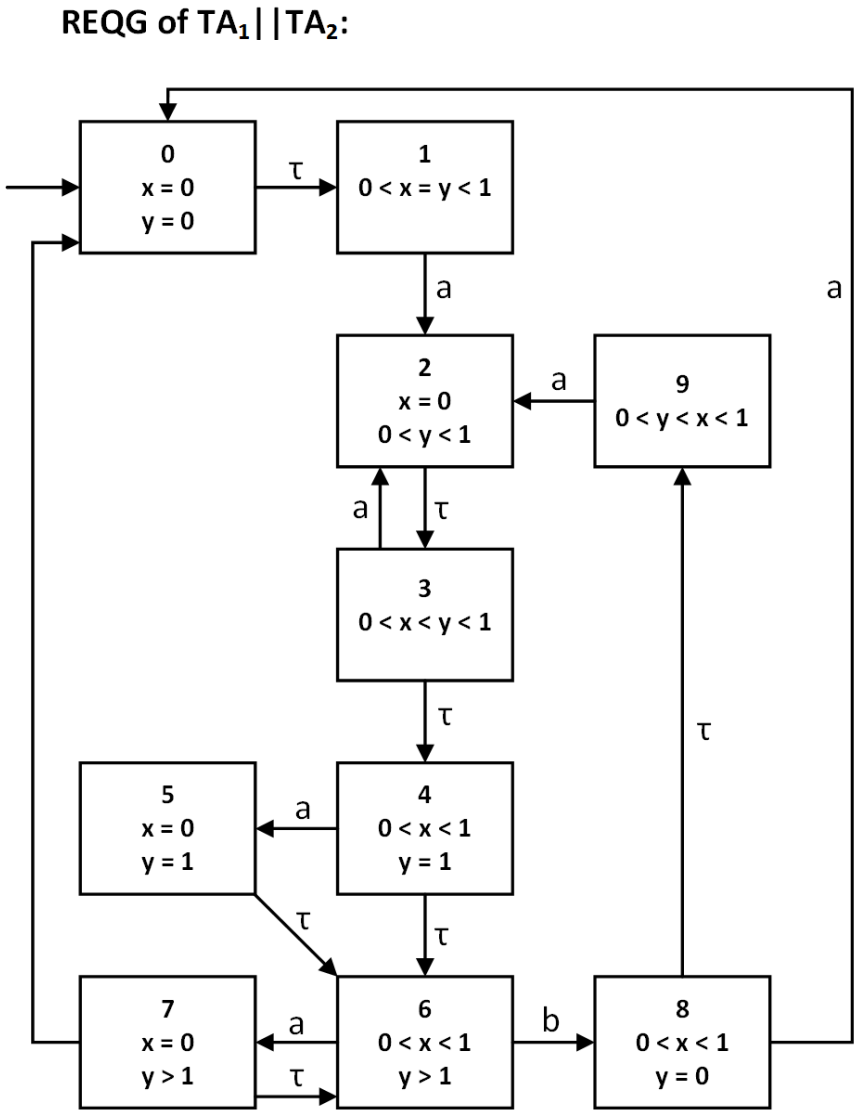


Figure B.5: *REQG that results for option 2: TA product → untime*

It is evident that for the second option, a lot more states are taken into account. More importantly, the REQG in Figure B.4 is not a strong time-abstracting bisimulation of the composed automata.

C I_i versus I_j in Direct Synthesis

If Example 5.2 is worked out with the target location invariant instead of the source location invariant (i.e. $g \wedge N_j^m[r] \wedge I_j[r]$), the resulting non-blocking conditions would be $N_0 = x \leq 1$ and $N_1 = true$. This is also valid, as there is no blocking in location 1 for all possible values of x and the correct conclusion for location 0 results. However, we prefer to use the source location invariant I_i as this also takes into account I_j via the $N_j^m[r]$ term. There are three reasons for this:

1. The result $true$ when using I_j can be misinterpreted as: "it holds for all values of x", when using I_i this is never the case.
2. The guards will be adapted to explicitly show under which conditions the edge exists in the underlying transition system. This is nicer for the graphical examples we show here.
3. The results are consistent, independent of the number of clocks.

In Example C.1, the problem of inconsistency in the non-blocking conditions when using I_j is shown.

Example C.1 – I_j inconsistency: consider the TA shown in Figure C.1.

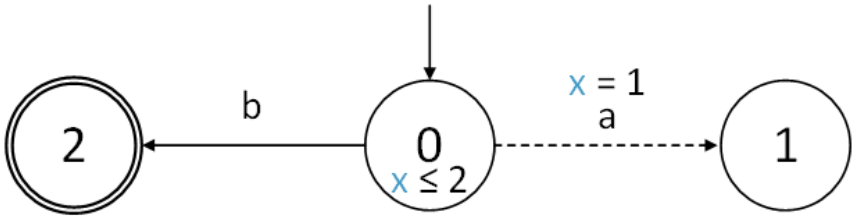


Figure C.1: *Example TA without guard*

The non-blocking condition for N_0 is worked out once using I_i and once I_j in Table C.1. Note that $N_2 = true$ and $N_1 = false$ for both cases.

Table C.1: *Non-blocking condition N_0 worked out for the TA in Figure C.1, once with I_i and once with I_j*

	N	
It	**Loc 0 (with I_i)**	**Loc 0 (with I_j)**
0	f	f
1	$f \vee (x = 1 \wedge f \wedge \boldsymbol{x \leq 2}) \vee$ $(t \wedge t \wedge \boldsymbol{x \leq 2}) \vee$ $\exists_\Delta(x + \Delta \leq 2 \wedge f) =$ $x \leq 2$	$f \vee (x = 1 \wedge f \wedge \boldsymbol{t}) \vee$ $(t \wedge t \wedge \boldsymbol{t}) \vee$ $\exists_\Delta(x + \Delta \leq 2 \wedge f) =$ t
2	$x \leq 2 \vee (x = 1 f \wedge \boldsymbol{x \leq 2}) \vee$ $(t \wedge t \wedge \boldsymbol{x \leq 2}) \vee$ $\exists_\Delta(x + \Delta \leq 2 \wedge x + \Delta \leq 2) =$ $x \leq 2$	t

Effectively, both $x \leq 2$ and t mean the same here. The term $x \leq 2$ represents the possible values in location 0 for which it is non-blocking and t is also valid as nothing needs to change in the TA in terms of non-blocking. The problem with using I_j becomes evident when introducing a guard on the b-labelled transition. This is shown in Figure C.2.

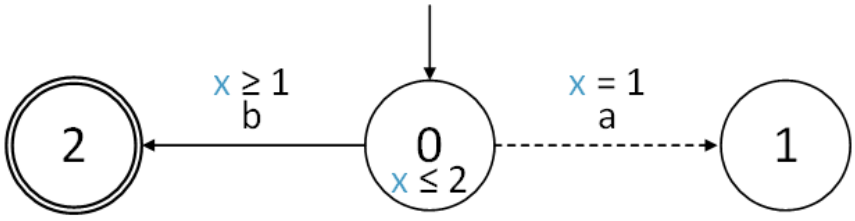


Figure C.2: *Example TA with guard $x \geq 1$*

N_0 is worked out again in Table C.2.

Table C.2: *Non-blocking condition N_0 worked out for the TA in Figure C.2, once with I_i and once with I_j*

	N	
It	**Loc 0 (with I_i)**	**Loc 0 (with I_j)**
0	f	f
1	$f \vee (x = 1 \wedge f \wedge \boldsymbol{x \leq 2}) \vee$ $(x \geq 1 \wedge t \wedge \boldsymbol{x \leq 2}) \vee$ $\exists_\Delta(x + \Delta \leq 2 \wedge f) =$ $x \geq 1 \wedge x \leq 2$	$f \vee (x = 1 \wedge f \wedge \boldsymbol{t}) \vee$ $(x \geq 1 \wedge t \wedge \boldsymbol{t}) \vee$ $\exists_\Delta(x + \Delta \leq 2 \wedge f) =$ $x \geq 1$
2	$x \geq 1 \wedge x \leq 2 \vee (x = 1 \wedge f \wedge \boldsymbol{x \leq 2}) \vee$ $(x \geq 1 \wedge t \wedge \boldsymbol{x \leq 2}) \vee$ $\exists_\Delta(x + \Delta \geq 1 \wedge x + \Delta \leq 2 \wedge x + \Delta \leq 2) =$ $x \leq 2$	$x \geq 1 \vee (x = 1 \wedge f \wedge \boldsymbol{t}) \vee$ $(x \geq 1 \wedge t \wedge \boldsymbol{t}) \vee$ $\exists_\Delta(x + \Delta \geq 1 \wedge x + \Delta \leq 2 \wedge x + \Delta \leq 2) =$ $x \leq 2$

In this case, there is a guard $x \geq 1$ to take into account. Therefore, $N_0 = x \geq 1$ after iteration 1 for both cases. This gives the $\exists_\Delta$ term the change to play a role in the non-blocking condition computation in iteration 2. The result of this, is that even though the guard does not influence blocking, in this case using I_j returns $x \leq 2$ instead of t. This inconsistency is prevented by using I_i instead.

D Alternative Algorithms

In this appendix section, we show alternative synthesis steps to the ones presented in Section 4.2.2 and 5.3 and for TPDS the algorithm that would represent these steps. In these sections we take into account that for TDES a τ and for TPDS an invariant may have been respectively removed and adapted prematurely. However, we are yet to find an example for which this is the case. Therefore we choose to share the original steps, which are more likely to terminate and more intuitive to understand. If a proof is found that forcible events are not influenced in such a way by adaptation of invariants or removal of τ events, that they can no longer be used to preempt time delays, then we propose to use the steps and algorithm that we show here.

D.1 TDES Indirect Synthesis

For TDES indirect synthesis, a five step algorithm was proposed, which added the τ events back to the states for which the outgoing forcible event that allowed this removal is removed at some point during synthesis. If it turns out that the latter is not possible as a result of removing on or more tau events, then it is sufficient to make sure that no controllable events excluding τ events can be removed any more before removing the controllable τ events, as shown in Figure 4.8.

1. compute the set of blocking states, If there are none, the resulting FA is the supervisor, otherwise continue,
2. compute the set of bad states,
3. remove all controllable events that point towards a bad state and that are not τ jump events. If no events can be removed, continue, otherwise remove all unreachable states and the edges between them and return to step 1,
4. remove all controllable τ events that point to a bad state,
5. remove all unreachable states and the edges between them. Return to step 1.

D.2 TPDS

The same holds for TPDS as in TEDS indirect synthesis. Therefore, if a proof is found to show that premature invariant adaptation is not possible as a result of adapting other invariants the following steps for TPDS are proposed:

1. compute the non-blocking conditions for each location of the plant,
2. compute the bad state conditions,
3. adapt the guards of the supervised plant based on these conditions,
4. if all guards are the same after a subsequent iteration continue to step 5, otherwise return to step 1,
5. adapt the location invariants of the supervised plant, according to the bad state conditions,
6. if the resulting invariants of all locations are the same after a subsequent iteration, this is the resulting supervisor, otherwise return to step 1.

Here, the non-blocking conditions are computed in the same fashion as for TUDS, and the bad states are extended with the notion of forcible events. At first, only the guards of $S^t||P^t$ are adapted until they no longer change. This is done to make sure that invariants are not adapted prematurely, e.g. when a controllable forcible event initially points towards a location that becomes blocking at some point, such that it is disabled. When all of the guards remain the same with respect to the previous iteration, the bad state conditions of the locations are used to adapt the location invariants. The following algorithm is proposed in relation to these steps.

Algorithm 2 Direct Supervisory Synthesis for TA (TPDSSTA)

procedure TPDSSTA($P^t||R^{p,t}$)
  Initialize iterators: $o := 0,\ p := 0$
  repeat
    repeat
      Initialize iterators: $m := 0,\ n := 0$
      Initialize non-blocking predicate of every location $l \in L$:

$$N_i^{o,0} = \begin{cases} I_i, & \text{if } l_i \in L_m \\ false, & \text{if } l_i \notin L_m \end{cases} \tag{D.1}$$

      repeat
        Update N_i^o of every location $l_i \in L$:

$$N_i^{o,m+1} := N_i^{o,m} \vee \bigvee_{i \xrightarrow{g^o\ e\ r} j} (g^o \wedge N_j^{o,m}[r] \wedge I_i^p) \vee \exists_\Delta (I_i^p[x+\Delta] \wedge N_i^{o,m}[x+\Delta]) \tag{D.2}$$

        $m := m + 1$
      until $\forall l_i \in L, N_i^{o,m} = N_i^{o,m-1}$
      $\forall l_i \in L : N_i^o := N_i^{o,m}$
      Initialize bad state condition predicate for every location $l \in L$:

$$B_i^{o,0} = \neg N_i^o \tag{D.3}$$

      repeat
        Update B_i^o of every location $l_i \in L$, for all edges $i \xrightarrow{g\ e\ r} j$ with $e \in E_u$ and $i \xrightarrow{g\ e\ r} k$ with $e \in E_f$ and $k \neq i$:

$$\begin{aligned} B_i^{n+1} = B_i^n \vee & \bigvee_{\substack{i \xrightarrow{g\ e\ r} j, \\ e \in E_u}} \big(g \wedge B_j^n[r] \wedge I_j[r]\big) \\ & \vee \Bigg(\exists_\Delta \big(B_i^n[x+\Delta] \wedge I_i^p[x+\Delta]\big) \wedge \forall_{\Delta' \leq \Delta} (B_i^n[x+\Delta'] \\ & \wedge I_i^p[x+\Delta']) \nexists_{\Delta'' < \Delta'} \bigvee_{\substack{i \xrightarrow{g\ e\ r} k, \\ e \in E_f, \\ k \neq i}} \Big(g_f[x+\Delta''] \wedge I_k^p[r][x+\Delta''] \wedge \neg B_k^n[r][x+\Delta''] \Big) \Bigg) \end{aligned} \tag{D.4}$$

        $n := n + 1$
      until $\forall l_i \in L, B_i^{o,n} = B_i^{o-1,n}$
      $\forall l_i \in L : B_i^o := B_i^{o,n}$
      Update guard of every edge in $\rightarrow$ with controllable event $e \in E_c$ via:

$$i \xrightarrow{g^{o+1},\ e,\ r} j := i \xrightarrow{g^o \wedge \neg B_j^o[r]} j. \tag{D.5}$$

      $o := o + 1$
    until For all edges of all locations $l_i \in L, g^o = g^{o-1}$
    Update invariant of every location via:

$$I_i^{p+1} := I_i^p \wedge \neg B_i^o. \tag{D.6}$$

  until $\forall l_i \in L, I^{p+1} = I^p$
end procedure

TU/e Technische Universiteit Eindhoven University of Technology

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[1].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

3-10-2019

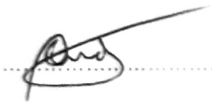
Name

Patrick v.d. Graaf

ID-number

0890738

Signature



Submit the signed declaration to the student administration of your department.

[1] See: http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU

January 15 2016