A Compositional Approach for Verifying Sampled-Data Supervisory Control

A Compositional Approach for Verifying Sampled-Data Supervisory Control

by

MAHVASH BALOCH, M.Sc.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the degree of

Master of Science

Department of Computing and Software

McMaster University

MASTER OF SCIENCE (2012) McMaster University
(Computer Science) Hamilton, Ontario

TITLE: A Compositional Approach for Verifying Sampled-Data Supervisory Control

AUTHOR: Mahvash Baloch, M.Sc. (Peshawar University)
SUPERVISOR: Dr. Ryan Leduc

NUMBER OF PAGES: i, 138

A Compositional Approach for Verifying Sampled-Data Supervisory Control

Abstract

Sampled-data supervisory control deals with timed discrete event systems (TDES) where the supervisors are to be implemented as sampled-data controllers. A sampled-data controller views the system as a series of inputs and outputs and is controlled by a periodic clock. It samples its inputs, changes state, and updates its outputs on each clock edge (the tick event). The sampled-data supervisory control framework provides a set of conditions that the TDES system must satisfy to ensure its correct behaviour in order to be implemented as sampled data controllers. A serious limitation for automatic verification of systems is the size of the system's synchronous product. To overcome this limitation, we propose the use of a compositional approach to the verification of sampled-data supervisory control. In this approach, first we recast the required conditions for sampled-data supervisory control in terms of other properties such as language inclusion, nonblocking or controllability, which already have existing compositional methods and algorithms. This makes the sampled-data properties suitable for compositional verification, considerably increasing the size of systems that can be handled using sampled-data supervisory control. We also develop and implement a set of algorithms for the compositional verification of these sampled-data properties. We provide an example of the SD Controlled Flexible Manufacturing System to test our algorithms.

Acknowledgments

First I would like to thank my supervisor, Dr. Ryan Leduc, for his insight and guidance. I am very grateful to him for his continuous support and sound advice throughout my masters study and research. Without his encouragement I would have never been able to complete this thesis.

I would also like to thank my unofficial co-supervisor, Dr. Robi Malik, for his help and contribution throughout my research and especially in the development of the software as a part of his DES research project, WATERS. I am deeply indebted to him for his time and patience throughout the course of my research. Without his detailed and timely replies to my queries I would not have been able to complete my research. It was a pleasure to work with him.

I am very grateful for the love and support of my parents and my siblings who encouraged me at every step. Thank you for everything.

My deep gratitude goes to my husband, Shahid, for his constant encouragement, support and love. A big thank you to him for his understanding and confidence in me. Finally I would like to thank my sons, Aareel and Romaan, for being a great source of motivation to me. I dedicate this thesis to them.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

A discrete event system (DES) [CL10, Won11] is a dynamic system whose state space is discrete and whose state can only change in response to instantaneous occurence of events over time. The applications of DES can be found in many areas including manufacturing systems, traffic systems, communication protocols, data communication networks and database management systems.

A DES model can be represented as either timed DES (TDES) model [Ost89, Ost90, OW90] or untimed DES model. In an untimed DES model, we are only concerned with the sequence of events taking place in order to satisfy a given set of specifications. Such a model is only concerned in the logical behavior of the system such as finding if a particular state (or set of states) of the system can be reached or not. In such models, the actual timing of events is not considered. In a TDES model, timing information is important as well as the sequence of events . Its important to consider the time at which the system reaches a particular state and the amount of time it spends there. Its also important to be able to specify when an event will occur, and not just the ordering of the event occurences.

The supervisory control theory introduced by Ramadge and Wonham [RW89, Won11, WR87] provides a formal frame work for analyzing discrete event systems. In this theory an automaton is used to model the system(the plant) to be controlled and the specification of the desired system behaviour. The aim is to impose a given specification on the system by means of a supervisor which restricts the behaviour of a system. The theory provides methods and algorithms

to obtain a supervisor that ensures that the system always produces the desired behaviour. It is desirable for a system to satisfy two properties, controllability (undesired actions don't occur) and nonblocking (system doesn't deadlock or livelock).

The supervisory control is often used for complex, safety critical systems, therefore it is very important to verify automatically that the supervisor satisfies the required properties. However, large and complex systems have a very large state space as they are usually modeled as the synchronous product of a group of component DES, thus the state space size increases exponentially in the number of the components. Such a problem is known as the state space explosion problem. This makes the verification of complex systems more challenging. To address the state space explosion problem, different approaches to system verification have been proposed and investigated.

Symbolic verification [McM93] is one method of verifying complex finite state safety critical systems. It is possible to verify extremely large systems having as many as 10^{20} states by using Binary decision diagrams (BDD) [Bry92]. Several systems of industrial complexity have been verified[CCMM95, HGCC96, HGKCL95] using this technique. In spite of being quite successful, symbolic model checking has its limitations. In some cases it can easily encounter the state space explosion problem.

Another technique which has been successful to overcome state space explosion is compositional verification [CLM89, GL94, KM00]. The compositional verification method splits up the verification of a system into the verification of its components. That means we can verify that a system satisfies the properties of interest by checking its components one by one, rather than checking the whole system. Separate verification of components limits state explosion.

Once we have designed our system, we next need to physically implement our

supervisors, which brings us to the focus of this thesis. The sampled-data (SD) supervisory control framework was proposed in [Wan09, LW10] to implement TDES supervisors as sampled-data controllers. An SD controller is controlled by a periodic clock and views the system as a series of inputs and outputs. The sampled-data supervisory control provides a set of conditions that can be used to ensure correct behaviour (i.e. our controller retains nonblocking and controllability) when implementing TDES supervisors as SD controllers. The algorithms in [Wan09] for verification of these properties have been implemented using symbolic methods.

In this thesis, we propose a compositional approach for verifying the properties of sampled-data supervisory control in order to increase the size of systems to which this theory can be applied. Our approach to develop a compositional verification method for sample-data supervisory control is to first redefine the required property for the given system in terms of either controllability, nonblocking or a language inclusion problem, and then show that this new representation is equivalent. We can then use the existing compositional approaches for verifying controllability, nonblocking, and language inclusion to check these conditions.

1.1 Related Work

The implementation of a controller in supervisory control has always been a challenging problem. A lot of research [BHG+93, Bal94, GR87] has been done to establish the fact that in order to implement a controller, its behaviour needs to satisfy additional properties not included in the traditional supervisory control theory. In [Mal03] the author suggests conditions to be satisfied for implementing nonblocking controllers for untimed nonblocking supervisors.

Sampled-data supervisory control theory [LW10, Wan09] establishes a set of conditions that need to be satisfied in order to implement TDES supervisors as

SD controllers. This theory is developed using the theory of supervisory control of timed DES [BW94, Bra92] as a basis.

Verification and synthesis of discrete event systems has been an interesting area for researchers in the recent years. [ÅFF02] takes advantage of the modular structure of the system to develop an efficient algorithm for verifying controllability. In [BMM04], the authors present a new approach for the verification of safety properties, controllability and language inclusion, based on subsystem verification. They combine the use of counter-examples and heuristics to identify subsystems incrementally.

The verification of nonblocking has been a more challenging problem. A lot of effort has gone into the study of compositional semantics of nonblocking [KS94, RMR06] and its verification [FM09, PCL06]. In [FM09], the authors propose a compositional approach to verify nonblocking for a large discrete event system. The method avoids computing the synchronous product directly, but instead the synchronous product is computed gradually, and the results are simplified using conflict-preserving abstractions based on process-algebraic results about fair testing. The authors propose using the the same method to verify controllability.

In [ML08] authors Malik and Leduc discuss the conversion of nonblocking property into a more expressive property called generalised nonblocking and present its compositional verification method in [ML09]. Its implementation is presented in [Fra11]. In a similar manner, Malik and Leduc provide a compositional approach for verifying HISC [Led02, LBLW05, LDS09, LLD06, LLW05, Led09] which is presented in [LM10].

An efficient algorithm which uses the compositional technique for the detection of control loops is given in [MM06]. The technique takes advantage of the modular structure of discrete event system models to identify and check subsys-

tems, in such a way that the results of checking the subsystems can be used to draw conclusions about the properties of the entire system.

The use of language projection can greatly improve the performance of compostional verification to verify properties of a large discrete event system. In [WM08], the authors propose the use of abstraction by language projection to prove or disprove that a large system of composed finite-state machines satisfies a given safety property.

1.2 Structure of the Thesis

In this thesis we present a compositional approach for verifying sampled-data supervisory control systems. In Chapter 2, we present the definitions and notations that are used throughout the thesis. Chapter 3 is an introduction to sampled-data systems and the required conditions for sampled-data supervisory control. It also gives examples for the properties to illustrate their meaning.

In Chapters 4 and 5, we provide the new definitions for the SD properties. We express each property in terms of a language inclusion, nonblocking or controllability property and provide detailed proofs to show that the new definitions are equivalent to the ones originally given in [Wan09, LW10].

In Chapter 6, we present our algorithms to verify the SD properties and discuss the implementation of the algorithms. In Chapter 7, we present the SD controlled FMS example from [Wan09] and discuss the results of applying our software to it. Finally we conclude with conclusions and future work in Chapter 8.

CHAPTER 2

Preliminaries

A discrete event system is a dynamic system whose state space is discrete and whose state can only change in response to instantaneous occurence of events over time. DES are modelled as automata that generate a formal language of distinct symbols. Supervisory control theory [RW89, Won11, WR87] provides a theoretical framework for the control of discrete event systems. This chapter presents an overview of the the terminology and concepts used in this thesis.

2.1 Languages

Let alphabet Σ be a non-empty finite set of distinct symbols. A *string* is a finite sequence of symbols which belong to Σ. The set Σ^* defines the set of all finite sequence of events of the form $\sigma_1\sigma_2...\sigma_n$ $(n \geq 1)$ as well as the *empty string*, ϵ. Given a string s, $|s|$ = n defines the *length* of the string. The empty string ϵ has length 0. Let Σ^+ denote the set of all finite non-empty symbol sequences. We can then define $\Sigma^* := \Sigma^+ \cup \{\epsilon\}$. The *catenation* operation of strings is defined as

$$\text{cat} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

where cat(ϵ , s) = cat(s, ϵ) = s, and cat(s,t) = st.

Let s, t $\in \Sigma^*$. String t is said to be a *prefix* of s, denoted as $t \leq s$, if ($\exists$u $\in \Sigma^*$)$su = t$. Clearly ϵ is a prefix of all strings, as ($\forall$s $\in \Sigma^*$)$\epsilon \leq$ s. Also, every string s $\in \Sigma^*$ is a prefix of itself, as s $\leq$ s.

A language over Σ is any subset L $\subseteq \Sigma^*$. The *prefix closure* of L, written as

$\overline{L}$, is defined as $\overline{L} = \{s \in \Sigma^* | (\exists t \in L) s \leq t\}$. Clearly, a language is a subset of its prefix closure, i.e. L $\subseteq \overline{L}$. A language is called *prefix closed* if $L = \overline{L}$.

The usual set operations, such as union, intersection, difference etc., are applicable to languages since languages are sets. For sets Σ and Σ_1, the notation $\Sigma - \Sigma_1$, or $\Sigma \setminus \Sigma_1$ refers to the set $\{x | x \in \Sigma \wedge x \notin \Sigma_1\}$, the set of elements of Σ that are not in Σ_1. Similarly Pwr(Σ) refers to the set of all subsets of Σ, including $\emptyset$, the empty set.

Let $L \subseteq \Sigma^*$. The *eligibility operator*, $\text{Elig}_L : \Sigma^* \rightarrow \text{Pwr}(\Sigma)$, is defined for s $\in \Sigma^*$ as Elig_L(s) := $\{\sigma \in \Sigma \mid s\sigma \in L\}$.

We define the *Nerode equivalence relation* on Σ^* with respect to L (or mod L) as follows. For strings s, t $\in \Sigma^*$,

$$\text{s} \equiv_L \text{t or s} \equiv \text{t (mod L) iff } (\forall \text{ u} \in \Sigma^*) \text{ su} \in \text{L iff tu} \in \text{L.}$$

The above definition states that we say s and t are Nerode equivalent with respect to the language L if and only if they can be extended by any string u $\in \Sigma^*$ such that either both strings are in L or neither string is in L. In other words, if strings s and t are equivalent *mod* L then both s and t can be continued in exactly the same way by right concatenation with respect to membership in L.

2.2 Discrete Event Systems

2.2.1 Generator

Formally, we model DES as a generator **G**, which is a tuple

$$\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$$

where Q is the state set, Σ is the event set, δ : Q $\times \Sigma \rightarrow$ Q is the (partial) transition function, q_0 is the initial state, and $Q_m \subseteq$ Q is the set of marked states.

As $\delta : Q \times \Sigma \rightarrow$ Q is a partial function, $\delta(q, \sigma)!$ means $\delta(q, \sigma)$ is defined for

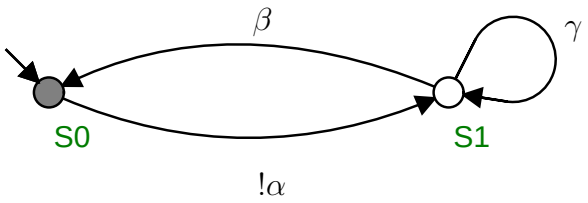


Figure 2.1: An Example DES

$\sigma \in \Sigma$ at state q $\in$ Q. We can extend the transition function to $\delta : Q \times \Sigma^* \rightarrow$ Q as

$$\delta(q, \epsilon) = q \text{ for q} \in \text{Q}.$$
$$\delta(q, s\sigma) = \delta(\delta(q, s), \sigma) \text{ for } s \in \Sigma^* \text{ , } \sigma \in \Sigma, \text{ and q} \in \text{Q},$$

provided $q' := \delta(q, s)!$ and $\delta(q', \sigma)!$.

A DES **G** can be represented by a transition graph. The states in Q are represented by the nodes(circles) in the graph. Transitions are represented by the edges. The small arrow pointing to a state labels the initial state. and the marked states are represented by a shaded circle. Usually, a controllable event is graphically shown by a slash on the transition edge. For the figures in this thesis, we have prefixed uncontrollable events with a "!" to distinguish between controllable and uncontrollable events.

For convenience, we use the notation Σ_G to define the event set that DES G is defined over. Similarly we will use the notations $Q_G, \delta_G, q_{0,G}$ and $Q_{m,G}$ for the other components of DES **G**.

The closed behaviour of DES **G** is described by the language

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

The marked behaviour of DES **G** is defined as

$$L_m(G) = \{s \in \Sigma^* \mid \delta(q_0, s)! \wedge \delta(q_0, s) \in Q_m\}$$

Note that L_m(G) $\subseteq$ L(G) and $\epsilon \in$ L(G) as long as $Q \neq \emptyset$.

A state q $\in$ Q is reachable if there is a string s $\in \Sigma^*$ with $\delta(q_0, s)!$ and

$\delta(q_0, s) = q$. **G** itself is reachable if q is reachable for all q ∈ Q .

A state q ∈ Q is coreachable if there is $s \in \Sigma^*$ such that $\delta(q, s) \in Q_m$. **G** is coreachable if q is coreachable for every q ∈ Q.

We define *equivalence relation* λ on Q as:

$(\forall q, q' \in Q) q \equiv q' \pmod{\lambda}$ iff

i) $(\forall s \in \Sigma^*)\ \eta(q, s)! \Leftrightarrow \eta(q', s)!$

ii) $(\forall s \in \Sigma^*)\ \eta(q, s)! \wedge \eta(q, s) \in Q_m \Leftrightarrow \eta(q's)! \wedge \eta(q', s) \in Q_m$

Nonblocking [Won11] is a very crucial property for discrete event systems. It is the ability to reach a marked state from all reachable states. Formally defined, **G** is said to be nonblocking if every reachable state is coreachable. The language form of the definition is:

$$\overline{L_m(G)} = \text{L(G)}$$

We say that a set of DES is nonconflicting, if the synchronous product (see definition in next section) is nonblocking.

2.2.2 Synchronous Product

In real life, it is more convenient to model a system using more than one DES that interact with each other when they run concurrently. To model a system, we need a way to represent the entire system running in parallel. The synchronous product [Won11] is used to combine two or more DES into a single one. Each DES in the system usually shares some events in their alphabet with other DES in the system. These shared events must be executed by all components at the same time.

To define the synchronous product, we first need to define the natural projection operator and its inverse. Let $\Sigma_i^* \subseteq \Sigma$, i=1,2. We define the *natural projection* $P_i : \Sigma^* \to \Sigma_i^*$ and $\Sigma = \Sigma_1 \cup \Sigma_2$ as:

$$P_i(\epsilon) = \epsilon$$

$$P_i(\sigma) = \begin{cases} \epsilon & \text{if } \sigma \notin \Sigma_i \\ \sigma & \text{if } \sigma \in \Sigma_i \end{cases}$$

$$P_i(s\sigma) = P_i(s)P_i(\sigma) \qquad\qquad s \in \Sigma^*,\ \sigma \in \Sigma$$

$P_i(s)$ removes all events that belong to $\Sigma - \Sigma_i$, for string s $\in \Sigma^*$. The *inverse image* of P_i is $P_i^{-1} : \text{Pwr}(\Sigma_i^*) \to \text{Pwr}(\Sigma^*)$. For L $\subseteq \Sigma_i^*$, we get

$$P_i^{-1}(L) := \{s \in \Sigma^* | P_i(s) \in L\}.$$

Let $L_i \subseteq \Sigma_i^*$. The synchronous product of L_1 and L_2 is defined as

$$L_1 \parallel L_2 = P_1^{-1}L_1 \cap P_2^{-1}L_2$$

For DES $G_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, Q_{m,1})$ and DES $G_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, Q_{m,2})$ defined over the same event set Σ, the *product* of two DES is defined as

$$G_1 \times G_2 = (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_{0,1}, q_{0,2}), Q_{m,1} \times Q_{m,2})$$

where $\delta_1 \times \delta_2 : Q_1 \times Q_2 \times \Sigma \to Q_1 \times Q_2$ is defined as

$$(\delta_1 \times \delta_2)((q_1, q_2), \sigma) := (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$$

whenever $\delta_1(q_1, \sigma)!$ and $\delta_2(q_2, \sigma)!$, otherwise undefined.

By this definition, we have

$L(G_1 \times G_2) = L(G_1) \cap L(G_2)$ and $L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2)$

For DES $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$ and DES $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ the synchronous product G $= G_1 \parallel G_2$ of the two DES is defined as :

$$\text{G} := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{0,1}, q_{0,2}), Q_{m,1} \times Q_{m,2}),$$

where $\delta((q_1, q_2), \sigma)$ is only defined and equals

(q_1', q_2') if $\sigma \in (\Sigma_1 \cap \Sigma_2), \delta_1(q_1, \sigma) = q_1', \delta_2(q_2, \sigma) = q_2'$ or

(q_1', q_2) if $\sigma \in \Sigma_1 - \Sigma_2, \delta_1(q_1, \sigma) = q_1'$ or

(q_1, q_2') if $\sigma \in \Sigma_2 - \Sigma_1, \delta_2(q_2, \sigma) = q_2'$.

When we synchronize two DES, we get a new DES with the synchronized langauges as its language i.e.

$L(G) = L(G_1) \parallel L(G_2)$, $L_m(G) = L_m(G_1) \parallel L_m(G_2)$

Let G_1 be a DES with event set Σ_1 and Σ_2 be another event set such that $\Sigma_1 \cap \Sigma_2 = \emptyset$. We define the selfloop operation on G_1 as **selfloop(G_1,Σ_2)** = $(Q_1, \Sigma_1 \cup \Sigma_2, \delta_2, q_{0,1}, Q_{m,1})$,
where $\delta_2 : Q_1 \times (\Sigma_1 \cup \Sigma_2) \rightarrow Q_1$ is a partial function defined as:

$$\delta_2(q,\sigma) := \begin{cases} \delta_1(q,\sigma) & \sigma \in \Sigma_1, \delta_1(q,\sigma)! \\ q & \sigma \in \Sigma_2 \\ undefined & \text{otherwise} \end{cases}$$

In this thesis, when using the synchronous product to combine individual DES G_i (i=1,2) we will typically assume that first the DES is extended to be over the alphabet Σ by adding selfloops, and then combined using the product operator. Therefore $G_1 \parallel G_2 = \text{selfloop}(G_1, \Sigma - \Sigma_1) \times \text{selfloop}(G_2, \Sigma - \Sigma_2)$.

2.2.3 Controllability and Supervision

The notion of supervisory control [Won11] is introduced by assuming that some events can be disabled i.e. not allowed to occur by an external agent (also known as a supervisor). The alphabet Σ is partitioned into two disjoint subsets Σ_c and Σ_u, where Σ_c is the set of controllable events (which can be enabled or disabled) and Σ_u is the set of uncontrollable events (which cannot be disabled). Once we reach a state in the DES where an uncontrollable event can occur, the event cannot be prevented.

Let the DES $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ represent the possible system behaviour (i.e. our plant) and let K $\subseteq \Sigma^*$ be a language describing the desired safe behaviour of our system. We say K is *controllable* with respect to **G** if

$$\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}.$$

The above definition says that K is controllable with respect to L(G) if every string in $\overline{K}\cap$ L(G) followed by an uncontrollable event σ that is possible in L(G),

then σ must be accepted in $\overline{K}$ as well. We can also express this definition as

$$(\forall s \in \overline{K} \cap L(G))Elig_{L(G)}(s) \cap \Sigma_u \subseteq Elig_{\overline{K}}(s)$$

Note that the languages $\emptyset$, L, and Σ^* are all controllable with respect to G.

To represent our desired system behaviour, we define a DES $\mathbf{S} = (X, \Sigma_S, \xi, x_0, X_m)$ called a supervisor. Supervisors monitor the events generated by the plant, and disable events according to some control law. We say supervisor **S** is controllable for plant **G** if

$$\text{L(S) } \Sigma_u \cap \text{ L(G)} \subseteq \text{L(S)}$$

2.2.4 Language Inclusion

Language inclusion is used to check if the language of a system is included by the language of a given DES. Let DES **G** represent our system and DES **R** define the requirements. Language inclusion is defined as follows:

Definition 2.1. Let **G** and **R** be two DES using the same alphabet Σ. We say **G** *satisfies* **R** if L(G) $\subseteq$ L(R).

In other words **G** satisfies **R**, if and only if every string in L(G) is also in L(R). Language inclusion is also used to check for language equality. For two languages L_1 and L_2, $L_1 = L_2$ can be expressed as $L_1 \subseteq L_2$ and $L_2 \subseteq L_1$.

2.3 Timed Discrete Event Systems

The timed DES (TDES) theory introduced by Brandin et al [BW94, Bra92], adds onto untimed DES theory. A special event *tick* is added to the event set which models the passage of time representing the tick of a global clock. The event set Σ, for a TDES, is expressed as the disjoint union of non-tick events called activity events (Σ_{act}) and the *tick* event, i.e. $\Sigma = \Sigma_{act} \dot{\cup} \{tick\}$. Additional

control method is introduced by means of forcible events (Σ_{for}). A forcible event is defined as an event which is guaranteed to occur before the next *tick* thus preempting the *tick*. It models the situation in which something is forced to occur within a certain time frame. Hence it is possible not only to prevent some events (prohibitable events $\Sigma_{hib} \subseteq \Sigma_{act}$) from occurring by disabling them, but also to force some events to occur before the next clock tick.

We formally define a TDES as the tuple $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ where, Q is the state set, $\Sigma = \Sigma_{act} \dot{\cup} \{tick\}$ is the event set, $\delta : \mathrm{Q} \times \Sigma \rightarrow \mathrm{Q}$ is the (partial) transition function, $q_0 \in \mathrm{Q}$ is the initial state and $Q_m \subseteq Q$ is the set of marked states. For convenience, we extend δ to function $\delta : \mathrm{Q} \times \Sigma^* \rightarrow \mathrm{Q}$ in the same way as we did in the untimed DES definition.

2.3.1 Controllability and Supervision

Same as untimed DES, control for timed DES is described by disabling controllable events. The event set Σ is divided into controllable and uncontrollable events. We define the set of controllable events as $\Sigma_c := \Sigma_{hib} \cup \{tick\}$ where $\Sigma_{hib} \subseteq \Sigma_{act}$ is the set of activity events that can disabled by supervisors. Clearly, the difference between controllable events and prohibitable events is that the prohibitable events do not contain the *tick* event, as a *tick* event is disabled in the system to model the forcing of an event. We define the set of uncontrollable events for $\mathbf{G}$ as $\Sigma_u := \Sigma - \Sigma_c$

Controllability for timed DES is defined as follows:

Definition 2.2. A language $\mathrm{K} \subseteq \mathrm{L(G)}$ is said to be *controllable* with respect to G if,

$$(\forall s \in \overline{K}) Elig_{\overline{K}}(s) \supseteq \begin{cases} Elig_{L(G)}(s) \cap (\Sigma_u \cup \{tick\}) & \text{if } Elig_{\overline{K}}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(G)}(s) \cap \Sigma_u & \text{if } Elig_{\overline{K}}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

The above definition says *tick* can only be disabled when there exists an eligible forcible event to preempt the *tick*.

Definition 2.3. Supervisor $\mathbf{S} = (X, \Sigma, \xi, x_0, X_m)$ is controllable with respect to $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ if for all s $\in$ L(S) $\cap$ L(G),

$$Elig_{L(S)}(s) \supseteq \begin{cases} Elig_{L(G)}(s) \cap (\Sigma_u \cup \{tick\}) & \text{if } Elig_{L(S)\cap L(G)}(s) \cap \Sigma_{for} = \emptyset \\ Elig_{L(G)}(s) \cap \Sigma_u & \text{if } Elig_{L(S)\cap L(G)}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Note that the closed and marked behaviour of a TDES is defined in the same way as for an untimed DES. The definitions for nonblocking and language inclusion remains the same as for untimed DES.

CHAPTER 3

Sampled-Data Supervisory Control

In this chapter we present an overview of sampled-data supervisory control. Sampled-data supervisory control deals with the implementation of TDES supervisors as SD controllers. In order to translate a TDES supervisor to an SD controller such that it exhibits correct behaviour, we have to verify some additional properties about our system. The basic definitions and notations are presented briefly but not in detail. The details can be found in [Wan09, LW10]. We briefly introduce the sampled-data setting for TDES and the properties required to implement TDES systems as SD controllers. Here we are only concerned about effeciently verifying these properties. For details about how to convert a TDES supervisor to an SD controller, see [Wan09].

3.1 SD Controller

When using an SD controller to control a system, an input is associated with each event, and an output is associated with each controllable event. An event is considered to have occured when its corresponding input becomes true during a given clock period. A controllable event is considered to be enabled when its output is set to true by the controller, otherwise it is considered to be disabled. Finally, we associate the clock edge that drives the SD controller with the TDES *tick* event.

An SD controller samples the value of its inputs on each clock edge, and then

uses the information to decide what its next state is going to be. This means the SD controller only knows about its inputs at each clock edge, and all it knows is whether a given input is true or false. This only tells the controller that the event has occurred during the clock period that just ended. This means that an SD controller has no information about event ordering; i.e which event occurred first or last, as well as how many times the event occurred if it occurred more than once. The only ordering information that remains is which clock period did the event occur in. Also, an SD controller only changes state on a clock edge, meaning its outputs can only change at a clock edge, and then stay constant for the rest of the clock period. This means enablement and forcing decisions are determined right after the clock edge and cannot change until the next clock edge.

In Figure 3.1, we see on the third rising edge of the clock, the SD controller

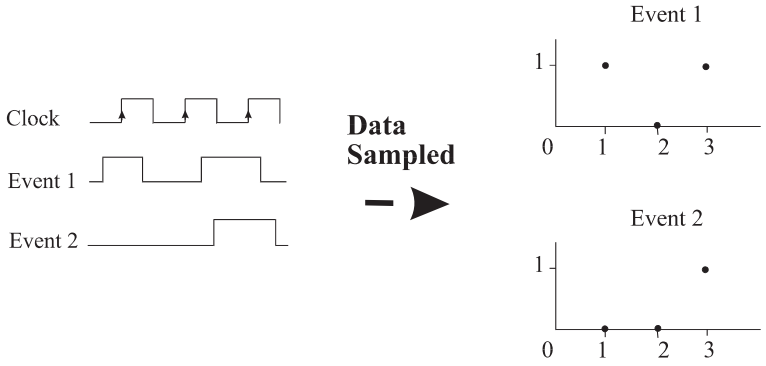


Figure 3.1: Sampling Events

knows that both events e1 and e2 have occurred, but not which one occured first. This means that the SD controller cannot tell the difference between the strings *Event1-Event2-tick*, *Event2-Event1-tick*, or *Event1-Event2-Event1-tick*.

In TDES, the general assumptions are that events occur in a particular order, we know immediately when events occur and that there is no communication delay; i.e. enablement and forcing occurs immediately. These assumptions are not

true in general for SD controllers, thus giving rise to concurrency and timing issues when implementing TDES as SD controllers. These issues bring up concerns with respect to controllability, nonblocking, plant model correctness, and the SD controller's ability to determine which state the TDES system currently is in. In [Wan09, LW10], the authors identify a number of existing TDES properties as well as introduce some new properties, in particular SD controllability, to address these issues and to make it easier to translate a TDES supervisor to an SD controller.

3.2 System Assumptions

To implement sampled-data controllers, the following assumptions are made about the system. It is up to the designer of the system to ensure that the system implements the following assumptions.

1) The set of prohibitable events is equal to the set of forcible events
($\Sigma_{hib} = \Sigma_{for}$).

2) Enabling a forcible event means the event is forced to occur once in that clock period.

3) All SD controllers are implemented centrally with a common clock, such that all inputs are sampled at the same time and all outputs updated at the same time.

4) An event is assumed to have occured as soon as its input to the controller goes true. If the input goes true very close to a clock edge, it is assumed to occur immediately in the next sampling period.

5) When an event is forced to occur in a sampling period, it should occur only during that clock period and not the next one.

6) The length of an input signal should always be appropriate, so as to not be missed by SD controllers (too short) or not to be intrepreted to occur in multiple

clock periods (too long).

Applicable systems should not find Assumptions 1, 2, 5, and 6 very restrictive. Assumptions 3 and 4 partially address timing delay issues and are likely to be removed in future work.

3.3 SD Preliminaries

An SD controller can only observe ϵ and strings ending with a *tick*. These strings are called *sampled strings* and are defined as:

$$L_{samp} = \Sigma^*.tick \cup \{\epsilon\}$$

where Σ is the system's event set.

Concurrent strings are all the strings possible in the system since the last sampled string and which end with a *tick* event. These strings determine the next state of an SD controller as the SD controller only changes state at each clock edge (the occurence of *tick* event). They are defined as

$$L_{conc} = \Sigma^*_{act}.tick \subset L_{samp}$$

However, if two concurrent strings contain the same events but in different order and/or number, an SD controller would not be able to distinguish between the strings. To express this uncertainty we define the *occurrence operator*. The occurrence operator takes a string and returns the *occurrence image* of the string (the set of events that make up the string). For s $\in \Sigma^*$, the occurrence operator, Occu: $\Sigma^* \rightarrow Pwr(\Sigma)$ is defined as:

$$Occu(s) = \{\sigma \in \Sigma \mid s \in \Sigma^*\sigma\Sigma^*\}$$

For a TDES S $= (X, \Sigma, \xi, x_0, X_m)$, *sampled states* are those states that are reached by sampled strings; thus they are partially observable. They are defined as:

$$X_{samp} = \{x \in X \mid (\exists s \in L(S) \cap L_{samp})x = \xi(x_0, s)\}$$

Essentially, a controller starts at the initial state x_0 and changes from one sampled state to the next as concurrent strings occur.

3.4 SD Properties

Described below are the conditions for TDES which are essential to implement sampled-data supervisory control.

3.4.1 Plant Completeness

The concept of plant completeness was introduced by Balemi in [Bal94]. Balemi states that controllable events often are part of the supervisors implementation and can occur when we want them to occur. However, to make the system easier to model and understand, a plant can be modeled more restrictively.

Definition 3.1. A plant G is *complete* for for its supervisor S if

$$(\forall s \in L(G) \cap L(S))(\forall \sigma \in \Sigma_c)s\sigma \in L(S) \Longrightarrow s\sigma \in L(G)$$

The above definition is given in terms of controllable events which includes the *tick* event. As timed DES is just concerned with the occurence of activity events, plant completeness has been adapted in [Wan09], to only apply to activity events and is defined below.

Definition 3.2. Let TDES G be a plant and TDES S be a supervisor. G is *complete* for S if

$$(\forall s \in L(G) \cap L(S))(\forall \sigma \in \Sigma_{hib})s\sigma \in L(S) \Longrightarrow s\sigma \in L(G)$$

The above definition states that for every state in the plant **G**, if a prohibitable event is enabled by the supervisor **S**, it must be possible in the plant. If compared to the definition of a supervisor **S** being controllable with respect to plant **G**, we can see that it's the same if we switch the supervisor **S** with plant **G** and replace controllable events with uncontrollable events.

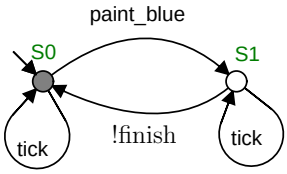


Figure 3.2: Plant

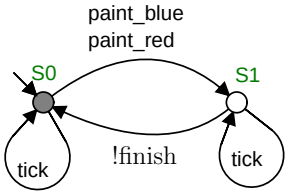


Figure 3.3: Supervisor

Figure 3.2 and 3.3 show a plant and a supervisor in which the plant is not complete for the supervisor. At state **S0** in the plant, event *paint_red* is not eligible, while its eligible at state **S0** in the supervisor.

3.4.2 Activity Loop Free

To ensure that a TDES does not allow a *tick* event to be indefinitely preempted by activity events, it is required that the TDES be activity loop free (ALF) [BW94]. In a TDES, activity events ($\Sigma_{act} \subseteq \Sigma$) are non-tick events which show a sequence of events occuring over a period of time. The occurence of a *tick* event represents the notion of time passing, allowing us to specify time bounds for the occurence of activity events. Therefore, it is desired that the sequence of activity events terminate after a finite number of steps, ensuring that our TDES does not model the unrealistic situation where an activity event loop could be executed continuously by indefinitely preempting a *tick* event.

Definition 3.3. A TDES $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ is *activity loop free* if

$$(\forall q \in Q_{reach})(\forall s \in \Sigma^+_{act})\delta(q, s) \neq q$$

where $Q_{reach} \subseteq Q$ is the set of all reachable states of $\mathbf{G}$.

The plant in Figure 3.2 fails the ALF condition since at state **S1**, the loop $paint\_blue - finish$ can occur an infinite number of times with no *tick* event.

The ALF condition is only applied to the system's closed-loop behaviour. The supervisors are not required to be ALF as they may contain self-loops that are not possible in the plant. However, selfloops essentially provide enablement information but do not have any useful next-state information for SD controllers, so they can be ignored for translation purposes. The definition given below makes supervisors much easier to translate. It states that if selfloops of any activity events are removed from the TDES, the rest of the TDES must be ALF.

Definition 3.4. Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ be a TDES, and let G' be $\mathbf{G}$ with all activity event selfloops removed. $\mathbf{G}$ is *non-selfloop activity loop* free if G' is ALF.

The proposition below states that if individual DES are all ALF, then the synchronous product of these DES is also ALF. This means that we can simply check the individual DES to check whether the system is ALF instead of checking the whole system at once.

Proposition 3.1. [Wan09] For TDES G_1 and G_2, if G_1 and G_2 are each ALF then their synchronous product $\mathbf{G} = G_1 \parallel G_2$ is ALF.

3.4.3 Proper Time Behaviour

To make sure that the closed-loop system would not reach a state at which no more *tick* events are ever possible, it is required for the plant to have proper time

behaviour, as defined by Kai Wong et. al. [WW96].

Definition 3.5. TDES G has a *proper time behaviour* if

$$(\forall q \in Q_{reach})(\exists \sigma \in \Sigma_u \cup tick)\delta(q, \sigma)!$$

The above definition says that at every reachable state of the system, either an uncontrollable event or a *tick* event must be possible.This condition ensures that when a controllable event is disabled by a supervisor, there is either a *tick* event or an uncontrollable event still possible. This condition is applied only to the plant TDES.

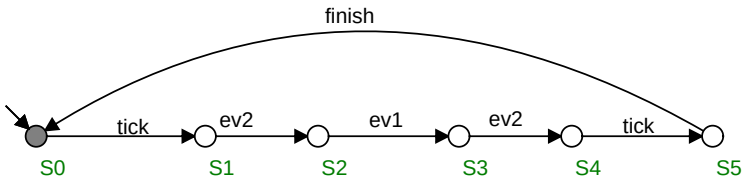


Figure 3.4: Example TDES for SD Properties

The TDES in Figure 3.4 fails proper time behaviour since at state **S5** only a prohibitable event $finish$ is eligible which could be disabled by the supervisor effectively "stopping the clock".

3.4.4 S-singular Prohibitable Behaviour

Another restriction on our system is S-singular prohibitable behaviour, where TDES S is the supervisor. This property reflects the fact that SD controllers only allow prohibitable events to occur once during a sampling period. This requirement make sure the plant models this correctly, instead of showing behavior that will never physically occur. We define this property as:

Definition 3.6. For TDES $\mathrm{G} = (Q, \Sigma, \delta, q_0, Q_m)$ and TDES $\mathrm{S} = (X, \Sigma, \xi, x_0, X_m)$, G has *S-singular prohibitable behaviour* if

$$(\forall s \in L(S) \cap L(G) \cap L_{samp})(\forall s \in \Sigma^*_{act}) ss' \in L(S) \cap L(G) \Longrightarrow$$
$$(\forall \sigma \in Occu(s') \cap \Sigma_{hib}) \sigma \notin Elig_{L(G)}(ss')$$

The TDES **G** in Figure 3.4 fails S-singular prohibitable behaviour as the prohibitable event $ev2$ occurs twice in a sampling period.

3.4.5 SD Controllable Languages

The conditions that a TDES closed-loop system has finite statespace, be ALF and nonblocking, that the plant have proper time behaviour and be complete for the supervisor, and that the supervisor be controllable for the plant does not guarantee that the system behaviour under the control of the SD controllers would be nonblocking and controllable. Therefore another property called SD controllability is introduced to address these concerns and is defined as:

Definition 3.7. A supervisor $\mathrm{S} = (X, \Sigma, \xi, x_0, X_m)$ is said to be *SD controllable* with respect to $\mathrm{G} = (Q, \Sigma, \delta, q_0, Q_m)$ if, $\forall s \in L(S) \cap L(G)$, the following statements are satisfied:

i) $Elig_{L(G)}(s) \cap \Sigma_u \subseteq Elig_{L(S)}(s)$

ii) If $tick \in Elig_{L(G)}(s)$ then

$$\text{tick} \in Elig_{L(S)}(s) \Leftrightarrow Elig_{L(S)\cap L(G)}(s) \cap \Sigma_{hib} = \emptyset$$

iii) If s $\in L_{samp}$ then

1. $(\forall s \in \Sigma^*_{act})[ss' \in L(S) \cap L(G)] \Longrightarrow$
$$[Elig_{L(S)\cap L(G)}(ss') \cup Occu(s')] \cap \Sigma_{hib} = Elig_{L(S)\cap L(G)}(s) \cap \Sigma_{hib}$$
2. $(\forall s', s'' \in L_{conc})[ss', ss'' \in L(S) \cap L(G) \wedge Occu(s') = Occu(s'')] \Rightarrow$
$$ss' \equiv_{L(S)\cap L(G)} ss'' \wedge ss' \equiv_{L_m(S)\cap L_m(G)} ss''$$

iv) $L_m(S) \cap L_m(G) \subseteq L_{samp}$

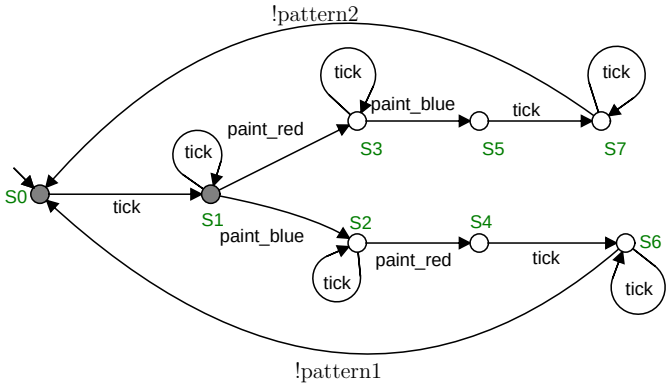


Figure 3.5: SD Controllability Example 1 - Plant

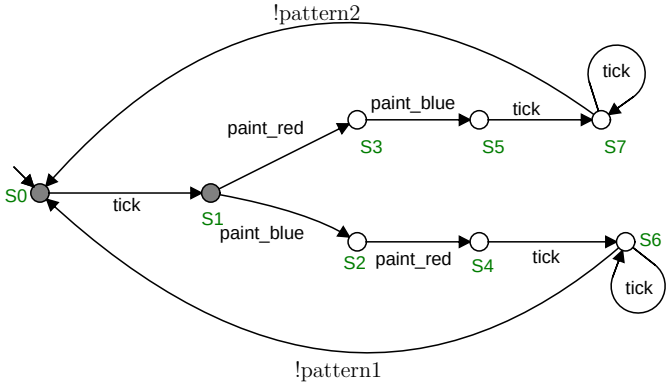


Figure 3.6: SD Controllability Example 1 - Supervisor

Point (i) in this definition is the standard untimed controllability condition. Combined with point ii ($\Leftarrow$) part it implies standard timed DES controllability. Point ii ($\Rightarrow$) part states that we must disable a *tick* event if we enable a prohibitable event that is possible in the plant. Point (iii.1) states that if an event is possible in a clock period, it must be possible immediately after the *tick* and stay possible in the clock period until it occurs. Point (iii.2) says all concurrent strings with the same occurence image in a clock period must have the same future with respect to the closed and marked behaviour of the system. Point (iv) says all marked strings must be sampled strings. Combined with point (iii.2) this ensures that if our TDES system is nonblocking then our plant and SD controller will be

nonblocking as well.

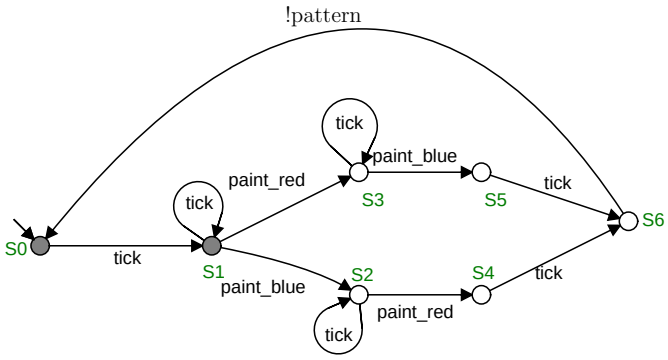


Figure 3.7: SD Controllability Example 2 - Plant

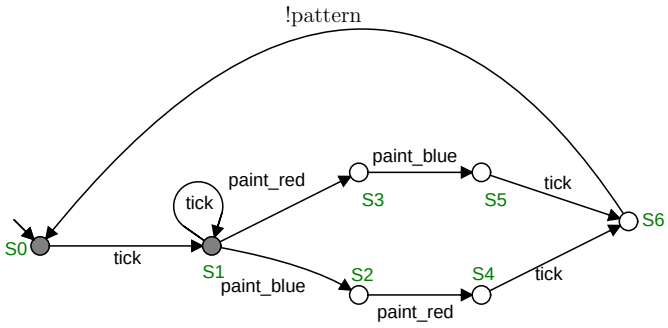


Figure 3.8: SD Controllability Example 2 - Supervisor

3.4.5.1 SD Controllability Examples

Figures 3.5 and 3.6 show a plant and supervisor such that the plant and supervisor satisfy SD controllability point(i), point(ii) and point(iii.1), but fails to satisfy point (iii.2) and (iv). In the system, state **S1**, which is a sampled state, leads to states **S6** and **S7** by concurrent strings with the same occurence image $\{paint\_blue, paint\_red, tick\}$.

According to point(iii.2), the states **S6** and **S7** should represent the same

nerode equivalent cells of the system with respect to its closed and marked behaviour. But state **S6** leads to state **S0** by *pattern1* event, while state **S7** leads to state **S0** by *pattern2* event. Clearly the states do not have the same future with respect to the closed behaviour of the system.

Also, event *pattern1* and *pattern2* lead to State S0 which is a marked state. This violates point(iv) of SD controllability definition as point(iv) requires all marked strings to be sampled strings; thus only the *tick* event should lead us to a marked state.

In Figure 3.7 and 3.8, we show a modified version of the the system from Figure 3.5 and 3.6. In this example, the system fails to satisfy point (ii), but satisfies the point(iii.2) of SD controllability definition. At state S1, the event *tick* and the prohibitable events *paint_blue*, and *paint_red* are eligible. According to point(ii), *tick* should be disabled by the supervisor if there is a prohibitable event eligible to preempt *tick*.

CHAPTER 4

Compositional Verification for SD Properties

In this thesis, we present new definitions for the sampled data supervisory control properties. These new representations express the required properties for sampled-data supervisory control in terms of other properties such as controllability, nonblocking or language inclusion. This makes it easy to see exactly what checks need to be performed for compositional verification, and on which components. These properties already have existing compositional methods defined for them, so the methods can be reused.

We have organised the compositional verification of the properties required for sampled-data supervisory control into two separate chapters for better readability. In this chapter, we discuss the properties plant completeness, activity loop free, S-singular prohibitable behaviour and proper time behaviour. The next chapter focuses on SD controllability. We will show that these new representations are equivalent to the definitions of the properties that we presented in Section 3.4.

4.1 Plant Completeness

We now restate the plant completeness property (see page 19) in terms of standard untimed controllability. We can state the untimed controllability definition from Section 2.2.3 in the alternate form below:

Definition 4.1. Let TDES G = $(Q, \Sigma, \delta, q_0, Q_m)$ be a plant and TDES S =

$(X, \Sigma, \xi, x_0, X_m)$ be a supervisor. S is said to be controllable with respect to G if

$$(\forall s \in L(S))(\forall \sigma \in \Sigma_u) s\sigma \in L(G) \Longrightarrow s\sigma \in L(S)$$

Comparing the plant completeness definition to the above definition, we can easily see that we can adapt the controllability check by relabelling plants as supervisors, supervisors as plants, prohibitable events as uncontrollable events, and uncontrollable events as controllable events. After relabelling, we can just apply the compositional verification algorithm for standard controllability [BMM04] on the relabelled system.

4.2 Activity Loop Free

In this section, we develop a method to verify if a TDES **G** is Activity-loop free (ALF) (see definition on page 20). To do this we will use the concept of paths.

Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$. A path in **G** is an alternating sequence of states and events

$$\pi = q_0 \xrightarrow{\sigma_1} ..... \xrightarrow{\sigma_n} q_n$$

where $\delta(q_i, \sigma_{i+1}) = q_{i+1}$ for all $0 \leq i \leq n$. State q_0 is called its origin and state q_n its end. $\mathrm{L}(\pi) = \sigma_1.....\sigma_n$ is the label of path π. The language accepted by the TDES G is

$$\mathrm{L(G)} = \{\ \mathrm{L}(\pi) \mid \pi \text{ is a path in G with } q_0 \text{ as origin } \}\ .$$

Definition 4.2. For TDES $\mathrm{G} = (Q, \Sigma, \delta, q_0, Q_m)$ and $\Sigma_{act} \subseteq \Sigma$, a path π in G is called a Σ_{act}-path if $\mathrm{L}(\pi) \in \Sigma_{act}^*$. A Σ_{act}-loop is a Σ_{act}-path of length (i.e. $|L(\pi)|$) at least one with the same origin and end. A Σ_{act}-loop is reachable in G if its origin is reachable in G. If G does not contain any reachable Σ_{act}-loops, G is called Σ_{act}-loop free.

For two TDES defined over the same alphabet set, it is sufficient to show that only one of the TDES is Σ_{act}-loop free, to show that the whole system is Σ_{act}-loop free [MM06].

The Lemma below shows that we can use the existing modular loop-detecting algorithm [MM06] to determine if **G** is ALF.

Lemma 4.1. ***G*** *is activity loop free iff G is* Σ_{act}*-loop free.*

Proof. This is obvious from the ALF definition and the Σ_{act}-loop free definition. □

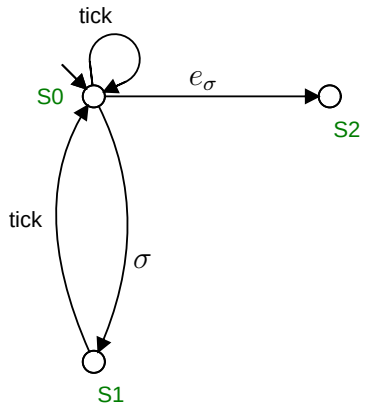
4.3 S-singular Prohibitable Behaviour

The S-singular prohibitable behaviour property (see page 22) requires that a prohibitable event be allowed to occur in the plant only once during a sampling period. We can state the definition in the alternative form below:

Definition 4.3. Let TDES $G = (Y, \Sigma, \delta, y_0, Y_m)$ and TDES $S = (X, \Sigma, \xi, x_0, X_m)$. G has S-singular prohibitable behaviour if

$(\forall s \in L(G) \cap L(S) \cap L_{samp})(\forall t \in \Sigma^*_{act})(\forall \sigma \in \Sigma_{hib})$

$$(st \in L(S) \cap L(G)) \wedge (st\sigma \in L(G)) \Rightarrow \sigma \notin Occu(t)$$



4.1: TDES T^σ

Definition 4.4. Let G = $(X, \Sigma, \delta, x_0, X_m), \sigma \in \Sigma_{hib}, e_\sigma \notin \Sigma, E_\sigma = \{\sigma, e_\sigma, tick\}$. Let $P_{E_\sigma} : \Sigma^* \to E_\sigma^*$ be a natural projection. Construct TDES G^σ such that L(G^σ) = L(G) $\cup\{se_\sigma \mid s\sigma \in L(G)\}$.

We express this property as a language inclusion problem by introducing, for each $\sigma \in \Sigma_{hib}$, a new DES T^σ which exhibits the behaviour that the event σ can only occur once between tick events. We then construct $T^\sigma = (X_T, \Sigma_T, \delta_T, x_0, X_{T_m})$ as shown in Figure 4.1.

We now present two lemmas related to T^σ that will be useful later, in this section.

Lemma 4.2. $L(T^\sigma) = \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

Proof. It is sufficient to show:

I) L(T^σ) $\supseteq \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$ and II) L(T^σ) $\subseteq \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

Part I) Show: L(T^σ) $\supseteq \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

Let s $\in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

We will show that s $\in L(T^\sigma)$

Having s $\in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$ implies that there exists $s' \in E_\sigma^*$ such that s$s' \in (\{\epsilon, \sigma\}tick)^* e_\sigma$.

Let $s'' = ss'$

Since $s'' \in (\{\epsilon, \sigma\}tick)^* e_\sigma$, there exists t $\in (\{\epsilon, \sigma\}tick)^*$ such that $s'' =$ te_σ.

We note that $\{\epsilon, \sigma\}tick = \{tick, \sigma tick\}$.

From Figure 4.1, we see that $\delta_T(S0, tick) = S0$

Also $\delta_T(S0, \sigma) = S1$ and $\delta_T(S1, tick) = S0$.

This implies $\delta_T(S0, t') = S0$ for all $t' \in \{\epsilon, \sigma\}tick$. (1)

Now we will show that for all t $\in (\{\epsilon, \sigma\}tick)^*$ it holds that $\delta_T(S0, t) = S0$.

Since t $\in (\{\epsilon, \sigma\}tick)^*$ there exists n $\in N_0$ such that t $\in (\{\epsilon, \sigma\}tick)^n$

We will show by induction on n that $\delta_T(S0, t) = S0$

Inductive Base: n = 0

t $\in (\{\epsilon, \sigma\}tick)^0 \Rightarrow$ t $= \epsilon$

Clearly $\delta_T(S0, \epsilon) = S0$.

Inductive Step: n $\rightarrow$ n+1, t $\in (\{\epsilon, \sigma\}tick)^{n+1}$

By inductive assumption, $\delta_T(S0, t) = S0$ for t $\in (\{\epsilon, \sigma\}tick)^n$.

$t' \in (\{\epsilon, \sigma\}tick)^{n+1} \Rightarrow t' \in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}tick$

By (1) we have $\delta_T(S0, t'') = S0$ for all $t'' \in \{\epsilon, \sigma\}tick$ and by inductive assumption we have $\delta_T(S0, t) = S0$ for t $\in (\{\epsilon, \sigma\}tick)^n$.

This implies that $\delta_T(S0, t') = S0$ for $t' \in (\{\epsilon, \sigma\}tick)^{n+1}$.

Inductive step complete.

We thus conclude that for t $\in (\{\epsilon, \sigma\}tick)^*$ it holds that $\delta_T(S0, t) = S0$

We have $s'' =$ te_σ

Clearly from the Figure 4.1, $\delta_T(S0, e_\sigma) = S2$.

$\Longrightarrow s'' \in L(T^\sigma)$

Since $L(T^\sigma)$ is a prefix closed language and s $\leq s''$, we have $s \in L(T^\sigma)$ as required.

Part I complete.

Part II) Show: L$(T^\sigma) \subseteq \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

Let s $\in L(T^\sigma)$.

We now show that s $\in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$.

Since s $\in L(T^\sigma)$ this implies $\delta_T(S0, s) = X$ for some X $\in \{S0, S1, S2\}$.

We will first examine case X $= S0$ and X $= S1$.

Clearly from Figure 4.1, X $= S0$ or X $= S1$ implies s does not contain e_σ.

First we will show by induction on $|s|$ that:

1) If $\delta_T(S0, s) = S0$ then s $\in (\{\epsilon, \sigma\}tick)^*$ and

2) if $\delta_T(S0, s) = S1$ then s $\in (\{\epsilon, \sigma\}tick)^*\sigma$

Inductive Base: n $= 0$, s $= \epsilon$

Clearly $\epsilon \in (\{\epsilon, \sigma\}tick)^*$ and $\delta_T(S0, \epsilon) = S0$.

Inductive Step: n $\rightarrow$ n+1, s $=$ tα with $\alpha \in \{\sigma, tick\}$

By inductive assumption:

1) If $\delta_T(S0, t) = S0$ then t $\in (\{\epsilon, \sigma\}tick)^*$ and

2) If $\delta_T(S0, t) = S1$ then t $\in (\{\epsilon, \sigma\}tick)^*\sigma$

Since s $\in L(T^\sigma)$ there exists state y such that $\delta_T(S0, t)$ = y and δ_T(y,α)=X.

Clearly from the Figure 4.1, y is either $S0$ or $S1$.

Case (X = $S0$)

Clearly from Figure 4.1, only a tick transition allows $\delta_T(y, \alpha) = S0$, thus $\alpha = tick$.

case i) if (y = $S0$)

$\Longrightarrow \delta_T(S0, t) = S0$

By inductive assumption, t $\in (\{\epsilon, \sigma\}tick)^*$

$\Longrightarrow s = t\alpha \in (\{\epsilon, \sigma\}tick)^*tick$

$\Longrightarrow s \in (\{\epsilon, \sigma\}tick)^*$

case ii) if (y = $S1$)

$\Longrightarrow \delta_T(S0, t) = S1$

By inductive assumption, t $\in (\{\epsilon, \sigma\}tick)^*\sigma$

$\Longrightarrow s = t\alpha \in (\{\epsilon, \sigma\}tick)^*\sigma tick$

$\Longrightarrow s \in (\{\epsilon, \sigma\}tick)^*$

By case i) and ii) we can conclude that

if $\delta_T(S0, s) = S0$ then s $\in (\{\epsilon, \sigma\}tick)^*$ (2)

Case (X = $S1$)

Clearly from Figure 4.1, only a σ transition allows $\delta_T(y, \alpha) = S1$ only from the state $S0$.

$\Rightarrow \alpha = \sigma$ and y = $S0$

$\Longrightarrow \delta_T(S0, t) = S0$

By inductive assumption, t $\in (\{\epsilon, \sigma\}tick)^*$ since $\delta_T(S0, \sigma) = S1$

$\Longrightarrow s = t\alpha \in (\{\epsilon, \sigma\}tick)^*\sigma$

$\Longrightarrow s \in (\{\epsilon, \sigma\}tick)^*\sigma$ as required.

Inductive step complete.

We have now shown that:

(a) for X = S0, s $\in (\{\epsilon, \sigma\}tick)^* \subseteq \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$ (3)

(b) for X = S1, s $\in (\{\epsilon, \sigma\}tick)^*\sigma \subseteq \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

Now we examine the case $\delta_T(S0, s) = S2$

Clearly from Figure 4.1, $\delta_T(S0, e_\sigma) =$ S2 and this is the only way to get to S2.

$\Longrightarrow (\exists t)$ s = te_σ and $\delta_T(S0, t) = S0$.

By (3), this implies t $\in (\{\epsilon, \sigma\}tick)^*$

$\Longrightarrow s = te_\sigma \in (\{\epsilon, \sigma\}tick)^* e_\sigma \subseteq \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

Now we have shown for all X $\in$ {S0, S1, S2 } , s $\in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

Part II complete.

Thus we can conclude that L(T^σ) = $\overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$ □

Lemma 4.3. *Let $\sigma \in \Sigma$ and $E_\sigma = \{\sigma, e_\sigma, tick\}$. Then*
$s \in \overline{(\{\epsilon, \sigma\}tick)^*}$ iff $s \in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some $n \in N_0 = \{$ 0,1,2,3,.....$\}$

Proof. To show our claim, we have to show:

I) s $\in \overline{(\{\epsilon, \sigma\}tick)^*} \Longrightarrow$ s $\in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some n $\in N_0$ and

II) s $\in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some n $\in N_0 \Longrightarrow$ s $\in \overline{(\{\epsilon, \sigma\}tick)^*}$

Part I) Show: s $\in \overline{(\{\epsilon, \sigma\}tick)^*} \Longrightarrow$ s $\in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some n $\in N_0$.

Let s $\in \overline{(\{\epsilon, \sigma\}tick)^*}$.

This implies there exists $s' \in E_\sigma^*$ such that s$s' \in (\{\epsilon, \sigma\}tick)^*$.

This implies there exists some n $\in N_0$ such that s$s' \in (\{\epsilon, \sigma\}tick)^n$.

Appending $\{\epsilon, \sigma\}$ to both sides we get

s$s'\{\epsilon, \sigma\} \in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ from some n $\in N_0$.

Since s $\leq$ s$s'\{\epsilon, \sigma\} \in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ from some n $\in N_0$.

Clearly by taking $n' \leq$ n, we can use $(\{\epsilon, \sigma\}tick)^{n'}\{\epsilon, \sigma\}$ to contain any prefix of s$s'\{\epsilon, \sigma\}$.

This implies s $\in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ from some n $\in N_0$ as required.

Part I complete.

Part II) Show: s $\in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some n $\in N_0 \Longrightarrow$ s $\in \overline{(\{\epsilon, \sigma\}tick)^*}$.

Let s $\in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some n $\in N_0$.

Appending *tick* to both sides, we get:

s tick $\in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}tick$ for some n $\in N_0$.

$\Longrightarrow$ s tick $\in (\{\epsilon, \sigma\}tick)^{n+1}$ for some n $\in N_0$.

$\Longrightarrow$ s tick $\in (\{\epsilon, \sigma\}tick)^*$.

$\Longrightarrow$ s $\in \overline{(\{\epsilon, \sigma\}tick)^*}$, as required.

Part II complete.

By part I and II we conclude that s $\in \overline{(\{\epsilon, \sigma\}tick)^*}$ iff s $\in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some n $\in N_0 = \{0,1,2,3,.....\}$. □

We will prove that our language inclusion property is equivalent to the S-singular prohibitable behaviour property. The proof that follows is based on an initial proof sketch by the author's thesis supervisors, Dr. R. Malik and Dr. R. Leduc.

Theorem 4.1. *Let TDES* $G = (X, \Sigma, \delta, x_0, X_m)$ *and* $S = (Y, \Sigma, \xi, y_0, Y_m)$. *G has S-singular prohibitable behaviour iff*

$(\forall \sigma \in \Sigma_{hib})\ P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma)$

Proof. To prove this claim, we will need to show

I) G has S-singular prohibitable behaviour

$$\Rightarrow (\forall \sigma \in \Sigma_{hib}) P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma).$$

II) $(\forall \sigma \in \Sigma_{hib}) P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma)$

$$\Rightarrow \text{G has S-singular prohibitable behaviour.}$$

Part I) Show G has S-singular prohibitable behaviour $\Longrightarrow$

$$(\forall \sigma \in \Sigma_{hib}) P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma)$$

Assume G has S-singular prohibitable behaviour.

Let $\sigma \in \Sigma_{hib}$, $s \in L(G^\sigma \parallel S)$ (1)

We show by induction on $n = |s|$ that $P_{E_\sigma}(s) \in L(T^\sigma)$.

Sufficient to show that $P_{E_\sigma}(s) \in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$ by Lemma 4.2

Inductive Base: n = 0, $s = \epsilon$

Then $P_{E_\sigma}(s) = P_{E_\sigma}(\epsilon) = \epsilon \in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

Inductive Step: $n \to n+1, s = t\alpha$

By inductive assumption, $P_{E_\sigma}(t) \in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

We know by (1) that $t\alpha = s \in L(G^\sigma) = L(G) \cup \{se_\sigma \mid s\sigma \in L(G)\}$.

That means if s contains e_σ, then e_σ is only the last event of s.

That is $e_\sigma \notin Occu(t)$, and t $\in \Sigma^*$.

Therefore $P_{E_\sigma}(t) \in \overline{(\{\epsilon, \sigma\}tick)^*}$ and $t \in L(G)$ (by construction of G^σ). (2)

Also, $t \leq s \in L(G^\sigma \parallel S)$ and $t \in \Sigma^* \Rightarrow t \in L(G) \cap L(S)$. (3)

Consider four cases:

a) $\alpha \notin \{\sigma, e_\sigma, tick\}$

Then $P_{E_\sigma}(s) = P_{E_\sigma}(t\alpha) = P_{E_\sigma}(t) \in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$ by inductive assumption.

b) $\alpha = tick$

Then since $P_{E_\sigma}(t) \in \overline{(\{\epsilon, \sigma\}tick)^*}$ by (2) we have:

$P_{E_\sigma}(t) \in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some n $\in N_0$, by Lemma 4.3.

Thus, $P_{E_\sigma}(s) = P_{E_\sigma}$ (t tick) $= P_{E_\sigma}(t)tick$ implies that:

$P_{E_\sigma}(s) \in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}tick$

$\Rightarrow P_{E_\sigma}(s) \in (\{\epsilon, \sigma\}tick)^{n+1}$

$\Rightarrow P_{E_\sigma}(s) \in (\{\epsilon, \sigma\}tick)^*$

Appending e_σ to this string gives: $P_{E_\sigma}(s)e_\sigma \in (\{\epsilon, \sigma\}tick)^* e_\sigma$

We thus have: $P_{E_\sigma}(s) \in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

c) $\alpha = \sigma$

Let t =uv such that $u \in L_{samp}$ and $v \in \Sigma^*_{act}$. We thus have:

$P_{E_\sigma}(u) \leq P_{E_\sigma}(uv) = P_{E_\sigma}(t) \in \overline{(\{\epsilon, \sigma\}tick)^*}$

$\Longrightarrow P_{E_\sigma}(u) \in \overline{(\{\epsilon, \sigma\}tick)^*}$

$\Longrightarrow P_{E_\sigma}(u) \in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some n $\in N_0$ by Lemma 4.3

Since $u \in L_{samp}$, $P_{E_\sigma}(u)$ cannot end with σ. We thus have:

$P_{E_\sigma}(u) \in (\{\epsilon, \sigma\}tick)^n \subseteq (\{\epsilon, \sigma\}tick)^*$.

From (3), it follows that $uv = t \in L(G) \cap L(S)$ and that:

$u \leq t \Longrightarrow u \in L(G) \cap L(S) \cap L_{samp}$.

Also $s = t\sigma \in L(G^\sigma) = L(G) \cup \{we_\sigma \mid w\sigma \in L(G)\}$ with $\sigma \neq e_\sigma$

$\Rightarrow uv\sigma = s \in L(G)$.

Then, given that $\sigma \in \Sigma_{hib}$, it follows from the definition of S-singular prohibitable behaviour that $\sigma \notin Occu(v)$.

$\Longrightarrow P_{E_\sigma}(v) = \epsilon$ since $v \in \Sigma^*_{act}$.

Thus, $P_{E_\sigma}(t) = P_{E_\sigma}(uv) = P_{E_\sigma}(u) \in (\{\epsilon, \sigma\}tick)^*$.

Appending $\sigma tick e_\sigma$ to this string gives:

$P_{E_\sigma}(s)tick e_\sigma = P_{E_\sigma}(t\sigma)tick e_\sigma = P_{E_\sigma}(t)\sigma tick e_\sigma$

$\Rightarrow P_{E_\sigma}(s)tick e_\sigma \in (\{\epsilon, \sigma\}tick)^*\sigma tick e_\sigma$

$\Rightarrow P_{E_\sigma}(s)tick e_\sigma \in (\{\epsilon, \sigma\}tick)^+ e_\sigma$

$\Rightarrow P_{E_\sigma}(s)tick e_\sigma \in (\{\epsilon, \sigma\}tick)^* e_\sigma$

We thus have: $P_{E_\sigma}(s) \in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

d) $\alpha = e_\sigma$

Let t = uv such that $u \in L_{samp}$ and $v \in \Sigma^*_{act}$.

Since $P_{E_\sigma}(u) \leq P_{E_\sigma}(uv) = P_{E_\sigma}(t) \in \overline{(\{\epsilon, \sigma\}tick)^*}$, we have:

$P_{E_\sigma}(u) \in \overline{(\{\epsilon, \sigma\}tick)^*}$

$\Longrightarrow P_{E_\sigma}(u) \in (\{\epsilon, \sigma\}tick)^n\{\epsilon, \sigma\}$ for some n $\in N_0$ by Lemma 4.3.

Since $u \in L_{samp}$, $P_{E_\sigma}(u)$ cannot end with σ. This implies:

$P_{E_\sigma}(u) \in (\{\epsilon, \sigma\}tick)^n \subseteq (\{\epsilon, \sigma\}tick)^*$.

From (3), it follows that $uv = t \in L(G) \cap L(S)$ and that:

$u \leq t \Rightarrow u \in L(G) \cap L(S) \cap L_{samp}$.

Also, $s = te_\sigma \in L(G^\sigma) = L(G) \cup \{we_\sigma \mid w\sigma \in L(G)\}$ implies $uv\sigma = t\sigma \in L(G)$.

Then, given that $\sigma \in \Sigma_{hib}$, it follows from the definition of S-singular prohibitable behaviour that: $\sigma \notin Occu(v)$.

Since $v \in \Sigma^*_{act}$, this implies $P_{E_\sigma}(v) = \epsilon$.

Thus, $P_{E_\sigma}(t) = P_{E_\sigma}(uv) = P_{E_\sigma}(u) \in (\{\epsilon, \sigma\}tick)^*$.

$\Longrightarrow P_{E_\sigma}(s) = P_{E_\sigma}(te_\sigma) = P_{E_\sigma}(uve_\sigma) = P_{E_\sigma}(u)e_\sigma \in (\{\epsilon, \sigma\}tick)^* e_\sigma$

We thus have: $P_{E_\sigma}(s) \in \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$

Part I complete.

Part II) Show $(\forall \sigma \in \Sigma_{hib}) P_{E_\sigma}(L(G^\sigma) \parallel S)) \subseteq L(T^\sigma) \Longrightarrow$ G has S-singular prohibitable behaviour.

Assume $(\forall \sigma \in \Sigma_{hib}) P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma)$.

Let $s \in L(G) \cap L(S) \cap L_{samp}, t \in \Sigma^*_{act}, \sigma \in \Sigma_{hib}$ such that $st \in L(G) \cap L(S)$ and $st\sigma \in L(G)$.

Must show $\sigma \notin Occu(t)$

Since $st\sigma \in L(G)$, by construction $ste_\sigma \in L(G^\sigma)$

Let $P_\Sigma : (\Sigma \cup \{e_\sigma\})^* \to \Sigma^*$ be a natural projection.

Since $st \in L(S)$, we have $P_\Sigma(ste_\sigma) = st \in L(S)$

$\Longrightarrow ste_\sigma \in L(G^\sigma \parallel S)$

$\Longrightarrow P_{E_\sigma}(st)e_\sigma = P_{E_\sigma}(ste_\sigma) \in P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma) = \overline{(\{\epsilon, \sigma\}tick)^* e_\sigma}$, by Lemma 4.2.

Then there exists $v \in E^*_\sigma$ such that $P_{E_\sigma}(st)e_\sigma v \in (\{\epsilon, \sigma\}tick)^* e_\sigma$.

This implies $v = \epsilon$ because e_σ cannot occur in $\{\epsilon, \sigma\}tick$.

$\Longrightarrow P_{E_\sigma}(st) \in (\{\epsilon, \sigma\}tick)^*$.

Since $P_{E_\sigma}(s) \leq P_{E_\sigma}(s)P_{E_\sigma}(t) = P_{E_\sigma}(st)$ it follows that $P_{E_\sigma}(s) \in \overline{(\{\epsilon, \sigma\}tick)^*}$

Then by Lemma 4.3, there exists $n \in N_0$ such that $P_{E_\sigma}(s) \in (\{\epsilon, \sigma\}tick)^n \{\epsilon, \sigma\}$.

$\Longrightarrow P_{E_\sigma}(s) \in (\{\epsilon, \sigma\}tick)^n$ since $s \in L_{samp}$ and $P_{E_\sigma}(s)$ cannot end with σ.

Combining with $P_{E_\sigma}(st) \in (\{\epsilon, \sigma\}tick)^*$ we see that there also exists $m \in N_0$ such that $P_{E_\sigma}(t) \in (\{\epsilon, \sigma\}tick)^m$.

However, since $t \in \Sigma^*_{act}$, t does not contain any occurrences of tick, so it follows that m=0, i.e. $P_{E_\sigma}(t) = \epsilon$.

Given $\sigma \in E_\sigma$, this implies $\sigma \notin Occu(t)$.

Part II complete.

By part I and II we conclude that G has S-singular prohibitable behaviour iff $(\forall \sigma \in \Sigma_{hib})\ P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma)$ □

4.4 Proper Time Behaviour

A TDES G has proper time behaviour (see page 22) if at every reachable state, either a tick or an uncontrollable event is possible.

Definition 4.5. Let $G = (Q, \Sigma, \delta, q_0, Q_m)$ be a TDES and $\Upsilon \subseteq \Sigma$. G is called Υ-*enabling* if for every trace $s \in L(G)$ there exists $\sigma \in \Upsilon$ such that $s\sigma \in L(G)$.

We will now show that if we take $\Upsilon = \Sigma_u \cup \{tick\}$, then verifying that **G** has has proper time behaviour is equivalent to verifying that **G** is Υ-enabling.

Lemma 4.4. *G has proper time behaviour iff G is $(\Sigma_u \cup \{tick\})$-enabling.*

Proof. To prove our claim we will show:

1) G has proper time behaviour implies G is $(\Sigma_u \cup \{\text{tick}\})$-enabling and

2) G is $(\Sigma_u \cup \{\text{tick}\})$-enabling implies G has proper time behaviour.

Part 1) Show G has proper time behaviour implies G is $(\Sigma_u \cup \{\text{tick}\})$-enabling.

Assume G has proper time behaviour.

By **Definition 3.5** (see page 22), this implies for every reachable state q of G $(\exists \sigma \in \Sigma_u \cup \{tick\})$ such that $\delta(q, \sigma)!$

We can rewrite this definition as

$(\forall s \in L(G))(\exists \sigma \in \Sigma_u \cup \{tick\})$ such that $s\sigma \in$ L(G)

It immediately follows that G is $(\Sigma_u \cup \{tick\})$-enabling, as required.

Part 2) Show G is $(\Sigma_u \cup \{\text{tick}\})$-enabling implies G has proper time behaviour.

Assume G is $(\Sigma_u \cup \{\text{tick}\})$-enabling.

From **Definition 4.5** we have:

$(\forall s \in L(G))(\exists \sigma \in \Sigma_u \cup \{tick\})$ such that $s\sigma \in$ L(G)

From **Definition 3.5** it follows that G has proper time behaviour. □

Let $G = (Q, \Sigma, \delta, q_0, Q_m)$, $\Upsilon \subseteq \Sigma$ and $? \notin \Sigma$. We will now show that we can express the Υ-enabling property as a nonblocking problem by first marking all states of the plant DES G. To do this we construct G^ω such that $L(G^\omega) = L_m(G^\omega)$ = L(G). Finally, we need to introduce a new TDES T_Υ as shown in Figure 4.2.

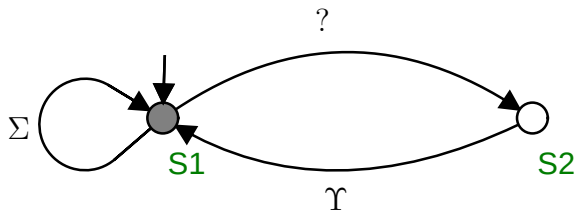


Figure 4.2: TDES T_Υ

We now prove that our nonblocking property is equivalent to the Υ-enabling property.

Theorem 4.2. *G is Υ-enabling iff $G^\omega \parallel T_\Upsilon$ is nonblocking*

Proof. We must show that:

I) G is Υ-enabling $\Rightarrow$ $G^\omega \parallel T_\Upsilon$ is nonblocking and

II) $G^\omega \parallel T_\Upsilon$ is nonblocking $\Rightarrow$ G is Υ-enabling.

Part I) Show G is Υ-enabling $\Rightarrow$ $G^\omega \parallel T_\Upsilon$ is nonblocking

Assume G is Υ-enabling. (1)

Let $\Omega = \Sigma \cup \{?\}$, and $P_\Sigma : \Omega^* \to \Sigma^*$ be a natural projection.

Let $s \in L(G^\omega \parallel T_\Upsilon) = P_\Sigma^{-1} L(G^\omega) \cap L(T_\Upsilon)$ (2)

Sufficient to show that: $(\exists s' \in \Omega^*) ss' \in P_\Sigma^{-1} L_m(G^\omega) \cap L_m(T_\Upsilon)$

Consider two cases:

Case i) $s \in L_m(T_\Upsilon)$

From (2), we have $s \in P_\Sigma^{-1} L(G^\omega)$.

$\Rightarrow s \in P_\Sigma^{-1} L_m(G^\omega)$ as $L(G^\omega) = L_m(G^\omega)$

$\Rightarrow s \in P_\Sigma^{-1} L_m(G^\omega) \cap L_m(T_\Upsilon)$

We then take $s' = \epsilon$, and we have

$ss' \in P_\Sigma^{-1} L_m(G^\omega) \cap L_m(T_\Upsilon)$, as required.

Case ii) $s \notin L_m(T_\Upsilon)$

From (2), we have $s \in P_\Sigma^{-1} L(G)$ as $L(G^\omega) = L(G)$

From (1) we have $(\exists \sigma \in \Upsilon)$ such that $P_\Sigma(s)\sigma \in L(G) = L_m(G^\omega)$

As $\sigma \in \Upsilon \subseteq \Sigma, P_\Sigma(\sigma) = \sigma$. We thus have:

$P_\Sigma(s)\sigma = P_\Sigma(s)P_\Sigma(\sigma) = P_\Sigma(s\sigma)$

$\Rightarrow P_\Sigma(s\sigma) \in L_m(G^\omega)$

$\Rightarrow s\sigma \in P_\Sigma^{-1} L_m(G^\omega)$

By construction of T_Υ , $s \notin L_m(T_\Upsilon)$ implies s takes T_Υ to state S2

As $\sigma \in \Upsilon$, this implies $s\sigma \in L_m(T_\Upsilon)$. We thus have:

$s\sigma \in P_\Sigma^{-1} L_m(G^\omega) \cap L_m(T_\Upsilon)$

Taking $s' = \sigma$ we have:

$ss' \in P_\Sigma^{-1} L_m(G^\omega) \cap L_m(T_\Upsilon)$, as required.

Part II) Show: $G^\omega \parallel T_\Upsilon$ is nonblocking $\Rightarrow$ G is Υ-enabling

Assume $G^\omega \parallel T_\Upsilon$ is nonblocking. (3)

We will show that this implies that G is Υ-enabling.

Let $s \in L(G) = L(G^\omega)$

Sufficient to show that $(\exists \sigma \in \Upsilon)$ such that $s\sigma \in L(G)$

By construction of $T_\Upsilon, s \in L(T_\Upsilon)$ (as $s \in \Sigma^*$) and $s? \in L(T_\Upsilon)$.

$\Rightarrow s? \in P_\Sigma^{-1}L(G^\omega) \cap L(T_\Upsilon)$

From (3), we have: $(\exists s' \in \Omega^*)s?s' \in P_\Sigma^{-1}L_m(G^\omega) \cap L_m(T_\Upsilon)$

By construction of T_Υ, s? takes T_Υ to state S2

$\Rightarrow (\exists \sigma \in \Upsilon)(\exists s'' \in \Omega^*)s' = \sigma s''$

$\Rightarrow s?\sigma s'' \in P_\Sigma^{-1}L_m(G^\omega) = P_\Sigma^{-1}L(G^\omega)$

$\Rightarrow sP_\Sigma(?\sigma s'') \in L(G^\omega)$ as $s \in \Sigma^*$

$\Rightarrow sP_\Sigma(?)P_\Sigma(\sigma)P_\Sigma(s'') \in L(G^\omega)$

As $L(G^\omega)$ is prefix closed, we have:

$$sP_\Sigma(?)P_\Sigma(\sigma) \in L(G^\omega)$$

As $\sigma \in \Upsilon \subseteq \Sigma$ and $? \notin \Sigma$ we have $s\sigma \in L(G^\omega)$

$s\sigma \in L(G)$ as required.

From part I and II we can conclude that G is Υ-enabling iff $G^\omega \parallel T_\Upsilon$ is nonblocking. □

CHAPTER 5

Compositional Verification of SD Controllability

In this chapter, we present SD controllability point(ii), point(iii) and point(iv). As SD controllability point(i) implies standard controllability, we will not discuss it in detail.

We redefine the SD controllability definitions in terms of other properties that already have well defined compositional methods and show that these new representations are equivalent to the original definitions in [Wan09] that we presented in section 3.4.

5.1 SD Controllability Point (ii)

The SD controllability point (ii) (see page 23) applies to strings accepted by the closed-loop system in which *tick* is eligible in the plant afterwards. It states that if both *tick* and a prohibitable event are possible, then *tick* must be disabled by the supervisor. Also, we cannot disable a *tick* unless there is an eligible prohibitable event to preempt the *tick*. For simplicity, we restate the SD controllability point (ii) definition below in a self contained form.

Definition 5.1. Let TDES $\mathrm{G} = (Q, \Sigma, \delta, q_0, Q_m)$ and TDES $\mathrm{S} = (X, \Sigma, \xi, x_0, X_m)$. G and S satisfy SD controllability point (ii), if for all $s \in L(G) \cap L(S)$ it follows that

$$\text{if } tick \in Elig_{L(G)}(s) \text{ then } tick \in Elig_{L(S)}(s) \iff Elig_{L(S)\cap L(G)}(s) \cap \Sigma_{hib} = \emptyset$$

We can express SD controllability point(ii) in two parts.

ii.a) ($\Rightarrow$): $(\forall s \in L(S) \cap L(G))\ s\ tick \in L(S) \cap L(G) \Longrightarrow$

$$Elig_{L(S)\cap L(G)}(s) \cap \Sigma_{hib} = \emptyset$$

ii.b) ($\Leftarrow$): $(\forall s \in L(S) \cap L(G)\ s\ tick \in L(G) \Longrightarrow$

$$Elig_{L(S)\cap L(G)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset$$

Now we present a lemma related to SD controllability point (ii.b) definition.

Lemma 5.1. $(\forall s \in L(S) \cap L(G))$ *if* $s\ tick \in L(G)$ *then*

$Elig_{L(S)\cap L(G)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset \Longleftrightarrow$

$$[(Elig_{L(S)\cap L(G)}(s) \cap \Sigma_{hib} = \emptyset) \Rightarrow tick \in Elig_{L(S)}(s)].$$

Proof. Let s $\in L(S) \cap L(G)$

Assume $s\ tick \in L(G)$ (1)

We have to show that

I)$Elig_{L(S)\cap L(G)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset \Longrightarrow$

$[(Elig_{L(S)\cap L(G)} \cap \Sigma_{hib} = \emptyset) \Rightarrow tick \in Elig_{L(S)}(s)]$ and

II) $[(Elig_{L(S)\cap L(G)} \cap \Sigma_{hib} = \emptyset) \Rightarrow tick \in Elig_{L(S)}(s)] \Longrightarrow$

$Elig_{L(S)\cap L(G)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset$

Part I) Show $Elig_{L(S)\cap L(G)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset \Longrightarrow$

$\big[(Elig_{L(S)\cap L(G)} \cap \Sigma_{hib} = \emptyset) \Rightarrow tick \in Elig_{L(S)}(s)\big]$

Assume $Elig_{L(S)\cap L(G)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset$. (2)

Assume $Elig_{L(S)\cap L(G)} \cap \Sigma_{hib} = \emptyset$

$\Longrightarrow tick \in Elig_{L(S)\cap L(G)}(s)$ by (2)

$\Longrightarrow tick \in Elig_{L(S)}(s)$ as required.

Part I complete.

Part II) Show $[(Elig_{L(S)\cap L(G)} \cap \Sigma_{hib} = \emptyset) \Rightarrow tick \in Elig_{L(S)}(s)] \Longrightarrow$

$Elig_{L(S)\cap L(G)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset$

Assume $[(Elig_{L(S)\cap L(G)} \cup \Sigma_{hib} = \emptyset) \Rightarrow tick \in Elig_{L(S)}(s)]$.

This implies $\neg(Elig_{L(S)\cap L(G)} \cap \Sigma_{hib} = \emptyset) \vee tick \in Elig_{L(S)}(s)$

We have two cases.

Case i) $\neg(Elig_{L(S)\cap L(G)} \cap \Sigma_{hib} = \emptyset)$

$\implies Elig_{L(S)\cap L(G)} \cap \Sigma_{hib} \neq \emptyset$

$\implies Elig_{L(S)\cap L(G)} \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset$

Case ii) $tick \in Elig_{L(S)}(s)$

$\implies tick \in L(S) \cap L(G)$ by (1)

$\implies Elig_{L(S)\cap L(G)} \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset$

Part II complete.

By part I and II we conclude that

$Elig_{L(S)\cap L(G)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset \iff$

$$[(Elig_{L(S)\cap L(G)}(s) \cap \Sigma_{hib} = \emptyset) \Rightarrow tick \in Elig_{L(S)}(s)].$$ □

5.1.1 SD Controllability Point (ii.a)

We will now express this property as a language inclusion problem. Our first step is to express as event occurrences whether a given event in Σ is enabled at a given state in our TDES.

Definition 5.2. Let $\mathbf{G} = (X, \Sigma, \delta, x_0, X_m)$ be a TDES, $\alpha \in \Sigma, e_\alpha \notin \Sigma$. Let $P_\Sigma : (\Sigma_\alpha^E)^* \to \Sigma^*$ be a natural projection. Construct TDES $E_\alpha(G)$ with alphabet $\Sigma_\alpha^E = \Sigma \dot\cup \{e_\alpha\}$ by adding e_α selfloops to all states in $\mathbf{G}$ with α enabled. Let $E_\alpha(G) = (X, \Sigma \cup \{e_\alpha\}, \delta_E, x_0, X_m)$ and for $x \in$ X

$$\delta_E(x,\sigma) = \begin{cases} \delta(x,\sigma) & \text{if } \sigma \in \Sigma \text{ and } \delta(x,\sigma) \text{ is defined} \\ x & \text{if } \sigma = e_\alpha \text{ and } \delta(x,\alpha) \text{ is defined} \\ undefined & \text{otherwise} \end{cases}$$

We first need to prove a result about $E_\alpha(G)$ that would be useful in the proofs that follow.

Lemma 5.2. *Let* $L' = \{s \in (\Sigma_\alpha^E)^* \mid P_\Sigma(s) \in L(G)$ *and*

$$(\forall t) \text{ if } te_\alpha \leq s \text{ then } P_\Sigma(t)\alpha \in L(G)\ \}$$

Then $L(E_\alpha(G)) = L'$.

Proof. It is sufficient to show: 1) $\mathrm{L}(E_\alpha(G)) \supseteq L'$ and 2) $\mathrm{L}(E_\alpha(G)) \subseteq L'$

Part 1) Show $L' \subseteq L(E_\alpha(G))$

Let s $\in (\Sigma_\alpha^E)^*$ and let $P_\Sigma(s) \in$ L(G). (1)

Assume ($\forall$t) if $te_\alpha \leq s$ then $P_\Sigma(t)\alpha \in L(G)$ (2)

Sufficient to show that s $\in L(E_\alpha(G))$

Since $P_\Sigma(s) \in$ L(G) there exists x such that $\delta(x_0, P_\Sigma(s)) = x$.

We will show by induction on $|s|$ that $\delta_E(x_0, s) = x$ which implies s $\in L(E_\alpha(G))$.

Inductive Base: s $= \epsilon$

Clearly, $\delta(x_0, P_\Sigma(\epsilon)) = \delta(x_0, \epsilon) = x_0 = \delta_E(x_0, \epsilon)$.

Inductive Step: s $= t'\sigma$

As s $\in (\Sigma_\alpha^E)^*$, we have $t' \in (\Sigma_\alpha^E)^*$ and $\sigma \in \Sigma_\alpha^E$.

Since $P_\Sigma(s) \in$ L(G) by (1), hence $\delta(x_0, P_\Sigma(s))!$ and $\delta(x_0, P_\Sigma(t'))!$

Then let $\delta(x_0, P_\Sigma(t')) = x'$

By inductive assumption, $t' \in L(E_\alpha(G))$, $\delta_E(x_0, t')!$, $\delta(x_0, P_\Sigma(t'))!$, and $\delta_E(x_0, t')$ $= \delta(x_0, P_\Sigma(t')) = x'$.

As $\sigma \in \Sigma_\alpha^E = \Sigma \cup \{e_\alpha\}$, we have two cases.

Case i) $\sigma \in \Sigma$

We thus have:

$P_\Sigma(t'\sigma) = P_\Sigma(t')\sigma \in L(G)$ by (1).

This implies there exists x in X such that $\delta(x_0, P_\Sigma(t')) = x'$ and $\delta(x', \sigma) = x$.

Since $\delta_E(x, \sigma) = \delta(x, \sigma)$ for such a case, by the definition of $E_\alpha(G)$, we have:

$\delta_E(x', \sigma) = x$

By inductive assumption $\delta_E(x_0, t') = x'$, and thus: $\delta_E(x_0, t'\sigma) = x$

Case ii) $\sigma = e_\alpha$

We thus have t$e_\alpha \leq$ s and by (2), we have $P_\Sigma(t')\alpha \in L(G)$.

$\Longrightarrow \delta(x', \alpha)!$

$\Longrightarrow \delta_E(x', e_\alpha) = x'$ by the definition of $E_\alpha(G)$.

We thus take x = x', and then, by inductive assumption $\delta_E(x_0, t') = x'$, which gives us:

$$\delta_E(x_0, t'e_\alpha) = x$$

From case i) and ii) we have s $\in L(E_\alpha(G))$.

Part 1 complete.

Part 2) Show L$(E_\alpha(G)) \subseteq L'$

Let s $\in$ L$(E_\alpha(G))$

We will show by induction on $|s|$ that

(a) $\delta(x_0, P_\Sigma(s)) = \delta_E(x_0, s)$ i.e. $P_\Sigma(s) \in$ L(G).

(b) $(\forall t)$ if $te_\alpha \leq$ s then $P_\Sigma(t)\alpha \in$ L(G)

Inductive Base: s $= \epsilon$

(a) Clearly, $\delta(x_0, P_\Sigma(\epsilon)) = \delta_E(x_0, \epsilon) = x_0$.

(b) As s $= \epsilon$ cannot have te_α as its prefix, so we are done.

Inductive Step: s $= t'\sigma$ with $\sigma \in \Sigma_\alpha^E$

Since s $\in$ L$(E_\alpha(G))$, also $t' \in L(E_\alpha(G))$ so $\delta_E(x_0, t') =$ x for some x $\in$ X.

By inductive assumption, we have:

(a) $\delta(x_0, P_\Sigma(t')) = \delta_E(x_0, t') = x$

(b) $(\forall t'')$ if $t''e_\alpha \leq t'$ then $P_\Sigma(t'')\alpha \in$ L(G)

As $\sigma \in \Sigma_\alpha^E = \Sigma \cup \{e_\alpha\}$ we have two cases:

Case i) $\sigma \in \Sigma$

By inductive assumption (a), $\delta(x_0, P_\Sigma(t')) = \delta_E(x_0, t') = x$.

$\Longrightarrow \delta_E(x, \sigma) = \delta(x, \sigma)$, by the definition of $E_\alpha(G)$.

It thus follows that $\delta(x_0, P_\Sigma(t')\sigma) = \delta_E(x_0, t'\sigma)$.

$\Longrightarrow \delta(x_0, P_\Sigma(s)) = \delta_E(x_0, s)$.

We have thus shown condition (a).

We note that (b) follows immediately from the inductive assumption and fact $\sigma \neq e_\alpha$.

Case ii) $\sigma = e_\alpha$

Note that $\delta_E(x, e_\alpha)$ is defined only if $\delta(x, \alpha)$ is defined, by the definitions of $E_\alpha(G)$. As $t'\sigma = t'e_\alpha \in L(E_\alpha(G))$, it follows that $\delta(x, \alpha)!$, $\delta(x_0, P_\Sigma(t')) = \delta_E(x_0, t') = x$, and $\delta_E(x_0, e_\alpha) = x$.

This implies that $\delta(x_0, P_\Sigma(s)) = \delta(x_0, P_\Sigma(t'e_\alpha)) = \delta(x_0, P_\Sigma(t)) = x = \delta_E(x_0, t') = \delta_E(x_0, t'e_\alpha) = \delta_E(x_0, s)$.

We have thus shown condition (a).

To show (b), let t$e_\alpha \leq$ s

If t is a proper prefix of t' it follows from inductive assumption (b) that for all t$e_\alpha \leq$ s, it holds that $P_\Sigma(t)\alpha \in L(G)$.

Otherwise t $= t'$ and since $\delta(x_0, P_\Sigma(t')\alpha)$ is defined, it follows that $P_\Sigma(t)\alpha \in$ L(G).

Part 2 complete.

By part 1 and 2 we have L$(E_\alpha(G)) = L'$ □

Typically we construct our TDES modularly. We thus have G $= G_1 \parallel$ ..... $\parallel G_n = (X, \Sigma, \delta, x_0, X_m)$, with TDES $G_i = (X_i, \Sigma_i, \delta_i, x_{0,i}, X_{m,i})$ for i = 1,...,n. For each G_i, we construct an $E_\alpha(G_i)$ as per Definition 5.2 with each $E_\alpha(G_i)$ sharing the same e_α event. For simplicity, we assume $\alpha \in \Sigma_i$, for all i = 1,...,n. We can simply selfloop α at each state in G_i, without changing L(G). We would then want to construct an $E_\alpha(G)$ modularly as $E_\alpha(G) = E_\alpha(G_1) \parallel$ ..... $\parallel E_\alpha(G_n)$. The Lemma below shows us that our modular $E_\alpha(G)$ gives us the same language as first constructing **G**, and then constructing $E_\alpha(G)$ from **G** using Definition 5.2 directly.

Lemma 5.3. *Let* $G = G_1 \parallel$ ..... $\parallel G_n = (X, \Sigma, \delta, x_0, X_m)$, *with TDES* $G_i = (X_i, \Sigma_i, \delta_i, x_{0,i}, X_{m,i})$ *and* $\alpha \in \Sigma_i$, *i=1,....,n. Then*

$$L(E_\alpha(G)) = L(E_\alpha(G_1) \parallel ..... \parallel E_\alpha(G_n))$$

Proof. Let $P_{\Sigma_i} : (\Sigma_\alpha^E)^* \to \Sigma_i^*$ and $P_i : (\Sigma_\alpha^E)^* \to (\Sigma_i \cup \{e_\alpha\})^*$ be natural projections.

We note that as $\Sigma_i \subseteq \Sigma$, we have $P_{\Sigma_i} \circ P_\Sigma = P_{\Sigma_i}$ (1)

We also note that as $\Sigma_i \subseteq \Sigma_i \cup \{e_\alpha\}$, we have $P_{\Sigma_i} \circ P_i = P_{\Sigma_i}$ (2)

Sufficient to show:

1) $L(E_\alpha(G)) \subseteq L(E_\alpha(G_1) \parallel ..... \parallel E_\alpha(G_n))$ and

2) $L(E_\alpha(G)) \supseteq L(E_\alpha(G_1) \parallel ..... \parallel E_\alpha(G_n))$

Part 1) Show $L(E_\alpha(G)) \subseteq L(E_\alpha(G_1) \parallel ..... \parallel E_\alpha(G_n))$

Let $s \in L(E_\alpha(G))$

Sufficient to show: $(\forall i \in \{1, ..., n\})\ P_i(s) \in L(E_\alpha(G_i)$

Let i $\in \{1, ..., n\}$

By Lemma 5.2, it is sufficient to show that $P_{\Sigma_i}(P_i(s)) \in L(G_i)$ and $(\forall t)$ if $te_\alpha \leq P_i(s)$ then $P_{\Sigma_i}(t)\alpha \in L(G_i)$

As s $\in L(E_\alpha(G))$, we have $P_\Sigma(s) \in L(G)$ by Lemma 5.2.

$\implies P_{\Sigma_i}(P_\Sigma(s)) \in L(G_i)$

$\implies P_{\Sigma_i}(s) \in L(G_i)$ by (1)

$\implies P_{\Sigma_i}(P_i(s)) \in L(G_i)$ by (2)

We will now show $(\forall t)$ if $te_\alpha \leq P_i(s)$ then $P_{\Sigma_i}(t)\alpha \in L(G_i)$

Let $te_\alpha \leq P_i(s)$

$\implies (\exists t' \in P_i^{-1}(t))\ t'e_\alpha \leq s$ (3)

$\implies P_\Sigma(t')\alpha \in L(G)$ by Lemma 5.2

$\implies P_{\Sigma_i}(P_\Sigma(t')\alpha) \in L(G_i)$

$\implies P_{\Sigma_i}(P_\Sigma(t'))\alpha \in L(G_i)$ as $\alpha \in \Sigma_i$

$\implies P_{\Sigma_i}(t')\alpha \in L(G_i)$ by (1)

$\implies P_{\Sigma_i}(P_i(t'))\alpha \in L(G_i)$ by (2)

$\implies P_{\Sigma_i}(t)\alpha \in L(G_i)$, as $P_i(t') = t$ by (3).

Part 1 complete.

Part 2) Show $L(E_\alpha(G)) \supseteq L(E_\alpha(G_1) \parallel ..... \parallel E_\alpha(G_n))$

Let $s \in L(E_\alpha(G_1) \parallel ..... \parallel E_\alpha(G_n))$

By Lemma 5.2 it is sufficient to show that $P_\Sigma(s) \in L(G)$ and $(\forall t)$ if $te_\alpha \leq s$ then $P_\Sigma(t)\alpha \in L(G)$

To show $P_\Sigma(s) \in$ L(G) it is sufficient to show:

$(\forall i \in \{1, ..., n\})P_{\Sigma_i}(P_\Sigma(s)) \in L(G_i)$

Let i $\in \{1, ...n\}$

Sufficient to show $P_{\Sigma_i}(s) \in L(G_i)$, by (1)

From (4), we have $P_i(s) \in L(E_\alpha(G_i))$

By Lemma 5.2, we have $P_{\Sigma_i}(P_i(s)) \in L(G_i)$

$\implies P_{\Sigma_i}(s) \in L(G_i)$ by (2)

We thus have $P_\Sigma(s) \in L(G)$

We will now show: $(\forall t)$ if $te_\alpha \leq s$ then $P_\Sigma(t)\alpha \in L(G)$

Let $te_\alpha \leq s$ (5)

To show $P_\Sigma(t)\alpha \in$ L(G), it is sufficient to show:

$(\forall i \in \{1, ..., n\})P_{\Sigma_i}(P_\Sigma(t)\alpha) \in L(G_i)$

Let i $\in \{1, ..., n\}$

We note that $P_{\Sigma_i}(P_\Sigma(t)\alpha) = P_{\Sigma_i}(P_\Sigma(t))\alpha = P_{\Sigma_i}(t)\alpha$ as $\alpha \in \Sigma_i$, and by (1).

It is sufficient to show $P_{\Sigma_i}(t)\alpha \in L(G_i)$

By (5), we have $te_\alpha \leq s$

$\implies P_i(te_\alpha) = P_i(t)e_\alpha \leq P_i(s)$

From (4), we have $P_i(s) \in L(E_\alpha(G_i))$

By Lemma 5.2, we thus have $P_{\Sigma_i}(P_i(t))\alpha \in L(G_i)$

$\implies P_{\Sigma_i}(t)\alpha \in L(G_i)$ by (2)

We thus conclude that $s \in L(E_\alpha(G))$

Part 2 complete.

By Part 1 and 2 we conclude

$L(E_\alpha(G)) = L(E_\alpha(G_1) \parallel ..... \parallel L(E_\alpha(G_n))$ □

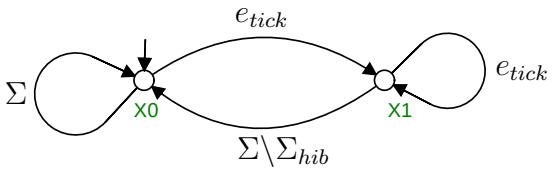


Figure 5.1: TDES T_Σ^E

Finally, we need to construct TDES T_Σ^E as shown in Figure 5.1. TDES T_Σ^E captures the idea that once *tick* is possible, it can't be followed by a prohibitable event. We note that the alphabet of T_Σ^E is $\Sigma_{tick}^E = \Sigma \cup \{e_{tick}\}$.

We now need to prove a result about T_Σ^E that we will need later in this section.

Lemma 5.4. *Let* $L' = \{s \in (\Sigma_{tick}^E)^* \mid$
$(\forall t)\ (\forall \alpha)$ *if* $te_{tick}\alpha \leq s$ *then* $\alpha \notin \Sigma_{hib}\}$
Then $L(T_\Sigma^E) = L'$

Proof. Let δ_T be the transition function for T_Σ^E.

It is sufficient to show: 1) L(T_Σ^E) $\supseteq L'$ and 2) L(T_Σ^E) $\subseteq L'$

Part 1) Show $L' \subseteq L(T_\Sigma^E)$

Let s $\in L'$.

Assume $(\forall t)\ (\forall \alpha)\ te_{tick}\alpha \leq s \Rightarrow \alpha \notin \Sigma_{hib}$ (1)

We show by induction on n = $|s|$ that the following holds:

a) If s = ue_{tick} for some u then δ_T(X0,s)=X1

b) If s $\neq$ ue_{tick} for all u then δ_T(X0,s)=X0

Inductive Base: n=0, s = ϵ

$\Longrightarrow$ s $\neq$ ue_{tick} for all u

So it is enough to show that δ_T(X0,s)=X0

Clearly δ_T(X0,ϵ)=X0.

Inductive Step: n $\rightarrow$ n+1 , s = $t'\sigma$ with $t' \in (\Sigma^E_{tick})^*$, $\mid t' \mid$ = n, and $\sigma \in \Sigma^E_{tick}$

By inductive assumption t' satisfies (a) and (b).

This implies $t' \in L(T^E_\Sigma)$ (1)

Consider the following cases:

Case i) $\sigma = e_{tick}$

Then s = ue_{tick} for u = t', so it is enough to show δ_T(X0,s)=X1

Since $t' \in L(T^E_\Sigma)$ by (1), it follows by the construction of T^E_Σ that δ_T(X0,s) = X1 as either $\delta_T(X0, t')$ = X0 or $\delta_T(X0, t')$ = X1, but in both cases the next state is X1.

Case ii) $\sigma \in \Sigma$

$\Longrightarrow$ s $\neq$ ue_{tick} for all u.

It is thus enough to show δ_T(X0,s) = X0

Since $t' \in L(T^E_\Sigma)$ by (1), there are two cases.

Case ii.a) $t' = t''e_{tick}$

By inductive assumption (a), δ_T(X0,t') = X1.

Since $\sigma \in \Sigma$, this means $t''e_{tick}\sigma \leq$ s $\in L'$.

This implies $\sigma \notin \Sigma_{hib}$ by definition of L'.

Then by construction of T^E_Σ, δ_T(X1,σ) = X0 for $\sigma \in \Sigma \backslash \Sigma_{hib}$.

Thus δ_T(X0,s) = X0

Case ii.b) $t' \neq t''e_{tick}$

Then by inductive assumption (b), δ_T(X0,t')=X0.

Then by the construction of T^E_Σ, δ_T(X0,σ) = X0 for $\sigma \in \Sigma$.

$\Longrightarrow \delta_T$(X0,s) = X0

Now we have shown for all s $\in$ L(T^E_Σ) that δ_T(X0,s) = X0 or δ_T(X0,s)=X1

$\Longrightarrow s \in L(T^E_\Sigma)$

Part 1 complete.

Part 2) Show L$(T_\Sigma^E) \subseteq L'$

Let s $\in L(T_\Sigma^E)$.

Show $te_{tick}\alpha \leq s$ implies $\alpha \notin \Sigma_{hib}$

Assume $te_{tick}\alpha \leq$ s.

Clearly by the construction on T_Σ^E, $\delta_T(X0, te_{tick}) = X_1$ as $te_{tick} \leq$ s $\in L(T_\Sigma^E)$.

As $\delta_T(X_1, \sigma)$ is undefined for $\sigma \in \Sigma_{hib}$, we have:

$te_{tick}\alpha \leq s \in L(T_\Sigma^E) \Longrightarrow \alpha \notin \Sigma_{hib}$

Part 2 complete.

From part 1 and 2 we conclude that L$(T_\Sigma^E) = L'$ □

Now we present a proposition which we will need for our main theorem later in this section. The proof is based on an initial proof sketch by the author's thesis supervisors, Dr. R. Malik and Dr. R. Leduc.

Proposition 5.1. Let TDES G $= (Q, \Sigma, \delta, q_0, Q_m)$ and TDES S $= (X, \Sigma, \xi, x_0, X_m)$. G and S satisfy SD controllability point (ii.a) iff

$$L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E)$$

Proof. Show

I) G and S satisfy SD controllability point (ii.a) $\Longrightarrow$

$$L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E)$$

II) $L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E) \Longrightarrow$

$[(\forall s \in L(S) \cap L(G))s\ tick \in L(G) \cap L(S) \Rightarrow Elig_{L(G)\cap L(S)}(s) \cap \Sigma_{hib} = \emptyset]$

Part I) Show G and S satisfy SD controllability point (ii.a) $\Rightarrow$

$L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E)$

Let G, S be TDES that satisfy SD controllability point (ii.a)

Let $s \in L(E_{tick}(G)) \cap L(E_{tick}(S))$

$\Longrightarrow s \in (\Sigma_{tick}^E)^*$ by Lemma 5.2

Let $t \in (\Sigma_{tick}^E)^*, \alpha \in \Sigma_{tick}^E$ such that $te_{tick}\alpha \leq s$

Sufficient to show $\alpha \notin \Sigma_{hib}$ by Lemma 5.4

Since $te_{tick} \leq te_{tick}\alpha \leq s \in L(E_{tick}(G))$, we have $P_\Sigma(t)tick \in L(G)$ by Lemma 5.2

Similarly, $P_\Sigma(t)tick \in L(S)$

$\Longrightarrow P_\Sigma(t)tick \in L(G) \cap L(S)$

Since G and S satisfy SD controllability point (ii.a), we have

$$Elig_{L(G)\cap L(S)}(P_\Sigma(t)) \cap \Sigma_{hib} = \emptyset \tag{1}$$

Also $te_{tick}\alpha \leq s \in L(E_{tick}(G))$

$\Longrightarrow P_\Sigma(t\alpha) = P_\Sigma(te_{tick}\alpha) \in L(G)$ by Lemma 5.2

Similarly, $P_\Sigma(t\alpha) \in L(S)$

If $\alpha = e_{tick}$ then $\alpha \notin \Sigma_{hib}$ and we are done. We can thus assume $P_\Sigma(\alpha) = \alpha$ without any loss of generality.

$\Longrightarrow P_\Sigma(t)\alpha = P_\Sigma(t\alpha) \in L(G) \cap L(S)$

which means that $\alpha \in Elig_{L(G)\cap L(S)}(P_\Sigma(t))$

Then from (1), we have $\alpha \notin \Sigma_{hib}$

Part I complete.

Part II) Show $L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E) \Longrightarrow$

$[(\forall s \in L(S) \cap L(G))s\ tick \in L(G) \cap L(S) \Rightarrow Elig_{L(G)\cap L(S)}(s) \cap \Sigma_{hib} = \emptyset]$

Assume $L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E)$

Let $s \in L(S) \cap L(G)$ and assume s tick $\in L(G) \cap L(S)$

Let $\alpha \in Elig_{L(G)\cap L(S)}(s)$.

$\Rightarrow s\alpha \in L(G) \cap L(S)$

Sufficient to show that $\alpha \notin \Sigma_{hib}$

First we note that $se_{tick}\alpha \in L(E_{tick}(G))$ by Lemma 5.2 because $P_\Sigma(se_{tick}\alpha) = s\alpha \in L(G)$ and $P_\Sigma(s)tick = s\ tick \in L(G)$

Likewise, $se_{tick}\alpha \in L(E_{tick}(S))$

Therefore $se_{tick}\alpha \in L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E)$

Then $\alpha \notin \Sigma_{hib}$ by Lemma 5.4

Part II complete.

By part I and II we conclude that G and S satisfy SD controllability point (ii.a) iff $L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_{\Sigma}^{E})$ □

5.1.2 SD Controllability Point (ii.b)

We will now express this property as a nonblocking problem. Our first step is to express as event occurrences whether *tick* is disabled at a given state in our TDES plant.

Definition 5.3. Let TDES G = $(X, \Sigma, \delta, x_0, X_m)$, $\alpha \in \Sigma$, $d_{\alpha,G} \notin \Sigma$ and $\Sigma_{\alpha,G}^{D} = \Sigma \cup \{d_{\alpha,G}\}$. Let $P_{\Sigma} : (\Sigma_{\alpha,G}^{D})^{*} \rightarrow \Sigma^{*}$ be a natural projection. Construct TDES $D_{\alpha}(G)$ from **G**, by adding a $d_{\alpha,G}$ selfloop at each state in **G** that α is disabled at. Let $D_{\alpha}(G) = (X, \Sigma \cup \{d_{\alpha,G}\}, \delta_D, x_0, X_m)$ and for $x \in$ X

$$\delta_D(x,\sigma) = \begin{cases} \delta(x,\sigma) & \text{if } \sigma \in \Sigma \text{ and } \delta(x,\sigma) \text{ is defined} \\ x & \text{if } \sigma = d_{\alpha,G} \text{ and } \delta(x,\alpha) \text{ is not defined} \\ undefined & \text{otherwise} \end{cases}$$

We first need to prove a result about $D_{\alpha}(G)$ that we will need later in other proofs.

Lemma 5.5. *Let* $L' = \{s \in (\Sigma_{\alpha,G}^{D})^{*} \mid P_{\Sigma}(s) \in L(G)$ *and*
$(\forall t)$ *if* $td_{\alpha,G} \leq s$ *then* $P_{\Sigma}(t)\alpha \notin L(G)\}$
Then $L(D_{\alpha}(G)) = L'$

Proof. It is sufficient to show: 1) L$(D_{\alpha}(G)) \supseteq L'$ and 2) L$(D_{\alpha}(G)) \subseteq L'$

Part 1) Show $L' \subseteq L(D_{\alpha}(G))$

Let s $\in (\Sigma_{\alpha,G}^{D})^{*}$ and $P_{\Sigma}(s) \in$ L(G). (1)

Assume ($\forall$t) if $td_{\alpha,G} \leq s$ then $P_{\Sigma}(t)\alpha \notin L(G)$ (2)

Sufficient to show that s $\in L(D_\alpha(G))$

Since $P_\Sigma(s) \in$ L(G), this implies $\delta(x_0, P_\Sigma(s))!$.

We will show by induction on $|s|$ that $\delta_D(x_0, s) = \delta(x_0, P_\Sigma(s))$. This will imply s $\in L(D_\alpha(G))$.

Inductive Base: s $= \epsilon$

Clearly $\delta(x_0, P_\Sigma(\epsilon)) = \delta(x_0, \epsilon) = x_0 = \delta_D(x_0, \epsilon)$

Inductive Step: s $= t'\sigma$

As s $\in (\Sigma^D_{\alpha,G})^*$, we have $t' \in (\Sigma^D_{\alpha,G})^*$ and $\sigma \in \Sigma^D_{\alpha,G}$.

Since $P_\Sigma(s) \in$ L(G) by (1), we have $\delta(x_0, P_\Sigma(s))!$ and $\delta(x_0, P_\Sigma(t'))!$.

Let $\delta(x_0, P_\Sigma(t')) = x'$.

By inductive assumption, $t' \in L(D_\alpha(G))$, $\delta_D(x_0, t')!$, and $\delta_D(x_0, t') = \delta(x_0, P_\Sigma(t'))$ $= x'$

As $\sigma \in \Sigma^D_{\alpha,G} = \Sigma \cup \{d_{\alpha,G}\}$, we have two cases:

case i) $\sigma \in \Sigma$

$\Longrightarrow P_\Sigma(t')\sigma = P_\Sigma(t'\sigma) = P_\Sigma(s) \in L(G)$ by (1)

This implies there exists x in X such that $\delta(x_0, P_\Sigma(t')) = x'$ and $\delta(x', \sigma) = x$.

Since $\delta_D(x, \sigma) = \delta(x, \sigma)$ by the definition of $D_\alpha(G)$, we have:

$\delta_D(x', \sigma) = x$

By inductive assumption $\delta_D(x_0, t') = x'$, which implies:

$\delta_D(x_0, t'\sigma) = \delta_D(x_0, s) = x$

case ii) $\sigma = d_{\alpha,G}$

This implies $t'd_{\alpha,G} \leq$ s and it follows by (2) that $P_\Sigma(t')\alpha \notin L(G)$

$\Longrightarrow \delta(x', \alpha)$ is not defined.

$\Longrightarrow \delta_D(x', d_{\alpha,G}) = x'$ by the definition of $D_\alpha(G)$

By inductive assumption, $\delta_D(x_0, t') = \delta(x_0, P_\Sigma(t')) = x'$. We thus have:

$\delta_D(x', d_{\alpha,G}) = \delta_D(x_0, t'd_{\alpha,G}) = x' = \delta(x_0, P_\Sigma(t'd_{\alpha,G})) = \delta(x_0, P_\Sigma(t'))$

From case i) and ii) we can conclude that s $\in L(D_\alpha(G))$

Part 1 complete.

Part 2) Show L($D_\alpha(G)$) $\subseteq L'$

Let s $\in$ L($D_\alpha(G)$)

We will show by induction on $|s|$ that

(a) $\delta(x_0, P_\Sigma(s)) = \delta_D(x_0, s)$ i.e. $P_\Sigma(s) \in$ L(G)

(b) ($\forall$t) if t$d_{\alpha,G} \leq$ s then $P_\Sigma(t)\alpha \notin$ L(G)

Inductive Base: s = ϵ

Then,

(a) clearly $\delta(x_0, P_\Sigma(\epsilon)) = \delta_D(x_0, s) = x_0$

(b) As s = ϵ cannot have t$d_{\alpha,G}$ as its prefix, therefore we are done.

Inductive Step: s = $t'\sigma$ with $\sigma \in \Sigma^D_{\alpha,G}$

Since s $\in$ L($D_\alpha(G)$), we have $t' \in L(D_\alpha(G))$ so $\delta_D(x_0, t')$ = x for some x $\in$ X.

By inductive assumption, we have:

(a) $\delta(x_0, P_\Sigma(t')) = \delta_D(x_0, t') = x$

(b) ($\forall t''$) if $t''d_{\alpha,G} \leq t'$ then $P_\Sigma(t'')\alpha \notin L(G)$

As $\sigma \in \Sigma^D_{\alpha,G} = \Sigma \cup \{d_{\alpha,G}\}$, we have two cases:

case i) $\sigma \in \Sigma$

By inductive assumption (a), $\delta(x_0, P_\Sigma(t')) = \delta_D(x_0, t') = x$. It thus follows that:

$\delta_D(x, \sigma) = \delta(x, \sigma)$, by the definition of $D_\alpha(G)$.

$\implies \delta(x_0, P_\Sigma(t')\sigma) = \delta(x_0, P_\Sigma(t'\sigma)) = \delta_D(x_0, t'\sigma)$

$\implies \delta(x_0, P_\Sigma(s)) = \delta_D(x_0, s)$

We have now shown condition (a).

We note that (b) follows immediately from the inductive assumption and fact $\sigma \neq d_{\alpha,G}$.

case ii) if $\sigma = d_{\alpha,G}$

Note $\delta_D(x, d_{\alpha,G})$ is only defined if $\delta(x, \alpha)$ is undefined, by the definition of $D_\alpha(G)$.

By inductive assumption, it follows that $\delta(x, \alpha)$ is not defined.

$\implies \delta_D(x, d_{\alpha,G}) = \text{x}$

This implies $\delta(x_0, P_\Sigma(s)) = \delta(x_0, P_\Sigma(t'd_{\alpha,G})) = \delta(x_0, P_\Sigma(t')) = \text{x} = \delta_D(x_0, t') = \delta_D(x_0, t'd_{\alpha,G}) = \delta_D(x_0, s)$

We have now shown condition (a).

We now have to show (b).

Let $\text{t}d_{\alpha,G} \leq \text{s}$.

If t is a proper prefix of t', it follows from inductive assumption (b) that for all $\text{t}d_{\alpha,G} \leq \text{s}$ it holds that $P_\Sigma(t)\alpha \notin \text{L(G)}$.

Otherwise $\text{t} = t'$ and since $\delta(x_0, P_\Sigma(t')\alpha)$ is not defined it also follows that $P_\Sigma(t)\alpha \notin L(G)$.

Part 2 complete.

By 1 and 2 we have $\text{L}(D_\alpha(G)) = L'$ □

Typically, we construct our TDES modularly. We thus have $\text{G} = G_1 \parallel .... \parallel G_n = (X, \Sigma, \delta, x_0, X_m)$, with $G_i = (X_i, \Sigma_i, \delta_i, x_{0,i}, X_{m,i})$ for i = 1,...,n. For each G_i, we construct a $D_\alpha(G_i)$ using **Definition 5.3**, with each G_i having their own d_{α,G_i} event. Note that if $\alpha \notin \Sigma_i$, then α is effectively selflooped at every state in G_i, so d_{α,G_i} will never occur in $D_\alpha(G_i)$.

We then construct $D_\alpha(G) = D_\alpha(G_1) \parallel .... \parallel D_\alpha(G_n)$ modularly as well. We note that $\Sigma^D_{\alpha,G} = \Sigma \dot\cup \{d_{\alpha,G_1}, ..., d_{\alpha,G_n}\}, D_{\alpha,G} = \{d_{\alpha,G_1}, ..., d_{\alpha,G_n}\}$. Let $P_\Sigma : (\Sigma^D_{\alpha,G})^* \to \Sigma^*$ be a natural projection.

We now present a lemma that is analogous to Lemma 5.5, but for a modularly constructed $D_\alpha(G)$.

Lemma 5.6. *Let* $G = G_1 \parallel .... \parallel G_n = (X, \Sigma, \delta, x_0, X_m)$ *with* $G_i = (X_i, \Sigma_i, \delta_i, x_{0,i}, X_{m,i})$ *for i = 1,...,n. Let* $D_\alpha(G) = D_\alpha(G_1) \parallel .... \parallel D_\alpha(G_n)$. *Let* $L' = \{s \in (\Sigma^D_{\alpha,G})^* \mid P_\Sigma(s) \in L(G)$ *and*

$$(\forall t)\ (\forall d \in D_{\alpha,G})\ \textit{if}\ td \leq s\ \textit{then}\ P_\Sigma(t)\alpha \notin L(G)\}$$

Then $L(D_\alpha(G)) = L'$

Proof. Let $P_{\Sigma_i} : (\Sigma^D_{\alpha,G})^* \to \Sigma_i^*$, and $P_i : (\Sigma^D_{\alpha,G})^* \to (\Sigma_i \cup \{d_{\alpha,G_i}\})^*$ be natural projections for i = 1,....,n.

We note that as $\Sigma_i \subseteq \Sigma, P_{\Sigma_i} \circ P_\Sigma = P_{\Sigma_i}$ (1)

We also note that as $\Sigma_i \subseteq \Sigma_i \cup \{d_{\alpha,G_i}\}, P_{\Sigma_i} \circ P_i = P_{\Sigma_i}$ (2)

It is sufficient to show: 1) L($D_\alpha(G)$) $\subseteq L'$ and 2) L($D_\alpha(G)$) $\supseteq L'$

Part 1) Show L($D_\alpha(G)$) $\subseteq L'$

Let s $\in L(D_\alpha(G))$

Sufficient to show $P_\Sigma(s) \in L(G)$ and $(\forall t)(\forall d \in D_\alpha(G))$ if $td \leq s$ then $P_\Sigma(t)\alpha \notin L(G)$

We start by showing $P_\Sigma(s) \in L(G)$

As G = $G_1 \parallel ..... \parallel G_n$, it is sufficient to show:

$(\forall i \in \{1, ..., n\})P_{\Sigma_i}(P_\Sigma(s)) \in L(G_i)$.

Let i $\in \{1, ..., n\}$

As $P_{\Sigma_i} \circ P_\Sigma = P_{\Sigma_i}$ by (1), it is sufficient to show $P_{\Sigma_i}(s) \in L(G_i)$

As $s \in L(D_\alpha(G))$, we have $P_i(s) \in L(D_\alpha(G_i))$

By Lemma 5.5, we have $P_{\Sigma_i}(P_i(s)) \in L(G_i)$

$\Longrightarrow P_{\Sigma_i}(s) \in L(G_i)$ by (2)

We thus have $P_\Sigma(s) \in L(G)$

We now show $(\forall t)(\forall d \in D_{\alpha,G})$ if $td \leq s$ then $P_\Sigma(t)\alpha \notin L(G)$

Let $d \in D_{\alpha,G}$ and $td \leq s$

As $d \in D_{\alpha,G}$, we have $(\exists i \in \{1, ...n\})\ d = d_{\alpha,G_i}$.

Let i be this index.

As $s \in L(D_\alpha(G))$, we have $P_i(s) \in L(D_\alpha(G_i))$

By Lemma 5.5 and (2), we have $P_{\Sigma_i}(P_i(s)) = P_{\Sigma_i}(s) \in L(G_i)$

As $td_{\alpha,G_i} \leq s$, we have $P_i(td_{\alpha,G_i}) = P_i(t)d_{\alpha,G_i} \leq P_i(s)$

By Lemma 5.5, we can conclude $P_{\Sigma_i}(P_i(t))\alpha \notin L(G_i)$

$\Longrightarrow P_{\Sigma_i}(t)\alpha \notin L(G_i)$ by (2)

As $P_i(s) \in L(D_\alpha(G_i))$, and $P_i(t)d_{\alpha,G_i} \leq P_i(s)$, this implies $\alpha \in \Sigma_i$, as otherwise d_{α,G_i} would never occur in $D_\alpha(G_i)$.

$\implies P_{\Sigma_i}(t)\alpha = P_{\Sigma_i}(t\alpha) \notin L(G_i)$

$\implies t\alpha \notin L(G)$

Part 1 complete.

Part 2) Show $L' \subseteq L(D_\alpha(G))$

Let $s \in L'$ (3)

Sufficient to show $(\forall i \in \{1, ...n\})P_i(s) \in L(D_\alpha(G_i))$

Let i $\in \{1, ...n\}$

By Lemma 5.5 and (2), it is sufficient to show that

$P_{\Sigma_i}(P_i(s)) = P_{\Sigma_i}(s) \in L(G_i)$ and $(\forall t)$ if $td_{\alpha,G_i} \leq P_i(s)$ then $P_{\Sigma_i}(t)\alpha \notin L(G_i)$

We first show $P_{\Sigma_i}(s) \in L(G_i)$

By (3) and definition of L', we have $P_\Sigma(s) \in L(G)$

$\implies P_{\Sigma_i}(P_\Sigma(s)) \in L(G_i)$

$\implies P_{\Sigma_i}(s) \in L(G)$ by (1)

We now show $(\forall t)$ if $td_{\alpha,G_i} \leq P_i(s)$ then $P_{\Sigma_i}(t)\alpha \notin L(G_i)$

We note that we have two cases:

If $\alpha \notin \Sigma_i$, then this is automatically true.

We can thus assume $\alpha \in \Sigma_i$ without any loss of generality.

Let $td_{\alpha,G_i} \leq P_i(s)$

$\implies (\exists t' \in P_i^{-1}(t))\ t'd_{\alpha,G_i} \leq s$ (4)

$\implies P_\Sigma(t')\alpha \notin L(G)$ by (3) and definition of L'

$\implies P_{\Sigma_i}(P_\Sigma(t')\alpha) \notin L(G_i)$

$\implies P_{\Sigma_i}(t'\alpha) \notin L(G_i)$ by (1)

$\implies P_{\Sigma_i}(t')\alpha \notin L(G_i)$, as $\alpha \in \Sigma_i$

$\implies P_{\Sigma_i}(P_i(t'))\alpha \notin L(G_i)$ by (2)

$\implies P_{\Sigma_i}(t)\alpha \notin L(G)$ as $P_i(t') = t$ by (4)

Part 2 complete.

By 1 and 2 we have $\mathrm{L}(D_\alpha(G)) = L'$ $\square$

We will now express the SD controllability point (ii.b) property as a Υ-enabling problem (see page 38) with $\Upsilon = D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}$. We can then use theorem 4.3 (see page 39) to express Υ-enabling property as a nonblocking check.

Now we present a proposition which we will need for our main theorem later in this section. TDES $D_{tick}(G)$ is constructed using the modular $D_\alpha(G)$ definition with $\alpha = tick$. The proof is based on an initial proof sketch by the author's thesis supervisors, Dr. R. Malik and Dr. R. Leduc.

Proposition 5.2. Let G $= G_1 \parallel .... \parallel G_n = (X, \Sigma, \delta, x_0, X_m)$ with $G_i = (X_i, \Sigma_i, \delta_i, x_{0,i}, X_{m,i})$, i= 1,...,n. Let S $= (Y, \Sigma, \xi, y_0, Y_m)$. G and S satisfy SD controllability part (ii.b) iff

$D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling.$

Proof. To prove this claim it is sufficient to show that

I) If G and S satisfy SD controllability part (ii.b) then

$D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$ and

II) If $D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$, then

$(\forall s \in L(S) \cap L(G))$ s tick $\in L(G) \Longrightarrow Elig_{L(G)\cap L(S)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset$

Let $P_{\Sigma_i} : (\Sigma^D_{\alpha,G})^* \rightarrow \Sigma_i^*$ and $P_i : (\Sigma_{\alpha,G})^* \rightarrow (\Sigma_i \cup \{d_{\alpha,G_i}\})^*$ be natural projections, i = 1,...n.

We note that as $\Sigma_i \subseteq \Sigma$, we have $P_{\Sigma_i} \circ P_\Sigma = P_{\Sigma_i}$ (1)

We also note that as $\Sigma_i \subseteq \Sigma_i \cup \{d_{\alpha,G_i}\}$, we have $P_{\Sigma_i} \circ P_i = P_{\Sigma_i}$ (2)

Part I) Show if G and S satisfy SD controllability part (ii.b) then

$D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling.$

Let G, S be TDES that satisfy SD controllability point (ii.b).

Let $s \in L(D_{tick}(G) \parallel S)$

$\Longrightarrow s \in L(D_{tick}(G))$ as G and S are both over Σ (3)

$\Longrightarrow P_{\Sigma}(s) \in L(G)$ by Lemma 5.6 (4)

We need to show that:

$(\exists \alpha \in D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) s\alpha \in L(D_{tick}(G) \parallel S)$

Consider two cases:

Case a) $P_{\Sigma}(s)tick \in L(G)$

By SD controllability point (ii.b) and (4), we have

$Elig_{L(G)\cap L(S)}(P_{\Sigma}(s)) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset$

i.e there exists $\alpha \in \Sigma_{hib} \cup \{tick\}$ such that $P_{\Sigma}(s)\alpha \in L(G) \cap L(S)$

Note $P_{\Sigma}(s\alpha) = P_{\Sigma}(s)\alpha \in L(G)$ as $\Sigma_{hib} \cup \{tick\} \subseteq \Sigma$

$\Longrightarrow s\alpha \in L(D_{tick}(G))$ by Lemma 5.6 since $s \in L(D_{tick}(G))$ by (3), $\alpha \notin D_{tick,G}$ and $P_{\Sigma}(s\alpha) \in L(G)$.

$\Rightarrow s\alpha \in L(D_{tick}(G) \parallel S)$ with $\alpha \in \Sigma_{hib} \cup \{tick\} \subseteq D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}$

Case b) $P_{\Sigma}(s)tick \notin L(G)$

We have $P_{\Sigma}(s) \in L(G)$ by (2)

$\Rightarrow tick \notin Elig_{L(G)}(P_{\Sigma}(s))$

$\Rightarrow (\exists i \in \{1....n\}) tick \notin Elig_{L(G_i)}(P_{\Sigma_i}(P_{\Sigma}(s)))$

i.e. $P_{\Sigma_i}(P_{\Sigma}(s))tick \notin L(G_i)$

$\Longrightarrow P_{\Sigma_i}(s)tick \notin L(G_i)$ by (1)

We note that $P_{\Sigma}(s) \in L(G)$ implies by (1):

$P_{\Sigma_i}(P_{\Sigma}(s)) = P_{\Sigma_i}(s) \in L(G_i)$

We next note that: $s \in L(D_{tick}(G)) \Rightarrow P_i(s) \in L(D_{tick}(G_i))$

Also, we have $P_{\Sigma_i}(P_i(s)) = P_{\Sigma_i}(s) \in L(G_i)$, by (2).

As $P_{\Sigma_i}(P_i(s))tick = P_{\Sigma_i}(s)tick \notin L(G_i)$, we can conlcude by Definition 5.3:

$P_i(s)d_{tick,G_i} \in L(D_{tick}(G_i))$

Also for $j \neq i$, $P_j(sd_{tick,G_i}) = P_j(s) \in L(D_{tick}(G_j))$ because $s \in L(D_{tick}(G))$

Therefore $sd_{tick,G_i} \in L(D_{tick}(G))$

Also $P_\Sigma(sd_{tick,G_i}) = s \in L(S)$ and thus $sd_{tick,G_i} \in L(D_{tick}(G) \parallel S)$ with $d_{tick,G_i} \in D_{tick,G} \subseteq D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}$.

Part I complete.

Part II) Show: If $D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$, then $(\forall s \in L(G) \cap L(S))$ s $tick \in L(G) \Longrightarrow Elig_{L(G)\cap L(S)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset$

Let $D_{tick}(G) \parallel S$ be $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$.

Let $s \in L(G) \cap L(S), s\ tick \in L(G)$

Need to show:

$(\exists \alpha \in \Sigma_{hib} \cup \{tick\}) s\alpha \in L(G) \cap L(S)$

Since $s \in L(G) \cap L(S)$ and $P_\Sigma(s) = s \in L(G)$, we have by Lemma 5.6:

$s \in L(D_{tick}(G))$

$\Longrightarrow s \in L(D_{tick}(G) \parallel S)$

Since $D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$, we have

$(\exists \alpha \in D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) s\alpha \in L(D_{tick}(G) \parallel S)$

We note for any $i \in \{1....n\}$ that:

s tick $\in L(G) \Rightarrow P_{\Sigma_i}$(s tick) $= P_{\Sigma_i}(s)tick \in L(G_i)$

$\Longrightarrow P_i(s)d_{tick,G_i} \notin L(D_{tick}(G_i))$ by Lemma 5.5

$\Longrightarrow P_i(sd_{tick,G_i}) \notin L(D_{tick}(G_i))$

$\Longrightarrow sd_{tick,G_i} \notin L(D_{tick}(G))$

$\Longrightarrow sd_{tick,G_i} \notin L(D_{tick}(G) \parallel S)$

Given $s\alpha \in L(D_{tick}(G) \parallel S)$ it follows that $\alpha \neq d_{tick,G_i}$ for i = 1,....,n.

We thus have $\alpha \notin D_{tick,G}$, and $\alpha \in \Sigma_{hib} \cup \{tick\}$ with $s\alpha \in L(D_{tick}(G) \parallel S)$.

$\Longrightarrow s\alpha \in L(S)$ as $P_\Sigma(s\alpha) = s\alpha$

We also have $s\alpha \in L(D_{tick,G})$, as $s\alpha \in L(D_{tick}(G) \parallel S)$ and $\Sigma^D_{\alpha,G} \supseteq \Sigma$.

$\Longrightarrow P_\Sigma(s\alpha) = s\alpha \in L(G)$, by Lemma 5.6

We thus have $\alpha \in \Sigma_{hib} \cup \{tick\}$ with $s\alpha \in L(G) \cap L(S)$ as required.

Part II complete.

By part I and II we conclude that G and S satisfy SD controllability point (ii.b) iff $D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$. □

We will now prove that checking SD controllability point (ii) is equivalent to checking the language inclusion and Υ-enabling property below.

Theorem 5.1. *Let* $G = G_1 \parallel .... \parallel G_n = (X, \Sigma, \delta, x_0, X_m)$ *with* $G_i = (X_i, \Sigma_i, \delta_i, x_{0,i}, X_{m,i})$, *i= 1,...,n. Let* $S = (Y, \Sigma, \xi, y_0, Y_m)$. *G and S satisfy SD Controllability point (ii) iff*

$$L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E) \text{ and } D_{tick}(G) \parallel S \text{ is } (D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling.$$

Proof. It is sufficient to show:

I) G and S satisfy SD controllability point (ii) $\Longrightarrow L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E)$ and $D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$.

II) $D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$ and $L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E) \Longrightarrow$ G and S satisfy SD controllability point (ii)

Part I) Let G and S satisfy SD controllability point (ii)

By proposition 5.1 we have $L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E)$

By proposition 5.2, we have $D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$

Part I complete.

Part II) Let $D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$ and $L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E)$

By proposition 5.1 we have $L(E_{tick}(G)) \cap L(E_{tick}(S)) \subseteq L(T_\Sigma^E)$

$\Longrightarrow [(\forall s \in L(S) \cap L(G)) s\ tick \in L(G) \cap L(S) Elig_{L(G) \cap L(S)}(s) \cap \Sigma_{hib} = \emptyset]$ which implies (ii.a).

By proposition 5.2 we have $D_{tick}(G) \parallel S$ is $(D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}) - enabling$

$\Longrightarrow [(\forall s \in L(S) \cap L(G))$ s tick $\in L(G) \Longrightarrow Elig_{L(G)\cap L(S)}(s) \cap (\{tick\} \cup \Sigma_{hib}) \neq \emptyset]$ which implies (ii.b).

$\Longrightarrow$ G and S satisfy SD controllability point (ii) by Lemma 5.1.

Part II complete.

□

5.2 SD Controllability Point (iii)

SD controllability point (iii) deals with sampled strings and is divided into two parts, point (iii.1) and point (iii.2).

5.2.1 SD Controllability Point (iii.1)

The SD controllability point (iii.1) property (see page 23) is defined in terms of the closed-loop system. For simplicity, we restate the property below in terms of some TDES **G**, which would correspond to our closed-loop system.

Definition 5.4. Let G $=(Q, \Sigma, \delta, q_0, Q_m)$ be a TDES. G satisfies SD controllability point (iii.1) if

$$(\forall s \in L(G) \cap L_{samp})(\forall t \in \Sigma^*_{act}) st \in L(G) \Longrightarrow$$
$$(Elig_{L(G)}(st) \cup Occu(t)) \cap \Sigma_{hib} = Elig_{L(G)}(s) \cap \Sigma_{hib}$$

This point states that if a prohibitable event is possible in a given clock period, it must be possible immediately after the *tick* and stay possible until it occurs. It also says that no new prohibitable events may become possible until after the next clock *tick*. For the compositional verification, we can separate the set equality condition into two parts, $\subseteq$ and $\supseteq$. They are defined as:

iii.1a) $(\forall s \in L(G) \cap L_{samp})(\forall t \in \Sigma^*_{act}) st \in L(G) \Longrightarrow$
$$Elig_{L(G)}(s) \cap \Sigma_{hib} \subseteq (Elig_{L(G)}(st) \cup Occu(t)) \cap \Sigma_{hib}$$

iii.1b) $(\forall s \in L(G) \cap L_{samp})(\forall t \in \Sigma^*_{act}) st \in L(G) \Longrightarrow$

$$(Elig_{L(G)}(st) \cup Occu(t)) \cap \Sigma_{hib} \subseteq Elig_{L(G)}(s) \cap \Sigma_{hib}$$

Here part 'a' expresses the condition that that an event eligible to occur after a *tick* stays eligible until it occurs, and part 'b' expresses that no new prohibitable events become eligible until after the next *tick*.

5.2.1.1 SD Controllability Point (iii.1a)

We will now express SD controllability point (iii.1a) as a language inclusion problem. Our first step is to express as event occurrences whether a given $\alpha \in \Sigma_{hib}$ is enabled or disabled at a given state in TDES **G**.

Definition 5.5. Let TDES G = $(X, \Sigma, \delta, x_0, X_m)$, $\alpha \in \Sigma$, $e_\alpha \notin \Sigma$, $d_{\alpha,G} \notin \Sigma$ and $\Sigma^{ED}_{\alpha,G} = \Sigma \cup \{e_\alpha, d_{\alpha,G}\}$. Let $P_\Sigma : (\Sigma^{ED}_{\alpha,G})^* \rightarrow \Sigma^*$ be a natural projection. Construct TDES $ED_\alpha(G)$ from **G** by adding selfloops of e_α to all states where α is enabled, and selfloops of $d_{\alpha,G}$ to all states where α is disabled in **G**. Let $ED_\alpha(G)$ = $(X, \Sigma \cup \{e_\alpha, d_{\alpha,G}\}, \delta_{ED}, x_0, X_m)$ and for $x \in$ X

$$\delta_{ED}(x,\sigma) = \begin{cases} \delta(x,\sigma) & \text{if } \sigma \in \Sigma \text{ and } \delta(x,\sigma) \text{ is defined} \\ x & \text{if } \sigma = e_\alpha \text{ and } \delta(x,\alpha) \text{ is defined} \\ x & \text{if } \sigma = d_{\alpha,G} \text{ and } \delta(x,\alpha) \text{ is not defined} \\ undefined & \text{otherwise} \end{cases}$$

We first need to prove a result about $ED_\alpha(G)$ that we will need later in this section.

Lemma 5.7. *Let* $L' = \{s \in (\Sigma^{ED}_{\alpha,G})^* | P_\Sigma(s) \in L(G)$, *and*

$$[(\forall t)\ \textit{if}\ te_\alpha \leq s\ \textit{then}\ P_\Sigma(t)\alpha \in L(G)]\ \textit{and}$$

$$[(\forall t)\ \textit{if}\ td_{\alpha,G} \leq s\ \textit{then}\ P_\Sigma(t)\alpha \notin L(G)]\ \}$$

Then $L(ED_\alpha(G)) = L'$.

Proof. It is sufficient to show: 1) L($ED_\alpha(G)$) $\supseteq L'$ and 2) L($ED_\alpha(G)$) $\subseteq L'$

Part 1) Show $L' \subseteq L(ED_\alpha(G))$

Let s $\in (\Sigma_{\alpha,G}^{ED})^*$ and $P_\Sigma(s) \in$ L(G) (1)

Assume ($\forall$t) if $td_{\alpha,G} \leq s$ then $P_\Sigma(t)\alpha \notin L(G)$ (2)

Sufficient to show that s $\in L(ED_\alpha(G))$

Since $P_\Sigma(s) \in$ L(G) this implies $\delta(x_0, P_\Sigma(s))!$

We will show by induction on $|s|$ that $\delta_{ED}(x_0, s) = \delta(x_0, P_\Sigma(s))$. This will imply s $\in L(ED_\alpha(G))$.

Inductive Base: s $= \epsilon$

Clearly $\delta(x_0, P_\Sigma(\epsilon)) = \delta(x_0, \epsilon) = x_0 = \delta_{ED}(x_0, \epsilon)$

Inductive Step: s $= t'\sigma$

As s $\in (\Sigma_{\alpha,G}^{ED})^*$, we have $t' \in (\Sigma_{\alpha,G}^{ED})^*$ and $\sigma \in \Sigma_{\alpha,G}^{ED}$.

Since $P_\Sigma(s) \in$ L(G) by (1), we have $\delta(x_0, P_\Sigma(s))!$ and $\delta(x_0, P_\Sigma(t'))!$

Let $\delta(x_0, P_\Sigma(t')) = x'$.

By inductive assumption, $t' \in L(D_\alpha(G))$, $\delta_{ED}(x_0, t')!$, $\delta(x_0, P_\Sigma(t'))!$ and $\delta_{ED}(x_0, t') = \delta(x_0, P_\Sigma(t')) = x'$.

As $\sigma \in \Sigma_{\alpha,G}^{ED} = \Sigma \cup \{e_\alpha, d_{\alpha,G}\}$ we have three cases

case i) $\sigma \in \Sigma$

$\Longrightarrow P_\Sigma(t')\sigma = P_\Sigma(t'\sigma) = P_\Sigma(s) \in L(G)$ by (1)

This implies there exists x in X such that $\delta(x_0, P_\Sigma(t')) = x'$ and $\delta(x', \sigma) = x$.

Since $\delta_{ED}(x, \sigma) = \delta(x, \sigma)$ by the definition of $ED_\alpha(G)$, we have:

$\delta_{ED}(x', \sigma) = x$

By inductive assumption, $\delta_{ED}(x_0, t') = x'$.

$\Longrightarrow \delta_{ED}(x_0, t'\sigma) = x = \delta(x', \sigma) = \delta(x_0, P_\Sigma(t'\sigma))$

case ii) $\sigma = d_{\alpha,G}$

$\Longrightarrow t'd_{\alpha,G} \leq$ s

It follows by (2) that $P_\Sigma(t')\alpha \notin L(G)$

$\implies \delta(x', \alpha)$ is not defined

$\implies \delta_{ED}(x', d_{\alpha,G}) = x'$ by the definition of $ED_\alpha(G)$.

By inductive assumption $\delta_{ED}(x_0, t') = x'$, and $\delta(x_0, P_\Sigma(t')) = x'$.

This implies $\delta_{ED}(x_0, t'd_{\alpha,G}) = x' = \delta(x_0, P_\Sigma(t')) = \delta(x_0, P_\Sigma(t'd_{\alpha,G}))$

Case iii) $\sigma = e_\alpha$

We thus have t$e_\alpha \leq$ s and it follows by (2) that $P_\Sigma(t')\alpha \in L(G)$

$\implies \delta(x', \alpha)!$

$\implies \delta_{ED}(x', e_\alpha) = x'$ by the definition of $ED_\alpha(G)$.

By inductive assumption $\delta_{ED}(x_0, t') = x'$, and $\delta(x_0, P_\Sigma(t')) = x'$.

This implies $\delta_{ED}(x_0, t'e_\alpha) = x' = \delta(x_0, P_\Sigma(t')) = \delta(x_0, P_\Sigma(t'e_\alpha))$.

From case (i), (ii) and (iii), we can conclude that s $\in L(ED_\alpha(G))$.

Part 1 complete.

Part 2) Show L$(ED_\alpha(G)) \subseteq L'$

Let s $\in$ L$(ED_\alpha(G))$.

We will show by induction on $|s|$ that:

(a) $\delta(x_0, P_\Sigma(s)) = \delta_{ED}(x_0, s)$ i.e. $P_\Sigma(s) \in$ L(G)

(b) [($\forall$t) if t$d_{\alpha,G} \leq$ s then $P_\Sigma(t)\alpha \notin$ L(G)] and [($\forall$t) if t$e_\alpha \leq$ s then $P_\Sigma(t)\alpha \in$ L(G)]

Inductive Base: s $= \epsilon$

Then,

(a) clearly $\delta(x_0, P_\Sigma(\epsilon)) = \delta_{ED}(x_0, s) = x_0$.

(b) As s $= \epsilon$ cannot have t$d_{\alpha,G}$ or te_α as its prefix, and we are done.

Inductive Step: s $= t'\sigma$ with $\sigma \in \Sigma^D_{\alpha,G}$

Since s $\in$ L$(D_\alpha(G))$, also $t' \in L(D_\alpha(G))$.

$\implies \delta_{ED}(x_0, t') =$ x for some x $\in$ X.

By inductive assumption:

(a) $\delta(x_0, P_\Sigma(t')) = \delta_{ED}(x_0, t') = x$

(b) $[(\forall t'')$ if $t''d_{\alpha,G} \leq t'$ then $P_\Sigma(t'')\alpha \notin L(G)]$ and $[(\forall t'')$ if $t''e_\alpha \leq t'$ then $P_\Sigma(t'')\alpha \in L(G)]$

As $\sigma \in \Sigma^D_{\alpha,G} = \Sigma \cup \{e_\alpha, d_{\alpha,G}\}$ we have three cases:

case i) $\sigma \in \Sigma$

By inductive assumption (a), $\delta(x_0, P_\Sigma(t')) = \delta_{ED}(x_0, t') = x$ so $\delta_{ED}(x, \sigma) = \delta(x, \sigma)$ by the definition of $ED_\alpha(G)$.

It then follows that $\delta(x_0, P_\Sigma(t')\sigma) = \delta(x_0, P_\Sigma(t'\sigma)) = \delta_{ED}(x_0, t'\sigma)$

$\Longrightarrow \delta(x_0, P_\Sigma(s)) = \delta_{ED}(x_0, s)$

We have now shown condition (a).

We note that (b) follows immediately from the inductive assumption and the facts that $\sigma \neq e_\alpha$ and $\sigma \neq d_{\alpha,G}$.

case ii) $\sigma = d_{\alpha,G}$

Note $\delta_{ED}(x, d_{\alpha,G})$ is only defined if $\delta(x, \alpha)$ is undefined, by the definition of $ED_\alpha(G)$.

By inductive assumption, it follows that $\delta(x, \alpha)$ is not defined.

$\Longrightarrow \delta_{ED}(x, d_{\alpha,G}) = \mathrm{x}$

This implies $\delta(x_0, P_\Sigma(s)) = \delta(x_0, P_\Sigma(t'd_{\alpha,G}) = \delta(x_0, P_\Sigma(t')) = \mathrm{x} = \delta_{ED}(x_0, t'd_{\alpha,G})$ $= \delta_{ED}(x_0, s)$.

We have thus shown condition (a).

Now to show (b), let $\mathrm{t}d_{\alpha,G} \leq \mathrm{s}$.

If t is a proper prefix of t' it follows from inductive assumption (b) that for all $\mathrm{t}d_{\alpha,G} \leq \mathrm{s}$ it holds that $P_\Sigma(t)\alpha \notin \mathrm{L(G)}$.

Otherwise $\mathrm{t} = t'$ and since $\delta(x_0, P_\Sigma(t')\alpha)$ is not defined, it also follows that $P_\Sigma(t)\alpha \notin L(G)$.

Now, let $\mathrm{t}e_\alpha \leq \mathrm{s}$.

As $\sigma = d_{\alpha,G}, te_\alpha < \mathrm{s}$, thus $\mathrm{t}e_\alpha \leq t'$. The result thus follows from the inductive assumption.

Case iii) $\sigma = e_\alpha$

Note that $\delta_{ED}(x, e_\alpha)$ is defined only if $\delta(x, \alpha)$ is defined, by the definition of $ED_\alpha(G)$.

By inductive assumption it follows that $\delta(x, \alpha)$ is defined.

$\Longrightarrow \delta_{ED}(x_0, e_\alpha) = x$

This implies $\delta(x_0, P_\Sigma(s)) = \delta(x_0, P_\Sigma(t'e_\alpha)) = \delta(x_0, P_\Sigma(t)) = x = \delta_{ED}(x_0, t'e_\alpha) = \delta_{ED}(x_0, s)$.

We have now shown condition (a).

To show (b), let t$e_\alpha \leq$ s.

If t is a proper prefix of t' it follows from inductive assumption (b) that for all t$e_\alpha \leq$ s, it holds that $P_\Sigma(t)\alpha \in L(G)$.

Otherwise t $= t'$ and since $\delta(x_0, P_\Sigma(t')\alpha)$ is defined, it follows that $P_\Sigma(t)\alpha \in$ L(G).

Now, let t$d_{\alpha,G} \leq$ s.

As $\sigma \neq d_{\alpha,G}, td_{\alpha,G} <$ s, thus $td_{\alpha,G} \leq t'$. The result thus follows from the inductive assumption.

Part 2 complete.

By part 1 and 2 we have L$(ED_\alpha(G)) = L'$ □

Typically we construct our system modularly. We thus have G $= G_1 \parallel$ ..... $\parallel G_n = (X, \Sigma, \delta, x_{0,i}, X_{m,i})$, with $G_i = (X_i, \Sigma_i, \delta, x_{0,i}, X_{m,i})$, for i=1,.....,n. For each G_i, we will construct an $ED_\alpha(G_i)$. They will all share $e_\alpha \notin \Sigma$, but will each have their own $d_{\alpha,G_i} \notin \Sigma$ event. This is to capture the fact that α is only possible when enabled by all, but disabled when at least one G_i disables it. We would then construct $ED_\alpha(G)$ modularly as $ED_\alpha(G) = ED_\alpha(G_1) \parallel$ ..... $\parallel ED_\alpha(G_n)$. We would thus have $D_{\alpha,G} = \{d_{\alpha,G_1}, ..., d_{\alpha,G_n}\}, \Sigma^{ED}_{\alpha,G} = \Sigma^{ED}_{\alpha,G_1} \cup ..... \cup \Sigma^{ED}_{\alpha,G_n} = \Sigma \dot\cup \{e_\alpha, d_{\alpha,G_1}, ..., d_{\alpha,G_n}\}$. Finally, let $P_\Sigma : (\Sigma^{ED}_{\alpha,G})^* \rightarrow \Sigma^*$ be a natural projection.

As we will be constructing in our proofs an $ED_\alpha(G_i)$ for each $\alpha \in \Sigma_{hib}$, and applying Lemma 5.7 to the result, this only makes sense if α is in the event set

of TDES $G_i = (X_i, \Sigma_i, \delta, x_{0,i}, X_{m,i})$. For simplicity of our proofs, we will assume each G_i is over event set Σ. This does not represent a loss in generality, since if $\Sigma_i < \Sigma$, we can just selfloop at every state in G_i the events in $\Sigma \setminus \Sigma_i$, without altering the resulting G = $G_1 \parallel ..... \parallel G_n$.

Finally, we need to construct TDES $T_{\alpha,G}$, for each $\alpha \in \Sigma$, as shown in Figure 5.2. We note that the alphabet of $T_{\alpha,G}$ is $\Sigma^{ED}_{\alpha,G}$. TDES $T_{\alpha,G}$ captures the idea that α should stay enabled for the sampling period until it occurs. Now we will prove a result about $T_{\alpha,G}$ that we will need later in this section.

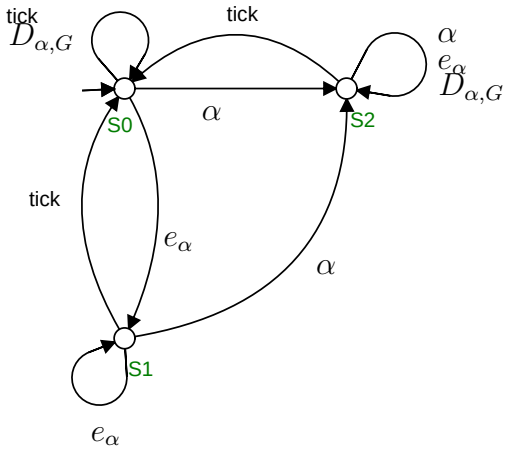


Figure 5.2: TDES $T_{\alpha,G}$
selfloop: $\Sigma \setminus \{tick, \alpha\}$

Lemma 5.8. *Let* $L' = \{s \in (\Sigma^{ED}_{\alpha,G})^* |$ *for all* $uvd \leq s$ *such that* $P_\Sigma(u) \in L_{samp}$, $P_\Sigma(v) \in \Sigma^*_{act}$ *and* $d \in D_{\alpha,G}$, *it holds that* $\alpha \in Occu(v)$ *or* $e_\alpha \notin Occu(v)\}$

Then $L(T_{\alpha,G}) = L'$.

Proof. Let $P_{ED} : (\Sigma^{ED}_{\alpha,G})^* \rightarrow (\{e_\alpha\} \cup D_{\alpha,G})^*$ be a natural projection.

Let δ_T be the transition function for $T_{\alpha,G}$.

It is sufficient to show: 1) $L(T_{\alpha,G}) \subseteq L'$ and 2) $L(T_{\alpha,G}) \supseteq L'$

Part 1) Show: $L(T_{\alpha,G}) \subseteq L'$.

Let s $\in L(T_{\alpha,G})$.

Let $uvd \leq s$ such that $P_\Sigma(u) \in L_{samp}, P_\Sigma(v) \in \Sigma^*_{act}, d \in D_{\alpha,G}$.

Show $\alpha \in Occu(v)$ or $e_\alpha \notin Occu(v)$

Since s $\in L(T_{\alpha,G})$, clearly s $\in (\Sigma^{ED}_{\alpha,G})^*$.

Also, this implies $\delta_T(S0, s)$!.

Since $uvd \leq s$ there exists $x, y, z \in \{S0, S1, S2\}$ such that:

$\delta_T(S0, u) = x$ and $\delta_T(x, v) = y$ and $\delta_T(y, d) = z$

Clearly by the construction of $T_{\alpha,G}$, it follows from $P_\Sigma(u) \in L_{samp}$ that x = S0 or x = S1. (1)

From d $\in D_{\alpha,G}$, it follows by the construction of $T_{\alpha,G}$, that y = z = S0 or y = z = S2 as $D_{\alpha,G}$ events are selfloops only in $T_{\alpha,G}$.

case i) y = z = S0

By (1), x = S0 or x = S1.

From P_Σ(v) $\in \Sigma^*_{act}$ it follows that v does not contain *tick*.

$\Longrightarrow$ x = S0 and P_{ED}(v) $\in (D_{\alpha,G})^*$

This implies $e_\alpha \notin$ Occu(v).

case ii) y = z = S2

From (1), we have x = S0 or x = S1. In either case, to reach S2, v must contain an "α" by construction of $T_{\alpha,G}$

$\Longrightarrow \alpha \in$ Occu(v)

By case (i) and (ii), we have: $\alpha \in$ Occu(v) or $e_\alpha \notin$ Occu(v).

Since u, v, d were chosen arbitrarily, s $\in L'$.

Part 1 complete.

Part 2) Show $L' \subseteq L(T_{\alpha,G})$

Let s $\in L'$.

We show by induction on $|s|$ that the following holds:

a) If $P_\Sigma(s) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$, then δ_T(S0,s) = S2

b) If $P_\Sigma(s) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ and s=u$e_\alpha t'$ for some u and some $t' \in (\Sigma \setminus \{\alpha, tick\})^*$, then δ_T(S0,s) = S1

c) If $P_\Sigma(s) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ and s$\neq$ u$e_\alpha t'$ for all u, and for all $t' \in (\Sigma \setminus \{\alpha, tick\})^*$ then δ_T(S0,s) = S0

Note that here we are using the fact that $\sigma' \in (\Sigma \setminus \{\alpha, tick\})^*$ are selflooped at every state in $T_{\alpha,G}$, so σ' does not cause a state change.

Inductive Base: s = ϵ

Then $P_\Sigma(s) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ and s $\neq$ u$e_\alpha t'$ for all u, and for all $t' \in (\Sigma \setminus \{\alpha, tick\})^*$ so we need to show δ_T(S0,s) = S0.

Clearly, δ_T(S0,ϵ) = S0

Inductive Step: s = tσ, with t $\in (\Sigma^{ED}_{\alpha,G})^*$ and $|t|$ = n and $\sigma \in (\Sigma^{ED}_{\alpha,G})$

By inductive assumption:

a) If $P_\Sigma(t) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$, then δ_T(S0,t) = S2

b) If $P_\Sigma(t) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ and t=u$e_\alpha t'$ for some u and some $t' \in (\Sigma \setminus \{\alpha, tick\})^*$, then δ_T(S0,t) = S1

c) If $P_\Sigma(t) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ and t$\neq$ u$e_\alpha t'$ for all u and for all $t' \in (\Sigma \setminus \{\alpha, tick\})^*$, then δ_T(S0,t) = S0

This implies t $\in L(T_{\alpha,G})$ as t must satisfy one of these cases, and each case implies δ_T(S0,t)!. (1)

Consider the following seven cases:

case i) $\sigma = tick$

$\implies P_\Sigma(s) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ and s $\neq$ u$e_\alpha t'$ for all u, and for all $t' \in (\Sigma \setminus \{\alpha, tick\})^*$.

It is sufficient to show δ_T(S0,s) = S0.

We have δ_T(S0,t)! by (1) and thus δ_T(S0,s) = $\delta_T(\delta_T$(S0,t),$tick)$ = S0, as tick transitions are defined in all states in $T_{\alpha,G}$, and all lead to S0.

case ii) $\sigma = \alpha$

$\Longrightarrow P_\Sigma(s) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$

It is sufficient to show δ_T(S0,s) = S2.

We have δ_T(S0,t)! by (1), and thus δ_T(S0,s) = $\delta_T(\delta_T$(S0,t),$\alpha)$ = S2, as α is defined as all states in $T_{\alpha,G}$ and always lead to S2.

case iii) $\sigma = e_\alpha$ and $P_\Sigma(s) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$

Sufficient to show δ_T(S0,s) = S1.

We have δ_T(S0,t)! by (1).

If $P_\Sigma(s) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ then also $P_\Sigma(t) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$, as $P_\Sigma(s) = P_\Sigma(te_\alpha) = P_\Sigma(t)$.

So by inductive assumption (b) and (c), either δ_T(S0,t) = S0 or δ_T(S0,t) = S1.

It thus follows by the construction of $T_{\alpha,G}$ that:

δ_T(S0,s) = δ_T(S0,tσ) = S1

case iv) $\sigma = e_\alpha$ and $P_\Sigma(s) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$

Sufficient to show δ_T(S0,s) = S2.

We have δ_T(S0,t)! by (1).

If $P_\Sigma(s) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ then also $P_\Sigma(t) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$

So by inductive assumption (a), δ_T(S0,t) = S2.

It follows by the construction of $T_{\alpha,G}$ that:

δ_T(S0,s) = S2

case v) $\sigma \in D_{\alpha,G}$ and $P_\Sigma(s) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$

Sufficient to show δ_T(S0,s) = S0.

We note that if $P_\Sigma(s) \in L_{samp}.(\Sigma\setminus\{\alpha, tick\})^*$, then $P_\Sigma(t) \in L_{samp}.(\Sigma\setminus\{\alpha, tick\})^*$.

Note s = td for some d $\in D_{\alpha,G}$.

Then since $P_\Sigma(s) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$, we have s = uvd for some u with $P_\Sigma(u) \in L_{samp}$ and v with $P_\Sigma(v) \in (\Sigma \setminus \{\alpha, tick\})^* \subseteq \Sigma^*_{act}$.

From the definition of L' it follows for all such u and v that $e_\alpha \notin$ Occu(v).

$\Longrightarrow t \neq u'e_\alpha t'$ for any u' and any $t' \in (\Sigma \setminus \{\alpha, tick\})^*$.

By inductive assumption (c), δ_T(S0,t) = S0.

It follows by the construction of $T_{\alpha,G}$ that:

δ_T(S0,s) = δ_T(S0,tσ)= S0

case vi) $\sigma \in D_{\alpha,G}$ and $P_\Sigma(s) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$

Sufficient to show δ_T(S0,s) = S2.

If $P_\Sigma(s) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ then $P_\Sigma(t) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$.

So by inductive assumption (a), δ_T(S0,t) = S2.

It follows by the construction of $T_{\alpha,G}$ that:

δ_T(S0,s) = S2

case vii) $\sigma \in \Sigma \setminus \{\alpha, tick\}$

For condition (a), if $P_\Sigma(t) \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$ then $P_\Sigma(t\sigma) = P_\Sigma(t)\sigma \notin L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$. Also as all $\sigma' \in (\Sigma \setminus \{\alpha, tick\})^*$ are selflooped in $T_{\alpha,G}$, δ_T(S0,t) = δ_T(S0,tσ) = S2.

For condition (b) and (c), if $P_\Sigma(t) \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$, then $P_\Sigma(t\sigma) = P_\Sigma(t)\sigma \in L_{samp}.(\Sigma \setminus \{\alpha, tick\})^*$.

For (b), if t =u$e_\alpha t'$ then tσ = u$e_\alpha t'\sigma$. Taking $t'' = t'\sigma \in (\Sigma \setminus \{\alpha, tick\})^*$, and (b) is still satisfied. Also, δ_T(S0,tσ) = δ_T(S0,t) = S1.

For (c), if for all u and $t' \in (\Sigma \setminus \{\alpha, tick\})^*$, t $\neq$ u$e_\alpha t'$, then t$\sigma \neq ue_\alpha t'\sigma$ for any such u and t'. Also, δ_T(S0,tσ) = δ_T(S0,t) = S0.

Inductive step complete.

Now we have shown for all s $\in L'$ that δ_T(S0,s)! and that δ_T(S0,s)=S0 or δ_T(S0,s)=S1 or δ_T(S0,s)=S2.

This implies s $\in L(T_{\alpha,G})$.

Part 2 complete.

By part 1 and 2 we have $L(T_{\alpha,G}) = L'$ □

Now we present two propositions that will be useful for our main theorem

later in this section. The two proofs that follow are based on initial proof sketches by the author's thesis supervisors, Dr. R. Malik and Dr. R. Leduc.

Proposition 5.3. Let G = $G_1 \parallel ..... \parallel G_n = (X, \Sigma, \delta, x_0, X_m)$ with $G_i = (X_i, \Sigma, \delta, x_{0,i}, X_{m,i})$, i = 1,....n. If G satisfies SD controllability point (iii.1) then $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$.

Proof. Assume G satisfies SD controllability point (iii.1).

Let $P_i : (\Sigma^{ED}_{\alpha,G})^* \to (\Sigma \cup \{e_\alpha, d_{\alpha,G_i}\})^*$ be a natural projection, i = 1,...,n.

We note that as $\Sigma \subseteq \Sigma \cup \{e_\alpha, d_{\alpha,G_i}\}$, that $P_\Sigma \circ P_i = P_\Sigma$. (1)

Let $\alpha \in \Sigma_{hib}, s \in L(ED_\alpha(G)) \subseteq (\Sigma^{ED}_{\alpha,G})^*$.

Let $uvd_{\alpha,G_i} \leq s$ such that $P_\Sigma(u) \in L_{samp}, P_\Sigma(v) \in \Sigma^*_{act}, i \in 1....n$, and $e_\alpha \in Occu(v)$.

Sufficient to show $\alpha \in Occu(v)$ by Lemma 5.8.

Since $uvd_{\alpha,G_i} \leq s \in L(ED_\alpha(G))$ we have:

$P_i(uv)d_{\alpha,G_i} \in L(ED_\alpha(G_i))$

$\Longrightarrow P_\Sigma(P_i(uv))\alpha \notin L(G_i)$ by Lemma 5.7

$\Longrightarrow P_\Sigma(uv)\alpha \notin L(G_i)$ by (1)

$\Longrightarrow P_\Sigma(uv)\alpha \notin L(G) = L(G_1) \cap ..... \cap L(G_n)$ (2)

As $s \in L(ED_\alpha(G))$, we have $P_j(s) \in L(ED_\alpha(G_j))$ for j = 1,....n.

$\Longrightarrow P_\Sigma(P_j(s)) \in L(G_j)$ by Lemma 5.8, j = 1,...,n

$\Longrightarrow P_\Sigma(s) \in L(G_j)$ by (1), j = 1,....,n

$\Longrightarrow P_\Sigma(s) \in L(G)$

$\Longrightarrow P_\Sigma(uv) \leq P_\Sigma(uvd_{\alpha,G_i}) \leq P_\Sigma(s) \in L(G)$ (3)

Since $e_\alpha \in Occu(v)$, v can be written as $v = v_1 e_\alpha v_2$ with $v_1, v_2 \in (\Sigma^{ED}_{\alpha,G})^*$ and $P_\Sigma(v_1) \leq P_\Sigma(v) \in \Sigma^*_{act}$.

Since $uv_1e_\alpha \leq uv_1e_\alpha v_2 = uv \leq s \in L(ED_\alpha(G))$, we have:

$P_j(uv_1)e_\alpha \in L(ED_\alpha(G_j))$ for all j=1 .... n

$\Longrightarrow P_\Sigma(P_j(uv_1))\alpha \in L(G_j)$ for all j by Lemma 5.7.

$\Longrightarrow P_\Sigma(uv_1)\alpha \in L(G_j)$ for all j by (1)

$\Longrightarrow P_\Sigma(uv_1)\alpha \in L(G)$

$\Longrightarrow P_\Sigma(u) \in L(G) \cap L_{samp}$ and $P_\Sigma(v_1) \in \Sigma^*_{act}$ with $P_\Sigma(uv_1) \in L(G)$

By definition of SD controllability point (iii.1), we have:

$$(Elig_{L(G)}(P_\Sigma(uv_1)) \cup Occu(P_\Sigma(v_1))) \cap \Sigma_{hib} = Elig_{L(G)}(P_\Sigma(u)) \cap \Sigma_{hib} \quad (4)$$

Similarly using $P_\Sigma(v) \in \Sigma^*_{act}$ with $P_\Sigma(uv) \in L(G)$ (by (3)) and definition of SD controllability point (iii.1), we have:

$$(Elig_{L(G)}(P_\Sigma(uv)) \cup Occu(P_\Sigma(v))) \cap \Sigma_{hib} = Elig_{L(G)}(P_\Sigma(u)) \cap \Sigma_{hib} \quad (5)$$

Since $P_\Sigma(uv_1)\alpha \in L(G) \Longrightarrow \alpha \in Elig_{L(G)}(P_\Sigma(uv_1))$, and also $\alpha \in \Sigma_{hib}$ we have:

$\alpha \in (Elig_{L(G)}(P_\Sigma(uv_1)) \cup Occu(P_\Sigma(v_1))) \cap \Sigma_{hib}$

$= Elig_{L(G)}(P_\Sigma(u)) \cap \Sigma_{hib}$ by (4)

$= (Elig_{L(G)}(P_\Sigma(uv)) \cup Occu(P_\Sigma(v))) \cap \Sigma_{hib}$ by (5)

$\subseteq (Elig_{L(G)}(P_\Sigma(uv)) \cup Occu(P_\Sigma(v)))$

By (2) $\alpha \notin Elig_{L(G)}(P_\Sigma(uv))$, thus we have $\alpha \in Occu(P_\Sigma(v)) \subseteq Occu(v)$, as required. □

Proposition 5.4. Let G = $G_1 \parallel ..... \parallel G_n = (X, \Sigma, \delta, x_0, X_m)$ with $G_i = (X_i, \Sigma, \delta, x_{0,i}, X_{m,i})$, i = 1,....,n. If $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$ then

$(\forall s \in L(G) \cap L_{samp})(\forall t \in \Sigma^*_{act})$

$st \in L(G) \Longrightarrow Elig_{L(G)}(s) \cap \Sigma_{hib} \subseteq (Elig_{L(G)}(st) \cup Occu(t)) \cap \Sigma_{hib}$

Proof. Assume $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$.

Let $s \in L(G) \cap L_{samp}$, $t \in \Sigma^*_{act}$ and assume $st \in L(G)$.

Let $\alpha \in Elig_{L(G)}(s) \cap \Sigma_{hib}$, i.e. $\alpha \in \Sigma_{hib}$ and $s\alpha \in L(G)$.

Let $\alpha \notin Elig_{L(G)}(st)$, i.e. $st\alpha \notin L(G)$.

Sufficient to show $\alpha \in Occu(t)$.

Since $st \in L(G)$ and $st\alpha \notin L(G)$ we have:

$(\exists i \in \{1....n\})st\alpha \notin L(G_i)$.

We thus have $se_\alpha td_{\alpha,G_i} \in L(ED_\alpha(G_i)$ because $P_\Sigma(se_\alpha td_{\alpha,G_i}) = st \in L(G) \subseteq L(G_i)$, $P_\Sigma(s)\alpha = s\alpha \in L(G) \subseteq L(G_i)$, $P_\Sigma(se_\alpha t)\alpha = st\alpha \notin L(G_i)$, and $d_{\alpha,G_i}, e_\alpha \notin Occu(s) \cup Occu(t)$ since $st \in L(G) \subseteq \Sigma^*$.

We also have $se_\alpha t \in L(ED_\alpha(G_j))$ for all j $\neq$ i because $P_\Sigma(se_\alpha t) = st \in L(G) \subseteq L(G_j)$, $P_\Sigma(s)\alpha = s\alpha \in L(G) \subseteq L(G_j)$, and $d_{\alpha,G_j}, e_\alpha \notin Occu(s) \cup Occu(t)$ since $st \in L(G) \subseteq \Sigma^*$.

It follows that $se_\alpha td_{\alpha,G_i} \in L(ED_\alpha(G))$ because d_{α,G_i} is not in the alphabet of $G_j, j \neq i$.

$\Longrightarrow se_\alpha td_{\alpha,G_i} \in L(T_{\alpha,G})$ as $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ by assumption.

Since $se_\alpha td_{\alpha,G_i} \leq se_\alpha td_{\alpha,G_i} \in L(T_{\alpha,G})$, $P_\Sigma(s) = s \in L_{samp}, P_\Sigma(e_\alpha t) \in \Sigma^*_{act}, d_{\alpha,G_i} \in D_{\alpha,G}$, and $e_\alpha \in Occu(e_\alpha t)$, we have by Lemma 5.8:

$\alpha \in Occu(e_\alpha t)$

$\Longrightarrow \alpha \in Occu(t)$ as required. □

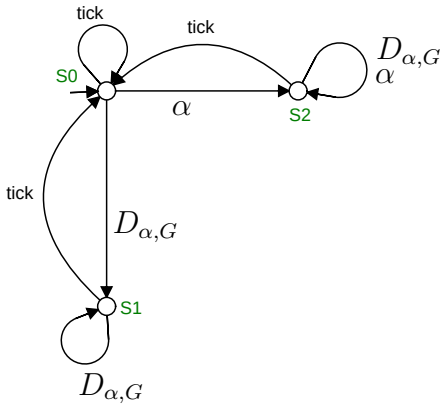
5.2.1.2 SD Controllability Point (iii.1b)

We will now express SD controllability point (iii.1b) as a language inclusion problem. Our first step is to express as event occurrences whether a given $\alpha \in \Sigma_{hib}$ is disabled at a given state in a given TDES.

Let $\alpha \in \Sigma$, and TDES G = $(X, \Sigma, \delta, x_0, X_m) = G_1 \parallel ..... \parallel G_n$ where **G** is constructed from the synchronous product of TDES $G_i = (X_i, \Sigma_i, \delta, x_{0,i}, X_{m,i})$, i = 1,....,n. For each G_i we will construct a $D_\alpha(G_i)$ using definition 5.3 on page 54, each with their own d_{α,G_i} event with alphabet $\Sigma^D_{\alpha,G_i} = \Sigma_i \cup \{d_{\alpha,G_i}\}$. Note that if $\alpha \notin \Sigma_i$, then α is effectively selflooped at every state in G_i, so d_{α,G_i}, will never occur in $D_\alpha(G_i)$.

We then construct $D_\alpha(G)$ modularly with $D_\alpha(G) = D_\alpha(G_1) \parallel ..... \parallel D_\alpha(G_n)$, $D_{\alpha,G} = \{d_{\alpha,G_1}, \ldots, d_{\alpha,G_n}\}$, $\Sigma^D_{\alpha,G} = \Sigma^D_{\alpha,G_1} \cup \ldots \cup \Sigma^D_{\alpha,G_n} = \Sigma \dot\cup \{d_{\alpha,G_1}, \ldots, d_{\alpha,G_n}\}$, and the natural projection $P_\Sigma{:}(\Sigma^D_{\alpha,G})^* \rightarrow \Sigma^*$.

Finally we need to construct TDES $U_{\alpha,G}$ as shown in Figure 5.3. TDES $U_{\alpha,G}$ captures the idea that α can only be enabled immediately after the *tick* has occured. We note that $U_{\alpha,G}$ has alphabet $\Sigma^D_{\alpha,G}$. We now need to prove a result about $U_{\alpha,G}$ that we will need later in this section.



5.3: TDES $U_{\alpha,G}$
selfloop: $\Sigma \setminus \{tick, \alpha\}$

Lemma 5.9. *Let* $P_D : (\Sigma^D_{\alpha,G})^* \to (D_{\alpha,G} \cup \{\alpha, tick\})^*$ *be a natural projection. Let*

$$L' = \{s \in (\Sigma^D_{\alpha,G})^* | (\forall u, v \in (\Sigma^D_{\alpha,G})^*)\ P_\Sigma(u) \in L_{samp}, P_D(v) \in (D_{\alpha,G})^*, \text{ and}$$
$$uv\alpha \leq s \Longrightarrow P_D(v) = \epsilon\}$$

Then $L(U_{\alpha,G}) = L'$.

Proof. Let δ_U be the transition function for $U_{\alpha,G}$.

It is sufficient to show: 1) $L(U_{\alpha,G}) \subseteq L'$ and 2) $L(U_{\alpha,G}) \supseteq L'$

Part 1) Show $L(U_{\alpha,G}) \subseteq L'$

Let s $\in L(U_{\alpha,G})$.

Let $u, v \in (\Sigma^D_{\alpha,G})^*$.

Assume $uvd \leq$ s, $P_\Sigma(u) \in L_{samp}$, and $P_D(v) \in (D_{\alpha,G})^*$.

Must show implies $P_D(v) = \epsilon$.

Since s $\in L(U_{\alpha,G})$, clearly s $\in (\Sigma^D_{\alpha,G})^*$.

Also, this implies $\delta_U(S0, s)!$.

Since $uv\alpha \leq s$ there exist $x, y, z \in \{S0, S1, S2\}$ such that:

$\delta_U(S0, u) = x$, $\delta_U(x, v) = y$, and $\delta_U(y, \alpha) = z$

For α, it follows by the construction of $U_{\alpha,G}$ that y = S0 and z = S2, or y = z = S2.

Since $P_D(v) \in (D_{\alpha,G})^*$, if follows that v does not contain α. As $P_\Sigma(u) \in L_{samp}$, the construction of $U_{\alpha,G}$ ensures u takes us to state S0 or S1. Also, state S2 cannot be reached from S0 or S1 without an α.

$\implies$ y $\neq$ S2.

$\implies$ y = S0 and z = S2.

Clearly, by the construction of $U_{\alpha,G}$ it follows from $P_\Sigma(u) \in L_{samp}$ that x = S0 or x = S1.

$\implies \delta_U$(S0,v) = S0 or δ_U(S1,v) = S0

As $P_D(v) \in (D_{\alpha,G})^*$ it follows that v does not contain *tick*.

This implies x = S0 and $P_D(v) = \epsilon$, as S0 cannot be reached from S1 without a *tick*, and any d $\in D_{\alpha,G}$ would take us to S1.

Since u, v were chosen arbitrarily, s $\in L'$.

Part 1 complete.

Part 2) Show $L' \subseteq L(U_{\alpha,G})$

Let $\Sigma_0^* = \Sigma \setminus \{\alpha, tick\}$. These are events selflooped at each state in $U_{\alpha,G}$.

Let $L^D_{samp} = (\Sigma^D_{\alpha,G})^*.tick.\Sigma_0^* \cup \Sigma_0^*$

Let s $\in L'$.

We show by induction on $|s|$ that the following holds:

a) If s $\in L^D_{samp}$ then δ_U(S0,s) = S0

b) If s $\notin L^D_{samp}$ and s = uv for some u $\in L^D_{samp}$ and some v such that $P_D(v) \in (D_{\alpha,G})^+$, then δ_U(S0,s) = S1

c) If s $\notin L^D_{samp}$ and s $\neq$ uv for any u $\in L^D_{samp}$ and any v such that $P_D(v) \in (D_{\alpha,G})^+$, then δ_U(S0,s) = S2

Inductive Base: s = ϵ

It is suffiecient to show δ_U(S0,s) = S0, as $\epsilon \in L^D_{samp}$.

Clearly, δ_U(S0,ϵ) = S0

Inductive Step: s = tσ with t $\in (\Sigma^D_{\alpha,G})^*$ and $|t|$ = n and $\sigma \in (\Sigma^D_{\alpha,G})$.

By inductive assumption, we have:

a) If t $\in L^D_{samp}$ then δ_U(S0,t) = S0

b) If t $\notin L^D_{samp}$ and t = uv for some u $\in L^D_{samp}$ and some v such that $P_D(v) \in (D_{\alpha,G})^+$, then δ_U(S0,t) = S1

c) If t $\notin L^D_{samp}$ t $\neq$ uv for any u $\in L^D_{samp}$ and any v such that $P_D(v) \in (D_{\alpha,G})^+$, then δ_U(S0,t) = S2

This implies t $\in L(U_{\alpha,G})$. (2)

Consider the following three cases:

case i) s $\in L^D_{samp}$

Note that $\sigma \in \Sigma_0 \cup \{tick\}$ and it is enough to show δ_U(S0,s) = S0.

By (2), δ_U(S0,t)!, and thus δ_U(S0,s) = δ_U(S0,t *tick*) = S0 by construction of $U_{\alpha,G}$ as *tick* is defined at all states, and always leads to S0.

If $\sigma \in \Sigma_0$, then if s $\in L^D_{samp}$, then t $\in L^D_{samp}$, thus δ_u(S0,t) = S0.

$\Longrightarrow \delta_u$(S0,s) = δ_u(S0,tσ) = δ_u(S0,t) = S0.

case ii) s $\notin L^D_{samp}$ and s = uv for some u $\in L^D_{samp}$ and some v such that $P_D(v) \in (D_{\alpha,G})^+$

It is sufficient to show δ_U(S0,s) = S1.

Since u $\in L^D_{samp}$ and v $\neq \epsilon$, we have u $\leq$ t. We thus have by inductive assumption (a) that δ_U(S0,u) = S0.

Since $P_D(v) \in (D_{\alpha,G})^+$, it follows that v contains at least one d $\in D_{\alpha,G}$, and no α or *tick* events. It follows by the construction of $U_{\alpha,G}$ that:

δ_U(S0,v) = S1

This implies δ_U(S0,s) = S1.

case iii) s $\notin L^D_{samp}$ and s $\neq$ uv for any u $\in L^D_{samp}$ and any v such that $P_D(v) \in$

$(D_{\alpha,G})^+$

It is sufficient to show δ_U(S0,s) = S2.

We first note that as $\epsilon \in L^D_{samp}$, there exists at least one u $\in L^D_{samp}$ such that u $\leq$ s, and $P_\Sigma(u) \in L_{samp}$.

As s $\neq$ uv for any u $\in L^D_{samp}$, we can choose the largest u $\in L^D_{samp}$ such that u $\leq$ s and $P_\Sigma(u) \in L^D_{samp}$ (i.e. for $(\forall u' \in L_{samp})$ $u' \leq$ s and $P_\Sigma(u') \in L_{samp} \Rightarrow u' \leq$ u) and there will still not exist a v such that $P_D(v) \in (D_{\alpha,G})^+$.

Let u be as defined above.

We note that as u $\in L^D_{samp}$, either u $\in \Sigma_0^*$ or u $\in (\Sigma^D_{\alpha,G})^*.tick.\Sigma_0^*$, i.e. either u at most contains events in Σ_0 (thus no $\sigma' \in D_{\alpha,G} \cup \{\alpha, tick\}$) or u ends in a tick followed by at most events in Σ_0.

As $P_\Sigma(u) \in L_{samp}$ as well, this implies either u $= \epsilon$, or ends in a *tick*.

We note that adding the $P_\Sigma(u) \in L_{samp}$ condition is equivalent to taking the largest u $\in L^D_{samp}$ and stripping off any trailing $\sigma \in \Sigma_0$.

Let v $\in (\Sigma^D_{\alpha,G})^*$ such that uv = s and we have $P_D(v) \notin (D_{\alpha,G})^+$ by the statement of case (iii).

This implies $P_D(v) \neq \epsilon$ since s $\notin L^D_{samp}$ and thus there must exist and α or d $\in D_{\alpha,G}$ that follows the largest L^D_{samp} prefix of s.

Also as u is the largest L^D_{samp} prefix of s such that $P_\Sigma(u) \in L_{samp}$, there cannot be a *tick* in Occu(v).

As $P_D(v) \neq \epsilon$, contains no ticks and does not contain only d $\in D_{\alpha,G}$, the definition of P_D implies that $\alpha \in$ Occu(v).

As u $\in L^D_{samp}$ and v $\neq \epsilon$, we have u $\leq$ t. We thus have by inductive assumption (a) that:

$$\delta_U(\text{S0,u}) = \text{S0} \tag{3}$$

As *tick* $\notin$ Occu(v), we have v $\in (D_{\alpha,G} \cup \{\alpha\} \cup \Sigma_0)^*$.

$\Longrightarrow (\exists t_1 \in (D_{\alpha,G} \cup \Sigma_0)^*)(\exists t_2 \in (D_{\alpha,G} \cup \{\alpha\} \cup \Sigma_0)^*)v = t_1 \alpha t_2$

We thus have $ut_1\alpha \leq uv \leq s, P_\Sigma(u) \in L_{samp}, t_1 \in (\Sigma^D_{\alpha,G})^*$, and $P_D(t_1) \in (D_{\alpha,G})^*$.
As s $\in L'$, we can conclude by definition of L' that: $P_D(t_1) = \epsilon$.
$\Longrightarrow t_1 \in \Sigma_0^*$
As δ_U(S0,u) = S0 by (3), we have δ_U(S0,ut_1) = S0 as $\sigma \in \Sigma_0$ are selflooped at every state in $U_{\alpha,G}$.
As uv = s, and v = $t_1\alpha t_2$, we see that $\delta_U(S0, ut_1\alpha) =$ S2 by construction of $U_{\alpha,G}$.
All that remains is to show that $\delta_U(S0, ut_1\alpha t_2) =$ S2.
We next note that each $\sigma' \in (D_{\alpha,G} \cup \{\alpha\} \cup \Sigma_0)$ is selflooped at state S2 in $U_{\alpha,G}$.
$\Longrightarrow \delta_U(S0, ut_1\alpha t_2) =$ S2 as $t_2 \in (D_{\alpha,G} \cup \{\alpha\} \cup \Sigma_0)^*$.
$\Longrightarrow \delta_U$(S0,s) = S2, as required.
Inductive step complete.

Now we have shown for s $\in L'$, that δ_U(S0,s)=S0, δ_U(S0,s)=S1 or δ_U(S0,s)=S2.
This implies s $\in L(U_{\alpha,G})$.
Part 2 complete.
By part 1 and 2 we have $L(U_{\alpha,G}) = L'$ □

Now we will present two propositions which will be useful for our main theorem later in this section. The proofs that follow are based on an initial proof sketch by the author's thesis supervisors, Dr. R. Malik and Dr. R. Leduc. For consistency with **Section 5.2.1.1**, we will assume that all of the component G_i are defined over event set Σ in the remaining propositions and theorems in this section. Please see the discussion on page 69 for more information.

Proposition 5.5. Let G = $G_1 \parallel ..... \parallel G_n = (X, \Sigma, \delta, x_0, X_m)$ with $G_i = (X_i, \Sigma, \delta, x_{0,i}, X_{m,i})$, i = 1,....,n. If G satisfies SD controllability point (iii.1) then $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$.

Proof. Assume G satisfies SD controllability point (iii.1).
Let $P_D : (\Sigma^D_{\alpha,G})^* \rightarrow (D_{\alpha,G} \cup \{\alpha, tick\})^*$ be a natural projection.

Let $\alpha \in \Sigma_{hib}, s \in L(D_\alpha(G)) \subseteq (\Sigma^D_{\alpha,G})^*$.

Let $uv\alpha \leq s$ for some $u, v \in (\Sigma^D_{\alpha,G})^*$.

Let $P_\Sigma(u) \in L_{samp}$, and $P_D(v) \in (D_{\alpha,G})^*$.

By Lemma 5.9 it is sufficient to show $P_D(v) = \epsilon$.

We first note that as $P_D(v) \in (D_{\alpha,G})^*, tick \notin Occu(v)$, thus $P_\Sigma(v) \in \Sigma^*_{act}$.

Since $uv\alpha \leq s \in L(D_\alpha(G))$ we have by Lemma 5.6:

$$P_\Sigma(uv\alpha) \in L(G)$$
$$\implies \alpha \in Elig_{L(G)}(P_\Sigma(uv)) \quad (1)$$

Then it holds that $P_\Sigma(u) \in L(G) \cap L_{samp}$, $\alpha \in \Sigma_{hib}, P_\Sigma(v) \in \Sigma^*_{act}$ and $P_\Sigma(uv) \in L(G)$.

Then by definition of SD controllability point (iii.1) and by (1) we have:

$$\alpha \in (Elig_{L(G)}(P_\Sigma(uv)) \cup Occu(P_\Sigma(v))) \cap \Sigma_{hib} = Elig_{L(G)}(P_\Sigma(u)) \cap \Sigma_{hib} \quad (2)$$

Then it also holds for every prefix $v' \leq v$ since $P_\Sigma(v') \in \Sigma^*_{act}$, and $P_\Sigma(uv') \in L(G)$.

We thus have:

$$Elig_{L(G)}(P_\Sigma(uv')) \cup Occu(P_\Sigma(v'))) \cap \Sigma_{hib} = Elig_{L(G)}(P_\Sigma(u)) \cap \Sigma_{hib}$$

We thus have by (2):

$$\alpha \in Elig_{L(G)}(P_\Sigma(uv')) \cup Occu(P_\Sigma(v'))) \cap \Sigma_{hib}$$
$$\subseteq Elig_{L(G)}(P_\Sigma(uv')) \cup Occu(P_\Sigma(v'))$$

Since $P_\Sigma(v') \leq P_\Sigma(v)$ and $P_D(v) \in ({D_{\alpha,G}}^*), \alpha \notin Occu(P_\Sigma(v'))$, we have:

$$\alpha \in Elig_{L(G)}(P_\Sigma(uv'))$$
$$\implies P_\Sigma(uv')\alpha \in L(G)$$

Let $P_i : (\Sigma^D_{\alpha,G})^* \to (\Sigma \cup \{d_{\alpha,G_i}\})^*$ be a natural projection, i = 1,...,n.

We note that as $\Sigma \subseteq \Sigma \cup \{d_{\alpha,G_i}\}$, we have $P_\Sigma \circ P_i = P_\Sigma$, i = 1,....,n. (3)

Let $i \in \{1, ..., n\}$.

We thus have: $P_\Sigma(uv')\alpha \in L(G_i)$.

Also, $s \in L(D_\alpha(G))$ implies $P_i(s) \in L(D_\alpha(G_i))$ and $P_i(uv') \leq P_i(uv) \leq P_i(s) \in L(D_\alpha(G_i))$.

Given that $P_\Sigma(P_i(uv)) = P_\Sigma(uv) \in L(G_i)$, $P_i(uv') \in (\Sigma^D_{\alpha,G_i})^*$; $P_i(uv') \leq P_i(uv)$, and $P_\Sigma(P_i(uv'))\alpha = P_\Sigma(uv')\alpha \in L(G_i)$, we have:

$P_i(uv')d_{\alpha,G_i} \not\leq P_i(uv)$ by Lemma 5.5

Since this holds for every prefix $v' \leq v$, we have:

$d_{\alpha,G_i} \notin Occu(P_i(v))$

Also, $d_{\alpha,G_i} \notin Occu(v)$ because $d_{\alpha,G_i} \in \Sigma^D_{\alpha,G_i}$.

Since this is true for all i $\in \{1, ..., n\}$, we have:

$(\forall d \in D_{\alpha,G})d \notin Occu(v)$.

Since $P_D(v) \in (D_{\alpha,G})^*$ this implies $P_D(v) = \epsilon$ as required. □

Proposition 5.6. Let G = $G_1 \parallel$ ..... $\parallel G_n = (X, \Sigma, \delta, x_0, X_m)$ with $G_i = (X_i, \Sigma, \delta, x_{0,i}, X_{m,i})$, i = 1,....,n. If $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$ then $(\forall s \in L(G) \cap L_{samp})(\forall t \in \Sigma^*_{act})$

$st \in L(G) \Longrightarrow (Elig_{L(G)}(st) \cup Occu(t)) \cap \Sigma_{hib} \subseteq Elig_{L(G)}(s) \cap \Sigma_{hib}$

Proof. Assume $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$.

Let $s \in L(G) \cap L_{samp}$, $t \in \Sigma^*_{act}$ and $st \in L(G)$.

Let $\alpha \in \Sigma_{hib}$ and $\alpha \notin Elig_{L(G)}(s)$.

Sufficient to show $\alpha \notin Elig_{L(G)}(st) \cup Occu(t)$.

Given $st \in L(G)$ and $\alpha \notin Elig_{L(G)}(s)$, it is sufficient to show $\alpha \notin Elig_{L(G)}(st')$ for every prefix $t' \leq t$ such that $t' \in (\Sigma_{act} \setminus \{\alpha\})^*$.

Let $t' \in (\Sigma_{act} \setminus \{\alpha\})^*, t' \leq t$ and assume $\alpha \in Elig_{L(G)}(st')$. (1)

We will show that $\alpha \in Elig_{L(G)}(st')$ creates a contradition.

Since $s \in L(G)$ and $\alpha \notin Elig_{L(G)}(s)$, it follows that:

$(\exists i \in \{1....n\})\ \alpha \notin Elig_{L(G_i)}(s)$.

Consider the string: $sd_{\alpha,G_i}t'\alpha \in \Sigma^{D*}_{\alpha,G_i} \subseteq \Sigma^{D*}_{\alpha,G}$

$\Longrightarrow P_\Sigma(sd_{\alpha,G_i}t'\alpha) = st'\alpha \in L(G_i)$ (by (1)) and $P_\Sigma(s)\alpha = s\alpha \notin L(G_i)$

$\Longrightarrow sd_{\alpha,G_i}t'\alpha \in L(D_\alpha(G_i))$ by Lemma 5.5

Let $P_j : (\Sigma^D_{\alpha,G})^* \to (\Sigma \cup \{d_{\alpha,G_j}\})^*$ be a natural projection, j = 1,....,n.

Also, $P_j(sd_{\alpha,G_i}t'\alpha) = st'\alpha$ for all $j \neq i$. This implies:

$P_j(sd_{\alpha,G_i}t'\alpha) = st'\alpha \in L(D_\alpha(G_j))$ because $st'\alpha \in L(G_j)$ by (1).

Therefore $sd_{\alpha,G_i}t'\alpha \in L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$.

We thus have:

$sd_{\alpha,G_i}t'\alpha \in (\Sigma^D_{\alpha,G})^*, P_\Sigma(s) = s \in L_{samp}, P_D(d_{\alpha,G_i}t') \in (D_{\alpha,G})^*$, and $sd_{\alpha,G_i}t'\alpha \leq sd_{\alpha,G_i}t'\alpha$.

This implies $P_D(d_{\alpha,G_i}t') = d_{\alpha,G_i} = \epsilon$, by Lemma 5.9, which is a contradition.

$\Longrightarrow \alpha \notin Elig_{L(G)}(st')$, as required. □

We will now prove that checking SD controllability point (iii.1) is equivalent to checking the language inclusion properties below.

Theorem 5.2. *Let $G = G_1 \parallel ..... \parallel G_n = (X, \Sigma, \delta, x_0, X_m)$ with $G_i = (X_i, \Sigma, \delta, x_{0,i}, X_{m,i})$, i = 1,....,n. G satisfies SD controllability point (iii.1) iff $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ and $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$*

Proof. To prove the above claim we have to show:

I) If G satisfies SD controllability point (iii.1) then $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ and $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$, and

II) If $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ and $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$, then G satisfies SD Controllability point (iii.1)

Part I) Show: If G satisfies SD controllability point (iii.1) then

$L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ and $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$.

By propositions 5.3 and 5.4 we have $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ and $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$ as required.

Part I complete.

Part II) Show if $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ and $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$, then G satisfies SD Controllability point (iii.1).

Applying propositions 5.5 and 5.6, we have that G satisfies SD Controllability

point (iii.1a) and (iii.1b).

$\Longrightarrow$ G satisfies SD Controllability point (iii.1).

Part II complete.

By Part I and II we conclude G satisfies SD controllability point (iii.1) iff $L(ED_\alpha(G)) \subseteq L(T_{\alpha,G})$ and $L(D_\alpha(G)) \subseteq L(U_{\alpha,G})$ for each $\alpha \in \Sigma_{hib}$ □

5.2.2 SD Controllability Point (iii.2)

SD controllability point (iii.2) (see page 23) states that for a sampled string s, extended by concurrent strings s' and s'' such that s' and s'' have the same occurrence images, the strings ss' and ss'' should have the same future with respect to the system's closed and marked behaviour. That means that the string ss' must be nerode equivalent to string ss'' with respect to the system's closed and marked behaviour. The SD controllability point (iii.2) property is defined in terms of the closed-loop system. For simplicity, we restate the property below in terms of some TDES **G**, where **G** would correspond to our closed-loop system.

Definition 5.6. Let G $= (Q, \Sigma, \delta, q_0, Q_m)$ be a TDES.

G satisfies SD controllability point (iii.2) if

$(\forall s \in L(G) \cap L_{samp})(\forall s', s'' \in L_{conc})ss', ss'' \in L(G) \wedge Occu(s') = Occu(s'')$

$\Rightarrow ss' \equiv_{L(G)} ss'' \wedge ss' \equiv_{L_m(G)} ss''$

As we have stated earlier, a typical system is composed of several TDES. For automatic verification of a system, it is necessary to compute its synchronous product which can easily lead to the state space explosion problem. We can avoid computing the synchronous product of a system by exploiting its modular structure. For the compositional verification of SD controllability point (iii.2) we present a theorem which states that if individual TDES all satisfy our property, then the synchronous product of these TDES also satisfies the SD controllability

point (iii.2) property. We can thus simply check the individual TDES to verify that our TDES system satisfies SD controllability point (iii.2).

Theorem 5.3. *Let* $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$ *and* $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ *be two TDES. If* G_1 *and* G_2 *each satisfy SD controllability point (iii.2) then* $G_1 \parallel G_2$ *satisfies SD controllability point(iii.2).*

Proof. Let $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ be two TDES and $\Sigma = \Sigma_1 \cup \Sigma_2$, and let $P_1 : \Sigma^* \to \Sigma_1^*$ and $P_2 : \Sigma^* \to \Sigma_2^*$ be natural projections. Let $G = G_1 \parallel G_2$, then $L(G) = P_1^{-1}L(G_1) \cap P_2^{-1}L(G_2)$.

Let G_1 and G_2 each satisfy SD controllability point (iii.2). (1)

Let $s \in L(G) \cap L_{samp}, s', s'' \in L_{conc}$. (2)

Assume $ss', ss'' \in L(G)$ and $Occu(s') = Occu(s'')$. (3)

Sufficient to show that $ss' \equiv_{L(G)} ss''$ and $ss' \equiv_{L_m(G)} ss''$.

Let $s_i = P_i(s), s_i' = P_i(s')$ and $s_i'' = P_i(s'')$, $i = 1, 2$.

Let $P_i(Occu(t)) := \{P_i(\sigma) | \sigma \in Occu(t)\} - \{\epsilon\}$, for t $\in \Sigma^*$.

To apply SD controllability point (iii.2) to G_1 and G_2, it is sufficient to show that $s_i \in L(G_i) \cap L_{samp}, s_i', s_i'' \in L_{conc}, Occu(s_i') = Occu(s_i'')$ and $s_i s_i', s_i s_i'' \in L(G_i)$.

From (2) we have $s \in L(G) \cap L_{samp}$

$\Rightarrow s_i = P_i(s) \in L(G_i)$

As s $\in L_{samp}$, either s $= \epsilon$ or s $\in \Sigma^*.tick$.

If s $= \epsilon, P_i(s) = \epsilon \in L_{samp}$.

If s $\in \Sigma^*.tick, P_i(s) \in \Sigma^*.tick$ as $tick \in \Sigma_i$.

We thus have: $P_i(s) \in L_{samp}$.

$\Rightarrow P_i(s) \in L(G_i) \cap L_{samp}$.

From (2) we also have $s', s'' \in L_{conc}$

$\Rightarrow P_i(s'), P_i(s'') \in L_{conc}$, as $tick \in \Sigma_i$.

$\Rightarrow s_i', s_i'' \in L_{conc}$

From (3) we also have $Occu(s') = Occu(s'')$

$\Rightarrow P_i(Occu(s')) = P_i(Occu(s''))$

$\Rightarrow Occu(P_i(s')) = Occu(P_i(s''))$

$\Rightarrow Occu(s_i') = Occu(s_i'')$

From (3) we have $ss', ss'' \in L(G)$

$\Rightarrow P_i(ss'), P_i(ss'') \in L(G_i)$

$\Rightarrow P_i(s)P_i(s'), P_i(s)P_i(s'') \in L(G_i)$

$\Rightarrow s_is_i', s_is_i'' \in L(G_i)$

Now we have $s_i \in L(G_i) \cap L_{samp}, s_i', s_i'' \in L_{conc}, Occu(s_i') = Occu(s_i'')$ and

$s_is_i', s_is_i'' \in L(G_i)$ for i $\in \{1, 2\}$.

By the definition of SD controllability point(iii.2), we have:

$$s_is_i' \equiv_{L(G_i)} s_is_i'' \text{ and } s_is_i' \equiv_{L_m(G_i)} s_is_i'', \text{ for i} \in \{1, 2\}. \tag{4}$$

Now we will show that $ss' \equiv_{L(G)} ss''$ and $ss' \equiv_{L_m(G)} ss''$

Part I) To show that $ss' \equiv_{L(G)} ss''$

Sufficient to show: $(\forall u \in \Sigma^*)ss'u \in L(G)$ iff $ss''u \in L(G)$

Let u $\in \Sigma^*$.

A) Show $ss'u \in L(G) \Rightarrow ss''u \in L(G)$

Assume $ss'u \in L(G)$.

Must show implies $ss''u \in L(G)$

Let $u_i = P_i(u)$ for i $\in \{1,2\}$.

Sufficient to show: $P_i(ss''u) = s_is_i''u_i \in L(G_i)$, i = 1,2

Let i $\in \{1,2\}$.

We note that $ss'u \in L(G)$ implies:

$P_i(ss'u) \in L(G_i)$

$\Rightarrow P_i(s)P_i(s')P_i(u) \in L(G_i)$

$\Rightarrow s_is_i'u_i \in L(G_i)$

From (4) and the definition of Nerode Equivalence we get:

$s_is_i''u_i \in L(G_i)$, as required.

B) Show: $ss''u \in L(G) \Rightarrow ss'u \in L(G)$

Proof is identical to proof of Part A after relabelling s' to s'' and s' to s''.

Part I complete.

Part II) $ss' \equiv_{L_m(G)} ss''$

Sufficient to show: $(\forall u \in \Sigma^*) ss'u \in L_m(G)$ iff $ss'u \in L_m(G)$

Let u $\in \Sigma^*$

A) Show $ss'u \in L_m(G) \Rightarrow ss''u \in L_m(G)$

Assume $ss'u \in L_m(G)$

Must show implies $ss''u \in L_m(G)$

Let $u_i = P_i(u)$ for i $\in$ {1,2}.

Sufficient to show: $P_i(ss''u) = s_i s_i'' u_i \in L_m(G_i)$, i = 1,2.

Let i = $\in$ {1,2}.

We note that $ss'u \in L_m(G)$ implies:

$P_i(ss'u) \in L_m(G_i)$

$\Rightarrow P_i(s)P_i(s')P_i(u) \in L_m(G_i)$

$\Rightarrow s_i s_i' u_i \in L_m(G_i)$

From (4) and the definition of Nerode Equivalence we have:

$s_i s_i'' u_i \in L_m(G_i)$, as required.

B) Show $ss''u \in L_m(G) \Rightarrow ss'u \in L_m(G)$

Proof is identical to proof of Part A after relabelling s' to s'' and s'' to s'.

Part II complete.

From part I and II we can conclude that $ss' \equiv_{L(G)} ss''$ and $ss' \equiv_{L_m(G)} ss''$, as required. □

5.3 SD Controllability Point (iv)

SD controllability point (iv) (see page 23) states that all marked strings must be sampled strings in the closed-loop system. For simplicity, we will express

this property below in terms of some TDES **G**, where **G** would be equal to our closed-loop system.

Definition 5.7. TDES G = $(Q, \Sigma, \delta, q_0, Q_m)$ satisfies SD controllability point(iv) if

$$L_m(G) \subseteq L_{samp} = \Sigma^*.tick \cup \{\epsilon\}$$

Let TDES G = $(Q, \Sigma, \delta, q_0, Q_m)$ with $\omega \notin \Sigma$, and $\Sigma = \Sigma_{act} \dot{\cup} \{tick\}$. Let $\Sigma_T = \Sigma \cup \{\omega\}$. We will now show that we can express SD controllability point (iv) as a language inclusion problem. We first introduce a new TDES **T** = $(X_T, \Sigma_T, \delta_T, x_0, X_{T_m})$, shown in Figure 5.4, that captures the concept of sampled strings. We also introduce a new event $\omega \notin \Sigma$ that takes us from the marked states of **G** to a dump state. Now, let G^ω be such that L(G^ω) = L(G) $\cup L_m(G)\omega$.

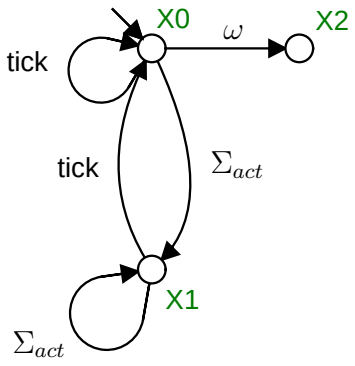


Figure 5.4: TDES T

We first prove the lemma below which we will need for our main theorem.

Lemma 5.10. $L(T) = \Sigma^* \cup L_{samp}\ \omega$

Proof. Let δ_T be the transition function for T.

To prove our claim it is sufficient to show:

1) L(T) $\supseteq \Sigma^* \cup L_{samp}\omega$ and 2) L(T) $\subseteq \Sigma^* \cup L_{samp}\omega$

Part 1) Show: $\Sigma^* \cup L_{samp}\ \omega \subseteq$ L(T)

Let s $\in \Sigma^* \cup L_{samp}\omega$

Must show implies s $\in$ L(T)

Either s $\in \Sigma^*$ or s $\in L_{samp}\omega$.

Case 1.1) s $\in \Sigma^*$

We show by induction on string length that s $\in$ L(T)

Let n $= |s|$.

Inductive Base:

n = 0, s $= \epsilon$

We have $\epsilon \in L(T)$ automatically as TDES T (in Figure 5.4) has an initial state.

Inductive Step:

$n \rightarrow n+1, s = t\alpha$ with $t \in \Sigma^*$ and $\alpha \in \Sigma$

By inductive assumption t $\in$ L(T)

$\Longrightarrow \delta_T(X_0, t)!$

Let X $= \delta_T(X_0, t)!$

Thus we need to show that $\delta_T(X, \alpha)!$

As $\omega \notin \Sigma$ and t $\in \Sigma^*, X \neq X_2$

$\Longrightarrow X = X_0$ or $X = X_1$ [from Figure 5.4]

$\alpha \in \Sigma = \Sigma_{act} \cup \{tick\}$

Clearly (from Figure 5.4) $(\forall \sigma \in \Sigma_{act})\delta_T(X_0, \sigma)!$ and $\delta_T(X_1, \sigma)!$

Also $\delta_T(X_0, tick)!$ and $\delta_T(X_1, tick)!$

$\Longrightarrow \delta_T(X, \alpha)!$

$\Longrightarrow \delta_T(X_0, t\alpha)!$

$\Longrightarrow s \in L(T)$ as required.

Part 1.1 complete.

Case 1.2) s $\in L_{samp}\ \omega$

Let s $\in L_{samp}\omega$. We must show s $\in L(T)$.

From definition of L_{samp} we have

$$\text{s} \in (\Sigma^*.tick \cup \{\epsilon\})\omega$$

We now have 2 cases:

Case a) $\text{s} \in \epsilon\omega = \omega$

Clearly (from Figure 5.4) $\delta_T(X_0, \omega) = X_2$

We thus we have $\omega \in L(T)$.

$\implies s \in L(T)$

Case b) $\text{s} \in \Sigma^*.tick\omega$

$\implies (\exists s' \in \Sigma^*)s = s'tick\omega$

From case 1.1 we know that $s' \in \text{L(T)}$ as $s' \in \Sigma^*$

$\implies (\exists X \in X_T)\delta_T(X_0, s') = X$

As $s' \in \Sigma^*$ and $\omega \notin \Sigma^*, X \neq X_2$, we have:

$X = X_0$ or $X = X_1$ [by Figure 5.4]

Case: $\mathbf{X} = X_0$

Clearly (from Figure 5.4)

$$\delta_T(X_0, tick) = X_0 \text{ and } \delta_T(X_0, \omega) = X_2$$

$\implies s = s'tick\omega \in L(T)$

Case: $\mathbf{X} = X_1$

Clearly (from Figure 5.4)

$$\delta_T(X_1, tick) = X_0 \text{ and } \delta_T(X_0, \omega) = X_2$$

$\implies s = s'tick\omega \in L(T)$

Part 1.2 complete.

Thus from part 1.1 and part 1.2 we can conclude that $\Sigma^* \cup L_{samp}\omega \subseteq \text{L(T)}$.

Part 1 complete.

Part 2) Show: $\text{L(T)} \subseteq \Sigma^* \cup L_{samp}\omega$

Let $\text{s} \in \text{L(T)}$.

$\implies \delta_T(X_0, s)!$

$\Longrightarrow (\exists X \in X_T)\delta_T(X_0, s) = X$

Case a) X = X_0 or X = X_1

By Figure 5.4, X = X_0 or X = X_1 implies that s does not contain ω.

$\Longrightarrow s \in \Sigma^*$ as L(T) $\subseteq \Sigma_T^*$

$\Longrightarrow s \in \Sigma^* \cup L_{samp}\omega$

Case b) X = X_2

$\Longrightarrow (\exists s' \in \Sigma_T^*)s = s'\omega$

$\Longrightarrow \delta_T(X_0, s') = X_0$

$\Longrightarrow s' \in \Sigma^*$

We have two cases: $s' = \epsilon$ or $s' \neq \epsilon$

Case b.1) $s' = \epsilon$

$\Longrightarrow s = \epsilon\omega = \omega \in L_{samp}\omega$

Case b.2) $s' \neq \epsilon$

$\Longrightarrow |\ s'\ | \neq 0$

$\Longrightarrow (\exists s'' \in \Sigma^*)(\exists \sigma \in \Sigma)s' = s''\sigma$ (1)

$\Longrightarrow (\exists X' \in X_T)\delta_T(X_0, s'') = X'$

$\Longrightarrow X' = X_0$ or $X' = X_1$ as s'' contains no ω

Case: $X' = X_0$

Clearly (by Figure 5.4) only *tick* allows $\delta_T(X_0, \sigma) = X_0$

We thus have $\sigma =$ *tick*.

$\Longrightarrow s''tick \in \Sigma^*tick \subseteq L_{samp}$

$\Longrightarrow s' = s''\sigma \in L_{samp}$ by (1)

$\Longrightarrow s = s'\omega \in L_{samp}\omega$, as required

Case: $X' = X_1$

Clearly (by Figure 5.4) only *tick* allows $\delta_T(X_1, \sigma) = X_0$

We thus have $\sigma =$ tick.

$\Longrightarrow s''tick \in \Sigma^*tick \subseteq L_{samp}$

$\Longrightarrow s' = s''\sigma \in L_{samp}$ by (1)

$\Longrightarrow s = s'\omega \in L_{samp}\omega$ as required

Part 2 complete.

Thus by part 1 and 2 we conclude L(T) = $\Sigma^* \cup L_{samp}\omega$ □

We will now prove that checking SD controllability point (iv) is equivalent to checking the language inclusion property below.

Theorem 5.4. *G satisfies SD controllability point (iv) iff* $L(G^\omega) \subseteq L(T)$

Proof. We must show that:

I) G satisfies SD controllability point (iv) $\Rightarrow L(G^\omega) \subseteq L(T)$ and

II) $L(G^\omega) \subseteq L(T) \Rightarrow$ G satisfies controllability point (iv).

Part I) Show: G satisfies SD controllability point (iv) $\Rightarrow L(G^\omega) \subseteq L(T)$

Assume G satisfies SD controllability point (iv). (1)

Let $u \in L(G^\omega)$

We will now show that this implies $u \in L(T)$.

Applying the definition of G^ω, we have:

$u \in L(G) \cup L_m(G)\omega$

$\Rightarrow u \in L(G)$ or u $\in L_m(G)\omega$

Case i) $u \in L(G) \subseteq \Sigma^*$

As $\Sigma^* \subseteq L(T)$ (by Lemma 5.10), we have u $\in L(T)$ as required.

Case ii) $u \in L_m(G)\omega$

$\Rightarrow (\exists s \in L_m(G))$ such that u = $s\omega$

$\Rightarrow s \in L_m(G)$

From (1) we have $s \in L_{samp}$.

$\Rightarrow s\omega \in L_{samp}\omega$

As u =sω and $L_{samp}\omega \subseteq L(T)$ by Lemma 5.10, we can conclude $u \in L(T)$ as required.

Part II) Show: $L(G^\omega) \subseteq L(T) \Rightarrow$ G satisfies SD controllability point (iv)

Let $L(G^\omega) \subseteq L(T)$. (2)

We will show that this implies $L_m(G) \subseteq L_{samp}$.

Let $u \in L_m(G) \subseteq \Sigma^*$.

Will show implies u $\in L_{samp}$.

By the definition of $G^\omega, L(G^\omega) = L(G) \cup L_m(G)\omega$

$\Rightarrow u\omega \in L_m(G)\omega$

As $L_m(G)\omega \subseteq L(G^\omega)$ (by the definition of $L(G^\omega)$) and $L(G^\omega) \subseteq L(T)$ (by (2)) we conclude: $u\omega \in L(T)$

Since $L(T) = \Sigma^* \cup L_{samp}\omega$ and $\omega \notin \Sigma^*$, we have that : $u\omega \notin \Sigma^*$

$\Rightarrow u\omega \in L_{samp}\omega$

$\Rightarrow u \in L_{samp}$ as required.

From part I and II, we can conclude that G satisfies SD controllability point (iv) iff $L(G^\omega) \subseteq L(T)$. □

CHAPTER 6

Algorithms

In this chapter we present the algorithms to check the required conditions for sampled data supervisory control which are plant completeness, ALF, S-singular prohibitable behaviour, proper time behaviour and SD controllability. Essentially, most of the algorithms presented in this chapter use existing compositional verification algorithms for controllability, nonblocking and language inclusion. We will discuss the construction/modification required for the needed automata to adapt the SD properties as described in Chapters 4 and 5, and specify which algorithm we will use to verify each property. The only new algorithm that we introduce in this chapter is for verifying SD controllability point (iii.2). We present both a monolithic algorithm and an incremental algorithm for this property. All algorithms have been implemented in Java and are a part of the DES research tool *Supremica* [Sup].

Typically we will be evaluating a system $\mathbf{G} = G_1 \parallel ..... \parallel G_n = (Y, \Sigma, \delta, y_0, Y_m)$ composed of the synchronous product of n TDES. For ease of reference we will specify $\mathbf{G}$ as the set of G_i that it is composed from and use the notation $G_i \in \mathbf{G}$ to represent one of these TDES.

In Chapter 3, when we discussed our system comprised of our TDES plant $\mathbf{G}$, and supervisor $\mathbf{S}$, we assumed that they were both defined over the common event set Σ, and that our closed-loop system was $\mathbf{G} \times \mathbf{S}$. However, *Supremica* only uses the synchronous product operator so our closed-loop system will be $\mathbf{G} \parallel \mathbf{S}$ and $\mathbf{G}$ and $\mathbf{S}$ may be defined over different alphabets.

In this chapter, we will assume **G** is defined over event set Σ_G, and **S** is defined over event set Σ_S. We define our system's event set to be $\Sigma := \Sigma_G \cup \Sigma_S$. To apply the definitions from Chapter 3 (that assume both **G** and **S** to be defined over Σ), we can construct TDES G' and S' from **G** and **S** by adding selfloops of any events in $\Sigma \setminus \Sigma_G$ to **G**, and in $\Sigma \setminus \Sigma_S$ to **S**. We can then apply the definitions to G' and S' directly.

6.1 Plant Completeness

This property can be evaluated using the standard untimed controllability algorithm, after relabeling a few variables. We relabel the plants as supervisors, supervisors as plants, all prohibitable events as uncontrollable events, and all uncontrollable events as controllable events. We then check that our supervisor "is controllable for our plant" using the existing incremental controllability algorithm [BMM04].

6.2 ALF

We can verify that our closed system is activity-loop free by using the existing modular control-loop checker algorithm [MM06], after relabeling a few variables. We relabel the event tick as an uncontrollable event and all other events as controllable events. We then check that **G** $\parallel$ **S** has no control-loops using the existing control-loop checker algorithm [MM06]. By **Lemma 4.1** in **Section 4.2**, this implies the system is ALF.

6.3 S-singular Prohibitable Behaviour

This algorithm (Algorithm 1) checks that our plant $\mathbf{G} = G_1 \parallel .... \parallel G_n$ has S-singular prohibitable behaviour, where **S** is our supervisor. The algorithm constructs for each $\sigma \in \Sigma_{hib}$, a new TDES G_i^σ (i = 1,....,n) and a requirement TDES T^σ, as described in **Section 4.3**. Then it checks if $P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma)$ (for each $\sigma \in \Sigma_{hib}$) by using the modular language inclusion check algorithm [BMM04][1]. If the language inclusion check is not satisfied for any σ, it returns false. If all checks are satisfied it returns true and we can conclude by **Theorem 4.1**, that **G** has S-singular prohibitable behaviour.

Algorithm 1 $S - singularProhibitableBehaviour(G, S)$

1: **for all** $\sigma \in \Sigma_{hib}$ **do**
2: Create event $e_\sigma \notin \Sigma$.
3: **for all** $G_i \in$ G **do**
4: Construct G_i^σ from G_i by using the Construct G^σ Algorithm (Algorithm 2) and add it to G^σ (i.e. $G^\sigma = G_1^\sigma \parallel ..... \parallel G_n^\sigma$).
5: **end for**
6: Construct T^σ as shown in Figure 4.1 on page 29.
7: Use the *Language Inclusion Check Algorithm* to check if $P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma)$.
8: **if** $\neg(P_{E_\sigma}(L(G^\sigma \parallel S)) \subseteq L(T^\sigma))$ **then**
9: **return** False
10: **end if**
11: **end for**
12: **return** True

6.4 Proper Time Behaviour

This algorithm (Algorithm 3) verifies that the TDES system has proper time behaviour by constructing a TDES T_Υ as per **Figure 4.2** with $\Upsilon = \Sigma_u \cup \{tick\}$, and the modified components G_i^ω (see **Section 4.4**) for each TDES plant G_i (i =

[1]The algorithm actually evaluates the equivalent $L(G^\sigma \parallel S) \subseteq P_{E_\sigma}^{-1}(L(T^\sigma))$ as adding self-loops to T^σ is easier than evaluating a natural projection.

Algorithm 2 Construct $G^{\sigma}(G, \sigma)$

1: Create a new TDES G^{σ} with the states, transitions and event set of TDES G.
2: Create a new state y' and add it to the states of G^{σ}
3: Add the event e_{σ} to the event set of G^{σ}
4: **for all** transitions t in G of the form (y_1, σ', y_2) **do**
5: **if** $(\sigma' = \sigma)$ **then**
6: Create a new transition (y_1, e_{σ}, y') and add it to the transitions of G^{σ}
7: **end if**
8: **end for**
9: **return** G^{σ}

1, ..., n) that our plant $\mathbf{G} = G_1 \parallel ..... \parallel G_n$ is composed from. It then constructs $G^{\omega} = G_1^{\omega} \parallel .... \parallel G_n^{\omega}$. Then we check if $T_{\Upsilon} \parallel G^{\omega}$ is nonblocking by running the existing compositional conflict checker algorithm [FM09]. We can then conclude, by **Theorem 4.2** and **Lemma 4.4**, that **G** has proper time behaviour.

Algorithm 3 ProperTimeBehaviour(G)

1: **for all** $G_i \in$ G **do**
2: Construct G_i^{ω} by marking all states of G_i and add it to G^{ω}.
3: **end for**
4: Construct T_{Υ} as shown in Figure 4.2 on page 39 with $\Upsilon = \Sigma_u \cup \{tick\}$
5: Use the compositional conflict checker to check if $G^{\omega} \parallel T_{\Upsilon}$ is nonblocking.
6: **if** $G^{\omega} \parallel T_{\Upsilon}$ is nonblocking **then**
7: **return** True
8: **else**
9: **return** False
10: **end if**

6.5 SD Controllability Point (ii)

We verify SD controllability point(ii) in two parts. If the TDES system passes both tests, we conclude it satisfies SD controllability point (ii) as per **Theorem 5.1**. Both algorithms are presented below.

6.5.1 SD Controllability Point (ii.a)

This algorithm (Algorithm 4) checks that the system satisfies SD controllability point (ii.a) (see page 43) by constructing a requirements TDES T_Σ^E, the modified plant components $E_{tick}(G_i)$ and the modified supervisor components $E_{tick}(S_i)$ for the plant TDES $\mathbf{G} = G_1 \parallel ... \parallel G_n$ and supervisor TDES $\mathbf{S} = S_1 \parallel ..... \parallel S_m$, as described in **Section 5.1.1**. It then constructs $E_{tick}(G) = E_{tick}(G_1) \parallel ..... \parallel E_{tick}(G_n)$ and $E_{tick}(S) = E_{tick}(S_1) \parallel ..... \parallel E_{tick}(S_m)$. Then it uses the incremental language inclusion checker [BMM04] to check if $L(E_{tick}(G) \parallel E_{tick}(S)) \subseteq L(T_\Sigma^E)$. We can then conclude by **Proposition 5.1** that the system satisfies SD controllability point (ii.a).

Algorithm 4 SD Controllability point (ii.a)

1: Create event $e_{tick} \notin \Sigma$
2: **for all** $A_i \in$ G $\parallel$ S **do**
3: Construct $E_{tick}(A_i)$ by adding a selfloop with the event e_{tick} to the states of A_i with tick enabled and add it to $E_{tick}(G \parallel S)$.
4: **end for**
5: Construct T_Σ^E as shown in Figure 5.1 on page 50.
6: Use the *Language Inclusion Check Algorithm* to check if $L(E_{tick}(G) \parallel E_{tick}(S)) \subseteq L(T_\Sigma^E)$.
7: **if** $\neg(L(E_{tick}(G) \parallel E_{tick}(S)) \subseteq L(T_\Sigma^E))$ **then**
8: **return** True
9: **else**
10: **return** False
11: **end if**

6.5.2 SD Controllability Point (ii.b)

This algorithm verifies that our system satisfies SD controllability point (ii.b) (see page 43). It constructs the TDES T_Υ, as per **Figure 4.2** with $\Upsilon = D_{tick,G} \cup \Sigma_{hib} \cup \{tick\}$, and the modified components $D_{tick}(G_i^\omega)$ of TDES plant $\mathbf{G} = G_1 \parallel ..... \parallel G_n$, as described in **Section 5.1.2**. Next it constructs $D_{tick}(G^\omega) = D_{tick}(G_1^\omega) \parallel$

.... $\|$ $D_{tick}(G_n^\omega)$. Then it checks if $D_{tick}(G^\omega)$ $\|$ S $\|$ T_Υ, where S is our supervisor, is nonblocking by running the compositional conflict checker algorithm [FM09] on the system. We can then conclude by **Proposition 5.2** and **Theorem 4.2** that the system satisfies SD controllability point (ii.b).

Algorithm 5 SD Controllability point (ii.b)

1: **for all** $G_i \in$ G **do**
2: Create event $d_{tick,G_i} \notin \Sigma$
3: Construct $D_{tick}(G_i)$ by adding a selfloop with the event d_{tick,G_i} to the states of G_i with tick disabled.
4: Construct $D_{tick}(G_i^\omega)$ by marking all states of $D_{tick}(G_i)$ and add it to $D_{tick}(G^\omega)$.
5: **end for**
6: Construct T_Υ as shown in Figure 4.2 on page 39 with $\Upsilon = \{d_{tick,G_1}, ..., d_{tick,G_n}\} \cup \{tick\} \cup \Sigma_{hib}$.
7: Use the compositional conflict checker to check if $D_{tick}(G^\omega)$ $\|$ S $\|$ T_Υ is nonblocking.
8: **if** $D_{tick}(G^\omega)$ $\|$ S $\|$ T_Υ is nonblocking **then**
9: **return** True
10: **else**
11: **return** False
12: **end if**

6.6 SD Controllability Point (iii.1)

The SD controllability point(iii.1) is verified in two parts. If the system passes both tests, we can conclude it passes SD controllability point (iii.1) by **Theorem 5.2**. Both algorithms are presented below.

6.6.1 SD Controllability Point (iii.1a) Algorithm

The first algorithm (Algorithm 6) checks that our TDES system satisfies SD controllability point (iii.1a) (see page 64) by constructing a requirements TDES $T_{\alpha,G\|S}$ as shown in **Figure 5.2** and the modified components $ED_\alpha(A_i)$ for **G** $\|$ **S**

where $\mathbf{G} = G_1 \parallel .... \parallel G_n$ and $\mathbf{S} = S_1 \parallel ....S_m$ and $\alpha \in \Sigma_{hib}$. For each component A_i that makes up $\mathbf{G} \parallel \mathbf{S}$, it constructs $ED_\alpha(A_i)$ using the method described in **Section 5.2.1.1**. It then creates $ED_\alpha(G \parallel S)$ as the synchronous product of all the $ED_\alpha(A_i)$. It then calls the incremental language inclusion checker algorithm [BMM04] to check if $L(ED_\alpha(G \parallel S)) \subseteq L(T_{\alpha,G\|S})$ for each $\alpha \in \Sigma_{hib}$. We can then conclude by **Proposition 5.3** that the system satisfies SD controllability point (iii.1a).

Algorithm 6 SD Controllability point (iii.1a)

```
1: for all α ∈ Σ_hib do
2:     Create event e_α ∉ Σ
3:     for all A_i ∈ G ∥ S do
4:         Create event d_{α,A_i}
5:         Construct ED_α(A_i) by adding a selfloops with event e_α to the states
           of A_i with α enabled and selfloops with the event d_{α,A_i} to the states
           of A_i with α disabled.
6:         Add ED_α(A_i) to ED_α(G ∥ S).
7:     end for
8:     Construct T_{α,G∥S} as shown in Figure 5.2 on page 70.
9:     Use the Language Inclusion Check Algorithm to check if L(ED_α(G ∥
       S)) ⊆ L(T_{α,G∥S})
10:    if ¬(L(ED_α(G ∥ S)) ⊆ L(T_{α,G∥S})) then
11:        return False
12:    end if
13: end for
14: return True
```

6.6.2 SD Controllability Point (iii.1b) Algorithm

This algorithm (Algorithm 7) checks that the system satisfies SD controllability point (iii.1b) (see page 64) by constructing a requirements TDES $U_{\alpha,G\|S}$, as shown in **Figure 5.3**, and the modified components $D_\alpha(A_i)$ for TDES $G \parallel S$ where $\mathbf{G} = G_1 \parallel .... \parallel G_n$, $\mathbf{S} = S_1 \parallel ....S_m$, and $\alpha \in \Sigma_{hib}$. For each component A_i that makes up $\mathbf{G} \parallel \mathbf{S}$, the algorithm constructs $D_\alpha(A_i)$ using the method described

in **Section 5.2.1.2**. It then creates $D_\alpha(G \parallel S)$ as the synchronous product of all the $D_\alpha(A_i)$. Then it uses the incremental language inclusion checker to check if $L(D_\alpha(G \parallel S)) \subseteq L(U_{\alpha,G\|S})$ for each $\alpha \in \Sigma_{hib}$. We can then conclude by **Proposition 5.5** that our system satisfies SD controllability point (iii.1b).

Algorithm 7 SD Controllability point (iii.1b)

```
1: for all α ∈ Σ_hib do
2:     for all A_i ∈ G ‖ S do
3:         Create event d_{α,A_i}
4:         Construct D_α(A_i) by adding a selfloop with the event d_{α,A_i} to the
           states of A_i with α disabled and then add D_α(A_i) to D_α(G ‖ S).
5:     end for
6:     Construct U_{α,G‖S} as shown in Figure 5.3 on page 78.
7:     Use the Language Inclusion Check Algorithm to check if L(D_α(G ‖ S)) ⊆
       L(U_{α,G‖S}).
8:     if ¬(L(D_α(G ‖ S)) ⊆ L(U_{α,G‖S})) then
9:         return False
10:    end if
11: end for
12: return True
```

6.7 SD Controllability Point (iii.2)

We introduce two algorithms here for verifying SD controllability point (iii.2). Since there is no existing algorithm in Supremica [Sup] to check the nerode equivalence of concurrent strings, first we develop a 'monolithic checker' for checking nerode equivalence of concurrent strings in a TDES. This algorithm constructs the synchronous product of the TDES components in the system and then verifies if the system satisfies SD controllability point(iii.2).

The second algorithm is developed using an incremental approach for verifying the nerode equivalence of concurrent strings, it verifies that the system satisfies SD controllability point (iii.2) by using an incremental approach guided by counterexamples similar to the incremental controllability checker in [BMM04].

Both algorithms are presented below.

6.7.1 Monolithic Verification Algorithm

This algorithm (Algorithm 8) checks point (iii.2) of the SD controllability definition for TDES $\mathbf{A} = (Y, \Sigma, \delta, y_0, y_m)$ using the definition given in **Section 5.2.2**. It analyses the concurrent behaviour of $\mathbf{A}$ by starting at sampled state y_0, and determining all the concurrent strings leaving y_0, noting the sampled states these strings lead to. If two concurrent strings leaving y_0 have the same occurrence image, point (iii.2) requires that they must go to the λ-equivalent states.

The above process is repeated for all reachable sampled states, and if they all pass, then we can conclude that $\mathbf{A}$ satisfies SD controllability point (iii.2). Otherwise we can conclude that it does not.

If TDES $\mathbf{A}$ fails point (iii.2), Algorithm 8 generates two counterexamples, *ce1* and *ce2*. They are composed of strings $s \in L_{samp}, s', s'' \in L_{conc}$ and u $\in \Sigma^*$ with ce1 = $ss'u$ and ce2 = ss''. Strings s, s' and s'' correspond to the strings in the definition of SD controllability point (iii.2) given on page 86. We have that ss', ss'' are accepted by TDES $\mathbf{A}$, s' and s'' have the same occurrence image, but ss' and ss'' are not nerode equivalent with respect to L(A) or L_m(A). String u is an example that causes nerode equivalence to fail. For example, if equivalence for L(A) failed, then $ss'u \in$ L(A) but $ss''u \notin$ L(A). The counterexamples will be used in Algorithm 13, the incremental verification algorithm. The two counterexamples, ce1 and ce2, are flagged to apply to either L(A) or L_m(A).

With respect to the state based algorithm presented here, sampled string s corresponds to the current sampled state being processed by Algorithm 8. Concurrent strings s' and s'' correspond to two sampled states added to the set Y_{equ} in Algorithm 10. Finally string u will be determined when a state pair fails the test at **line 11** or **lines 15-16** in Algorithm 12. If **line 11** fails, string u takes

us to the marked state, if **lines 15-16** fail, string u takes us to state reached by the $\sigma \in \Sigma$ transition.

We first need to define some variables that we will use in Algorithm 8. Let $Y_{SF} \subseteq Y$ represent sampled states found by the algorithm. Let $Y_{SP} \subseteq Y$ represent sampled states found and waiting to be processed. Let $pNerFail \subseteq Pwr(Y)$ be the subset of Y that we need to check for λ-equivalence. We will use the notation "pop" to remove an element from a set, and "push" to add an element. To determine which concurrent strings are possible from our sampled state, we not only need to know which state we have reached, but the occurrence image of the string that brought us to that state. We will use the concept of nodes, where a node is a tuple $(\Sigma', y) \subseteq Pwr(\Sigma) \times$ Y. The set Σ' represents the occurrence image of the string that brought us to state y.

For our algorithm, let $CN \subseteq Pwr(\Sigma) \times$ Y be the set of nodes that represent

Algorithm 8 Monolithic SD Controllability point (iii.2)(A)

```
1: Y_SF := Y_SP := {y_0}
2: pNerFail := ∅
3: while (Y_SP ≠ ∅) do
4:       y := pop (Y_SP)
5:       do AnalyseSampleState(y, Y_SF, Y_SP, CN)
6:       do CheckNerodeCells(CN,pNerFail)
7: end while
8: if (pNerFail = ∅) then
9:       return True
10: end if
11: controllable := RecheckNerodeCells(pNerFail)
12: if (controllable) then
13:       return True
14: else
15:       create Counterexamples
16:       return False
17: end if
```

sampled states that were reached from the current sampled state. This means the occurrence image will correspond to a concurrent string.

The subalgorithms used in Algorithm 8 are defined in the following sections. We also assume that TDES **A** is available to all subalgorithms. The algorithms for creating a counterexample can be found in [BMM04].

6.7.1.1 AnalyzeSampledState Algorithm

We first need to define some variables that we will use in Algorithm 9. Let FN $\subseteq Pwr(\Sigma)\times$ Y be the set of nodes that we have already found. Let PN $\subseteq Pwr(\Sigma)\times$ Y be the set of nodes that we have found, but not yet processed.

Algorithm 9 AnalyzeSampledState(y, Y_{SF}, Y_{SP},CN)

```
1:  FN:= PN:= {(∅, y)}, CN = ∅
2:  while (PN ≠ ∅) do
3:      (Σ', y') = pop(PN);
4:      for all σ ∈ Σ do
5:          if (δ(y', σ)!) then
6:              y'' := δ(y', σ)
7:              Σ'' := Σ' ∪ {σ}
8:              if (σ = tick) then
9:                  push ((Σ'', y''), CN)
10:                 if (y'' ∉ Y_SF) then
11:                     push(y'', Z_SF)
12:                     push(y'', Z_SP)
13:                 end if
14:             else
15:                 if ((Σ, y'') ∉ FN) then
16:                     push((Σ'', y''), FN)
17:                     push((Σ'', y''), PN)
18:                 end if
19:             end if
20:         end if
21:     end for
22: end while
```

The *analyzeSampledState* algorithm (Algorithm 9) determines which concurrent strings are possible from sampled state y, and to which sampled states the

strings lead. Actually it is sufficient for us to track the occurrence image of our strings, and use the strings themselves thus reducing complexity.

Algorithm 9 keeps track of the nodes ($(\Sigma', y') \subseteq Pwr(\Sigma) \times$ Y) that we have found, not the actual state. The reason is that a state could be reached by multiple strings. We start with the node $(\emptyset, y)$ which corresponds to Occu(ϵ) and our starting sampled state, y, and loop until there are no more nodes to process. As Σ and Y are finite sets, it follows that Pwr(Σ) $\times$ Y is finite. As **lines 14-17** ensure a node is added at most once to *PN*, we are assured our algorithm will process a node at most once, thus we will terminate in a finite number of steps.

For each node (Σ', y') added to *PN*, we determine **(lines 4-19)**, which events are possible at state y'. If event $\sigma \in \Sigma$ is possible at y', we determine the next state y'' and the occurrence image of the string that takes us to y'' (**line 7**). If σ is the *tick* event, then y'' is a sampled state. We don't add node (Σ'', y'') to *PN* as the *tick* terminates our concurrent string, but we add (Σ'', y'') to *CN* (**line 9**). We add y'' to Y_{SF} and Y_{SP} only if y'' has not been found already. If σ is not *tick*, we add node (Σ'', y'') to *PN* only if we have not already seen (Σ', y'').

Finally, we note that when Algorithm 9 terminates, *CN* will contain the set of all sampled states reached from y, and the occurrence image of the concurrent strings that reached them. This will be used by the calling routine to determine which states point (iii.2) requires to be λ-equivalent.

6.7.1.2 CheckNerodeCells Algorithm

The CheckNerodeCells algorithm (Algorithm 10) examines nodes (Σ', y') in *CN*, to determine which states point (iii.2) requires to be λ-equivalent. *CN* contains all the sampled states reached from the concurrent sampled states and the occurrence images of the concurrent string that reached that state.

Algorithm 10 examines each $(\Sigma', y') \in CN$, and groups into a set (Y_{equ}) all of

the sampled states with the same occurrence image (**line 2-13**). If the resulting set contains more than one distinct state, the Y_{equ} is added to *pNerFail*. Each $Y_{equ} \in$ *pNerFail* will be later tested for λ-equivalence.

Algorithm 10 CheckNerodeCells(CN, pNerFail)

```
1:  while (CN ≠ ∅) do
2:      (Σ′, y′) := pop(CN)
3:      Y_equ := {y′}
4:      SameCell := True
5:      for all (Σ″, y″) ∈ CN do
6:          if (Σ′ = Σ″) then
7:              push(y″, Y_equ)
8:              CN := CN - (Σ″, y″)
9:              if (y′ ≠ y″) then
10:                 SameCell := False
11:             end if
12:         end if
13:     end for
14:     if (¬SameCell) then
15:         push(Y_equ,pNerFail)
16:     end if
17: end while
```

6.7.1.3 ReCheckNerodeCells Algorithm

We first define variables $Y_{vis} \subseteq Y \times Y$. This variable represents state pairs that are required to be λ-equivalent. If algorithm *ReCheckNerodeCells* returns "True", then it means they have all been found to be equivalent.

Algorithm 11 checks the state subsets stored in *pNerFail* to see if the states in a given subset are λ-equivalent to each other. If *pNerFail* is not empty, we call the subalgorithm *RecheckNerodeCell* for each element Y_{equ} in *pNerFail*. If *RecheckNerodeCell* returns true, this means that the set Y_{Vis} will contain the state pairs encountered so far which satisfy the definition of λ-equivalent. If

Algorithm 11 ReCheckNerodeCells(pNerFail)

```
1: Y_Vis := ∅
2: while (pNerFail ≠ ∅) do
3:     Y_equ := pop(pNerFail)
4:     Pass := ReCheckNerodeCells(Y_equ, Y_vis)
5:     if (¬Pass) then
6:         return False
7:     end if
8: end while
9: return True
```

RecheckNerodeCell returns false, the system fails to satisfy SD controllability Point (iii.2).

6.7.1.4 RecheckNerodeCell Algorithm

We first define variables $Y_{pend} \subseteq Y \times Y$. This variable represents state pairs that we need to test for λ-equivalence (see λ-equivalence definition on page 9).

For each set of states (Y_{equ}) that Algorithm 12 is passed as a parameter, we check that these states are λ-equivalent to each other. The algorithm starts by removing a state y_1 from the set Y_{equ} and initializing the pending list Y_{pend} to the empty set. While Y_{equ} is not empty, the algorithm adds the state pairs of the form (y_1, y_2) to Y_{pend} where y_2 is removed from Y_{equ}. We also store the state pair (y_1, y_2) and (y_2, y_1) both in the visited list, Y_{Vis}, since λ is an equivalence relationship, thus symmetric. We note that the state pairs in Y_{vis} represents states already shown to be λ-equivalent from a previous call to Algorithm 12, or represents state pairs that must be shown to be λ-equivalent, if the system satisfies SD controllability point (iii.2). The set Y_{vis} also is used to ensure a state pair is only added once to Y_{pend}, ensuring that the algorithm terminates in a finite number of steps.

Our approach to checking that the state pairs in Y_{pend} are λ-equivalent is

Algorithm 12 ReCheckNerodeCell(Y_{eq}, Y_{Vis})

```
 1: y_1 := pop(Y_equ)
 2: Y_pend = ∅
 3: while (Y_equ ≠ ∅) do
 4:       y_2:= pop (Y_eq)
 5:       push ((y_1, y_2), Y_Vis)
 6:       push ((y_2, y_1), Y_Vis)
 7:       push ((y_1, y_2), Y_pend)
 8: end while
 9: while Y_pend ≠ ∅ do
10:       (y_1, y_2) := pop(Y_pend)
11:       if ((y_1 ∈ Y_m ∧ y_2 ∉ Y_m) ∨ (y_2 ∈ Y_m ∧ y_1 ∉ Y_m)) then
12:            return False
13:       end if
14:       for all σ ∈ Σ do
15:            if (¬(¬δ(y_1, σ)! ∧ ¬δ(y_2, σ)!)) then
16:                 if (¬δ(y_1, σ)! ∨ ¬δ(y_2, σ)¬!) then
17:                      return False
18:                 end if
19:                 y'_1 = δ(y_1, σ)
20:                 y'_2 = δ(y_2, σ)
21:                 if ((y'_1 ≠ y'_2) ∧ ((y'_1, y'_2) ∉ Y_Vis)) then
22:                      push ((y'_1, y'_2), Y_Vis)
23:                      push ((y'_2, y'_1), Y_Vis)
24:                      push ((y'_1, y'_2), Y_pend)
25:                 end if
26:            end if
27:       end for
28: end while
29: return True
```

to attempt to disprove that the pairs are λ-equivalent. If we can't disprove this, then they must be λ-equivalent.

Examining the λ-equivalent definition, we see that if state y_1 and y_2 are λ-equivalent, then they must be both marked, or neither. **Lines 11-13** test this, returning false if they fail the test.

The next test for λ-equivalence is that for each $\sigma \in \Sigma$, there must be a σ transition at both states, or at neither. **Lines 14-18** check this, returning false

if the test fails.

The final test is that if there is a $\sigma \in \Sigma$ transition at both y_1 and y_2, then their next states y_1' and y_2' must be λ-equivalent. We test this by adding state pair (y_1', y_2') to Y_{pend}. To make sure we terminate, we only add (y_1', y_2') to Y_{pend} if (y_1', y_2') are already in Y_{vis}. If they are in Y_{vis}, then they are known already to be equivalent from a previous call to Algorithm 12 , or they have already been added to Y_{pend}. This is handled by **lines 19-25**.

If states y_1 and y_2 are not λ-equivalent, then at some point a set of states must be added to Y_{pend} that will fail the test at **lines 11-13** or **lines 14-18**, causing the algorithm to return false, ending the point (iii.2) check. If we empty Y_{pend} and find no state pairs that failed our checks, then all state pairs that we added to Y_{vis} must be λ-equivalent. This means we can return true for Y_{equ}, and the state pairs in Y_{vis} do not need to be tested again, and can thus be reused in future calls for the current automata.

6.7.2 Incremental Verification Algorithm

The incremental verification algorithm (Algorithm 13) presented here is developed on the lines of the incremental verification algorithms for controllability and language inclusion guided by counter-examples [BMM04].

For plant $\mathbf{G} = G_1 \parallel .... \parallel G_n$ and our supervisor $\mathbf{S} = S_1 \parallel .... \parallel S_m$, the algorithm verifies that $\mathbf{G} \parallel \mathbf{S}$ satisfies SD controllability point (iii.2) using Theorem 5.3. Theorem 5.3 says that if TDES G_1 and G_2 satisfy point (iii.2) then $G_1 \parallel G_2$ satisfy point (iii.2) also. This means that if we partition the automata that make up $\mathbf{G} \parallel \mathbf{S}$ into subsystems, and show that each subsystem satisfies point (iii.2), then $\mathbf{G} \parallel \mathbf{S}$ also satisfies point (iii.2).

We first need to define some variables that we will use in Algorithm 13. Let **UnProcessed** be the set of TDES that we have yet to process. Let **Processed**

be the set of TDES that we have already processed. Let **Current** be the set of TDES that we are currently testing. All three are subsets of the set of TDES that comprise our plant G and supervisor S. We also use *Sync*(**Current**) to stand for the synchronous product of the TDES contained in **Current**.

Algorithm 13 SD Controllability point (iii.2)(G ‖ S)

```
1: UnProcessed = ∅, Processed = ∅.
2: for all A_i ∈ G ‖ S do
3:     push(A_i, UnProcessed)
4: end for
5: while (Unprocessed ≠ ∅) do
6:     A_i := pop(UnProcessed)
7:     Current := {A_i}
8:     Use Algorithm 8 to check if Sync(Current) satisfies point (iii.2)
9:     while (Sync(Current) fails point(iii.2)) do
10:        Get Counterexamples ce1=ss'u and ce2=ss'' from Algorithm 8
11:        if (there exists A_j ∈ ((Unprocessed ∪ Processed) \ Current) such
           that A_j does not accept both ce1 and ce2) then
12:            push(A_j,Current)
13:        else
14:            return False
15:        end if
16:        Use Algorithm 8 to check if Sync(Current) satisfies point (iii.2)
17:    end while
18:    UnProcessed := Unprocessed \ Current
19:    push(Current,Processed)
20: end while
21: return True
```

Algorithm 13 selects unprocessed TDES component A_i and runs the monolithic checker (Algorithm 8) to see if A_i satisfies point (iii.2) **(lines 6-8)**. If the check fails, Algorithm 8 provides counterexamples ce1 := $ss'u$ and ce2 := ss'', which are described in **Section 6.7.1**. The algorithm then tries to determine another TDES A_j in the system that rejects either counterexamples. If it finds one, it's added to the set **Current**, and the algorithm checks if *Sync*(**Current**) now satisfies point (iii.2). If no such automata exists, then it's clear from the definition of the counterexamples in **Section 6.7.1** that **G ‖ S** must also fail

point (iii.2), by the same counterexamples.

Heuristics are used to choose which TDES A_j to add to **Current**. For our algorithm, we have used the same methods and heuristics as the algorithms presented in [BMM04] for controllability and language inclusion; therefore we do not give the details of the heuristics here. The details can be found in [BMM04].

6.8 SD Controllability Point (iv)

This algorithm (Algorithm 14) checks that the TDES system satisfies SD controllability point (iv) by constructing a TDES T, as shown in **Figure 5.4**, and the modified components A_i^ω for TDES $\mathbf{G} \parallel \mathbf{S}$ where $\mathbf{G} = G_1 \parallel ..... \parallel G_n$, $\mathbf{S} = S_1 \parallel .... \parallel S_m$, and $\omega \notin \Sigma$.

Algorithm 14 SD controllability Point (iv) Algorithm

```
1: Create event ω ∉ Σ
2: for all A_i ∈ G ∥ S do
3:     Construct A_i^ω using Algorithm 15 and add it to A^ω
4: end for
5: Construct TDES T as shown in Figure 5.4 on page 90.
6: Use the Language Inclusion Check Algorithm to check if L(A^ω) ⊆ L(T)
7: if (L(A^ω) ⊆ L(T)) then
8:     return True
9: else
10:    return False
11: end if
```

For each component TDES that makes up **G** and **S**, the algorithm creates an A_i^ω using method described in **Section 5.3**. For each marked state, the algorithm creates an ω transition from the marked state to a dump state. It then creates the TDES A^ω which is the synchronous product of the above A_i^ω. It then uses the existing language inclusion checker to check if $L(A^\omega) \subseteq L(T)$. If the check passes, we can then conclude by **Theorem 5.4** that the system passes SD controllability

point (iv).

Algorithm 15 Construct $A^{\omega}(A)$

1: create a new TDES A^{ω} with the states, transitions and event set of TDES A.
2: Add new state y' to A^{ω}
3: **for all** states y in A **do**
4: **if** (y is a marked state) **then**
5: create a new transition (y, ω, y') and add it to the transitions of A^{ω}
6: **end if**
7: **end for**
8: add the event $\omega \notin \Sigma$ to the event set of A^{ω}
9: **return** A^{ω}

CHAPTER 7

SD Controlled FMS Example

For the application of our compositional verification algorithms, we consider the example of the SD controlled flexible manufacturing system (FMS) from [Wan09].

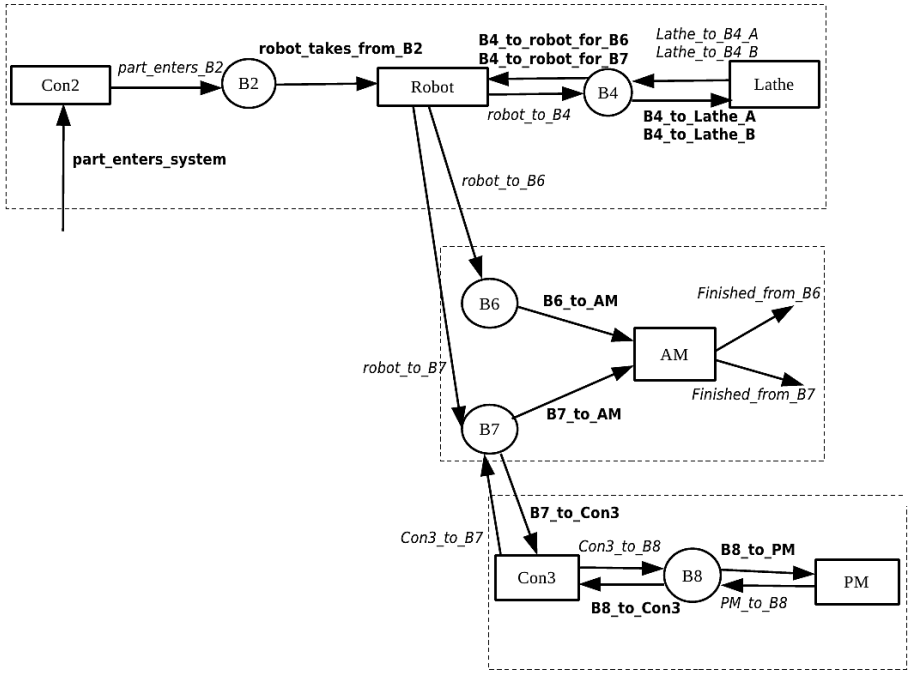


Figure 7.1: Flexible Manufacturing System Overview

The example is based on the untimed flexible manufacturing System from [Hil08]. A detailed description on how it is adapted to the SD controllable setting can be found in [Wan09]. The original FMS consists of six plant components

and five one slot buffers. The buffers are treated as specifications, with the requirement that they do not overflow or underflow. In [Hil08] and [Wan09] the event are labeled as mostly numbers, whereas we choose to label them with event names which represent their meanings, for convenience. Figure 7.1 shows the structure of the flexible manufacturing system. In the following sections we give a brief description of the plants and supervisors for the example. In a TDES, we use a small circle to represent a state. The initial state has an incoming arrow with no label and all marked states are indicated by a gray filled circle. In the individual TDES if an arrow is labelled by multiple events, then it represents a transition for each event. An uncontrollable event is represented by a **!** prefix attached to its label.

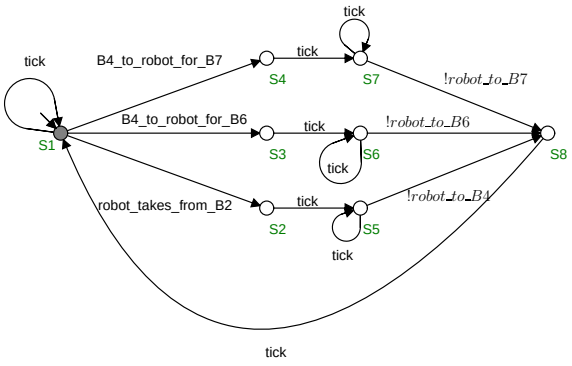


Figure 7.2: Robot

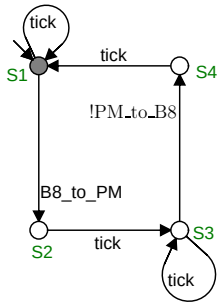


Figure 7.3: Painting Machine - PM

7.1 Plants

The SD Controlled flexible manufacturing system consists of the following plant components: two conveyors **Con2** and **Con3**, a **Robot**, a **Lathe** that can produce two different part types (A and B), a painting machine **PM**, a finishing machine **AM**, **SystDownNup** which handles the shutdown and restart of the

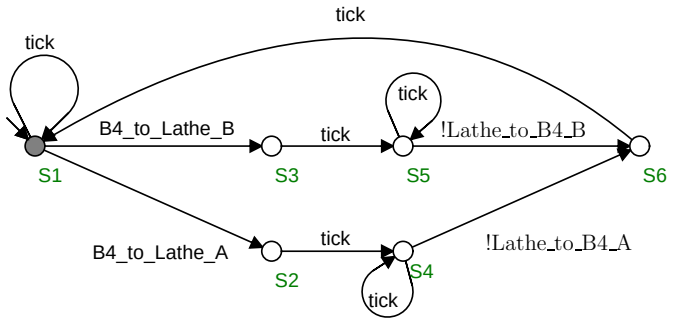


Figure 7.4: Lathe

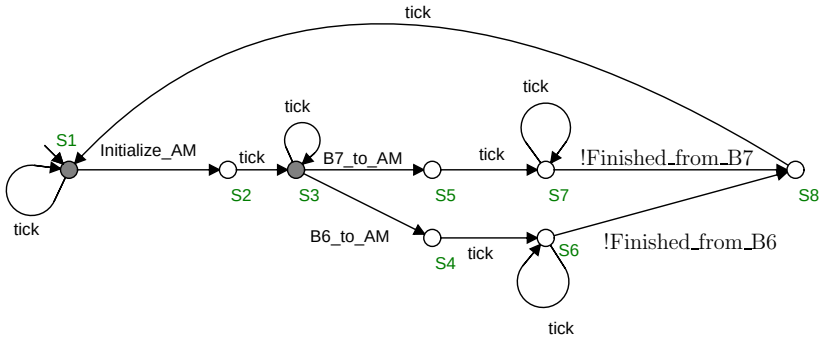


Figure 7.5: Finishing Machine - AM

system, and three other plants **AddNoPartEntersSystem**, **AddNoB6toAM** and **AddNoB7toAM**.

These last three plant components add several new expansion events to the system, namely *no_part_enters_system*, *noB6_to_AM_a*, *noB6_to_AM_b*, *noB7_to_AM_a*, and *noB7_to_AM_b*. Expansion events are events that are not part of the physical system, but were added as part of the supervisor implementation to aid in communication between supervisors. **Figures 7.2 - 7.11** show the TDES for the plant components.

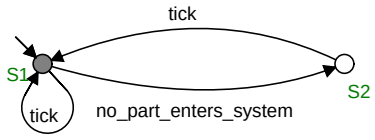


Figure 7.6: AddNoPartEntersSystem

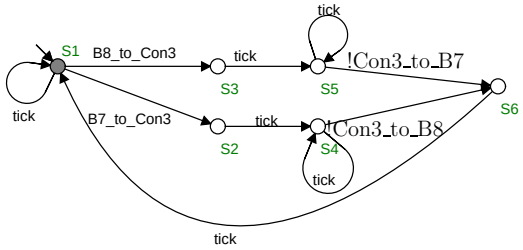


Figure 7.7: Conveyer- Con3

7.2 Supervisors

In this section we give a brief overview of each supervisor for the SD controlled flexible manufacturing system. Supervisors **B2**, **B4**, **B6**, **B7** and **B8** control the flow of parts through the corresponding buffers in order to prevent overflow and underflow of the buffers. The supervisors **TakeB2**, **SupB4**, **B4Path** and **LathePick** handle the flow of parts from buffer B4 to **Lathe** and back again, whereas moving parts from buffer B4 to buffer B6 and buffer B7 is handled by **TakeB4PutB6** and **TakeB4PutB7**. The supervisors **ForceB6toAM**, **ForceB7toAM**, **ForceInitializeAM** and **AMChooser** deal with the paths from buffers B6 and B7 leading through the **AM** and including when the parts exit the system. Each supervisor is briefly described in the following sections.

7.2.1 B2

In addition to preventing underflow and overflow of buffer B2, supervisor **B2** also decides when to force the event *part_enters_system*. It immediately forces

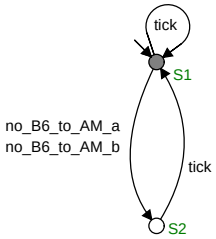


Figure 7.8: AddNoB6toAM

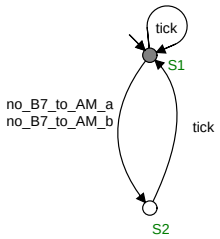


Figure 7.9: AddNoB7toAM

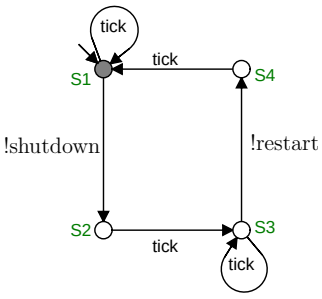


Figure 7.10: SystDownNup

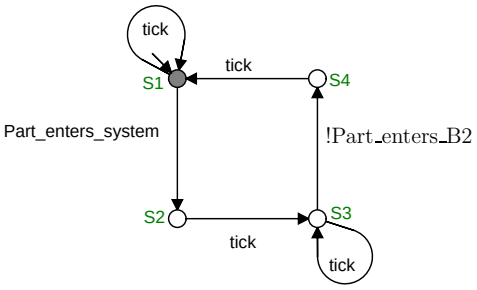


Figure 7.11: Conveyer - Con2

this event when the system starts so that **Con2** accepts a new piece into the system. Then it waits for the piece to enter buffer B2 before enabling the event $robot\_takes\_from\_B2$. It also makes sure that another part cannot enter the system until the event $robot\_takes\_from\_B2$ occurs, thus preventing overflow.

7.2.2 B6 and B7

The supervisors **B6** and **B7** control their own respective buffers, enabling and disabling events to prevent overflow or underflow. They do not force any events.

7.2.3 B8

The supervisor **B8** not only prevents overflow and underflow but also controls the movement of a piece between buffer B7 and machine **PM**. It observes the progress

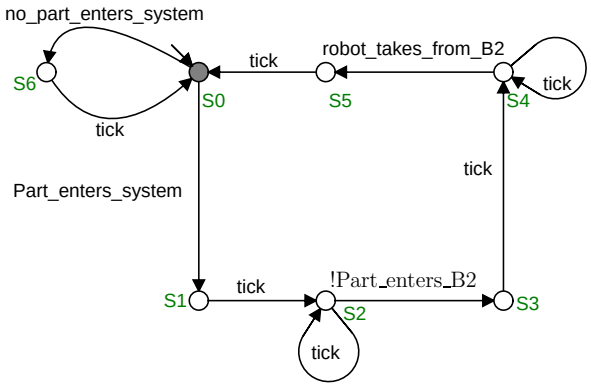


Figure 7.12: Supervisor B2

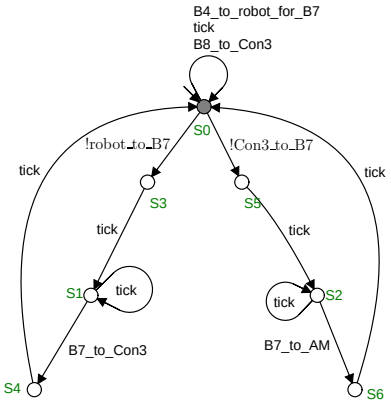


Figure 7.13: Supervsior B7

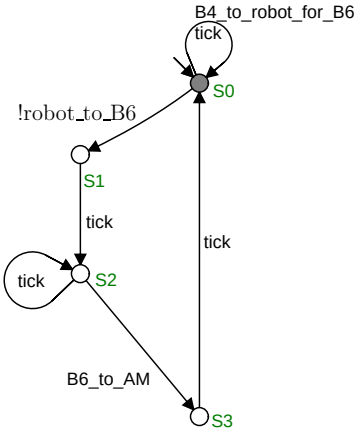


Figure 7.14: Supervisor B6

of the parts and forces the events $B7\_to\_Con3$, $B8\_to\_PM$ and $B8\_to\_Con3$ when required.

7.2.4 TakeB2

Supervisors **TakeB2** addresses two issues. First it decides which part the Robot should take when a part is waiting in both buffers B2 and B4. Supervisor **TakeB2** forces the Robot to first take the part from buffer B2, then from buffer B4 and then again from buffer B2 and so on. It waits for a piece to enter buffer B2 (event $part\_enters\_B2$) and immediately enables and forces the event

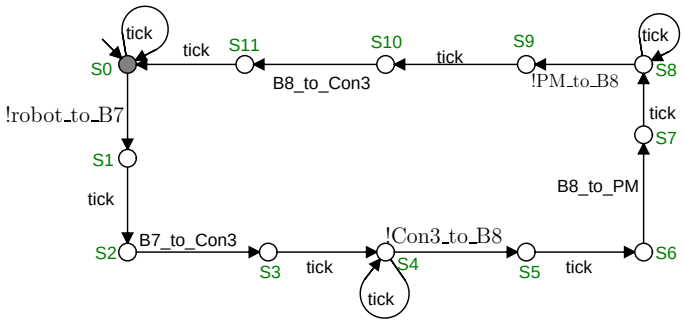


Figure 7.15: Supervisor B8

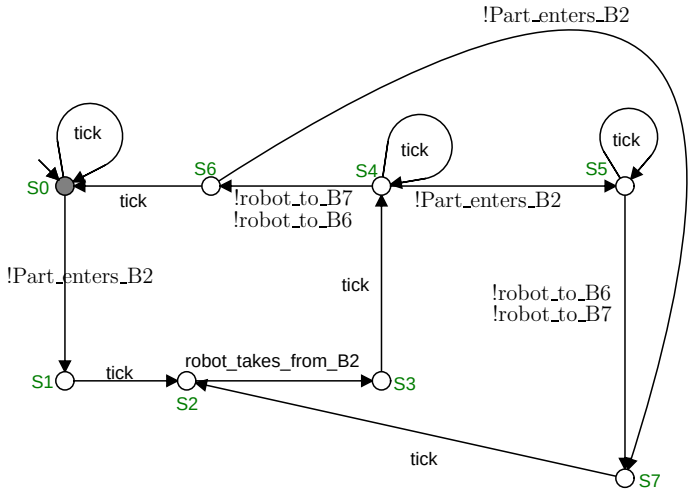


Figure 7.16: Supervsior TakeB2

robot_takes_from_B2 to move the piece to buffer B4. Then it waits for the piece to go to Lathe for processing, then be returned to buffer B4. Then it takes the piece from buffer B4 and moves the piece to buffer B6 or B7 (depending upon the type of the piece), before it allows the Robot to take a piece from buffer B4 again.

Second **TakeB2** deals with a blocking issue related to buffer B4. The buffer B4 accepts a piece from the Robot and gives it to the Lathe for processing. While this piece is being processed, it is possible that buffer B4 might accept another piece from the Robot, thus leaving no place for the Lathe to return the part

that it was processing. This would result in making the system block. **TakeB2** resolves this issue by disabling event $robot\_takes\_from\_B2$ until the part being processed by Lathe is returned to buffer B4 and then moved to buffer B6 or buffer B7.

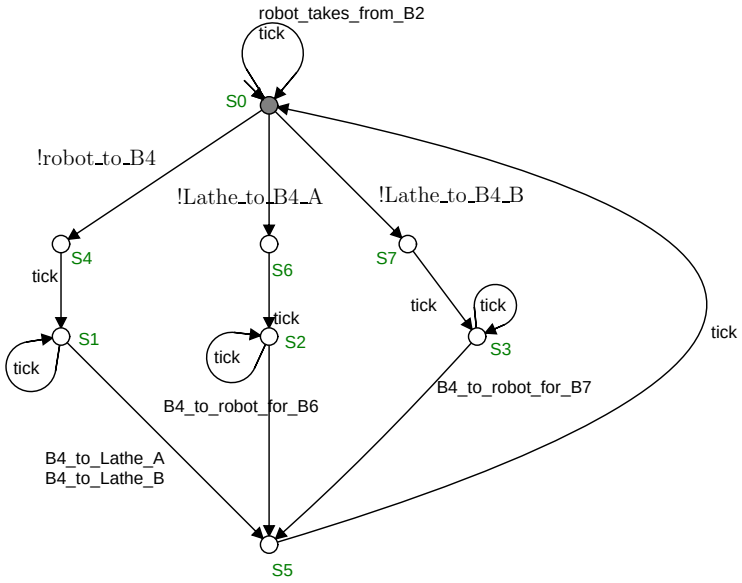


Figure 7.17: Supervsior B4

7.2.5 B4

In addition to ensuring that buffer B4 does not overflow or underflow, supervisor **B4** also makes sure that once there is a piece in buffer B4, appropriate action is taken when the piece is taken out. When the Robot puts a piece in buffer B4 by means of event $robot\_to\_B4$, supervisor **B4** makes sure that only event $B4\_to\_Lathe\_A$ and $B4\_to\_Lathe\_B$ can occur to move the piece out of buffer B4 to Lathe. The piece can then be processed by the Lathe depending on its type. The piece is then returned by the Lathe to buffer B4 using events $Lathe\_to\_B4\_A$ or $Lathe\_to\_B4\_B$. The type **A** piece then proceeds to buffer B6 by the event $B4\_to\_robot\_for\_B6$, the type **B** piece proceeds by $B4\_to\_robot\_for\_B7$.

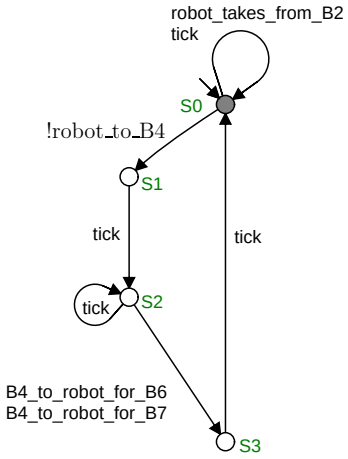


Figure 7.18: Supervisor B4Path

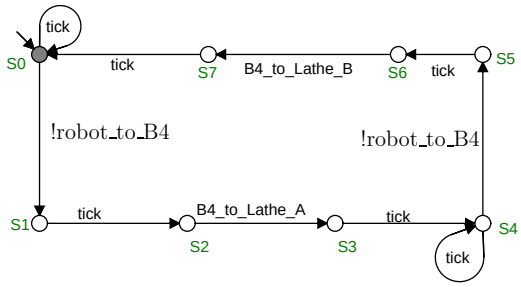


Figure 7.19: Supervsior LathePick

7.2.6 B4Path

The supervisor **B4Path** ensures correct flow of material through the B4 to Lathe path. It disables the event $robot\_takes\_from\_B2$ once a part is taken to buffer B4 from buffer B2, and disables $B4\_to\_robot\_for\_B6$ and $B4\_to\_robot\_for\_B7$ until the Robot puts a part in buffer B4.

7.2.7 LathePick

This supervisor helps in controlling the flow of material through the B4 to Lathe path as well. Supervisor **LathePick** enforces the decision that the Lathe first process a part of type **A**, and then a part of type **B**, and so on.

7.2.8 TakeB4PutB6

The event $B4\_to\_robot\_for\_B6$ is used for moving a part from buffer B4 to buffer B6. Supervisor **TakeB4PutB6** decides when to enable and force this event. After waiting for event $Lathe\_to\_B4\_A$ to occur, it forces event $B4\_to\_robot\_for\_B6$. It then waits for the event $B6\_to\_AM$ to occur which indicates that buffer B6 is

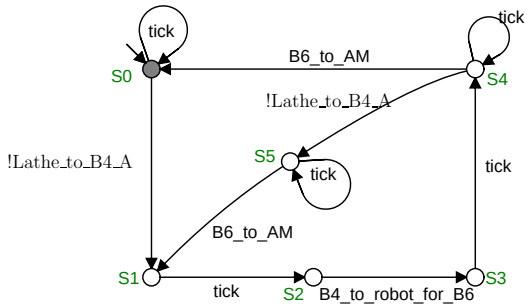


Figure 7.20: Supervisor TakeB4PutB6

empty and ready to take a new part.

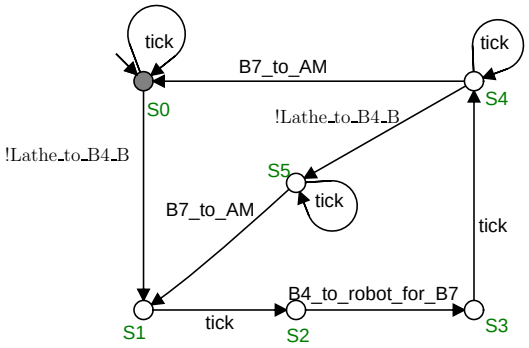


Figure 7.21: Supervisor TakeB4PutB7

7.2.9 TakeB4PutB7

This supervisor takes care of two issues. First, it decides when to enable and force the event $B4\_to\_robot\_for\_B7$. Second, we have a nonblocking issue. When a part put in buffer B7 goes to PM for processing, a situation could arise where the Robot puts another part in buffer B7 such that there is no place for the first part being processed to return to buffer B7. Supervisor **TakeB4PutB7** observes and waits until buffer B4 has a part of type **B** ready to be transferred to buffer B7 (event $Lathe\_to\_B4\_B$). Then it enables and forces the event $B4\_to\_robot\_for\_B7$ to transfer the part. Next it waits for the part to leave from buffer B7 (event

$B7\_to\_AM$) before it allows Robot to take another part from buffer B4 (event $B4\_to\_robot\_for\_B7$).

7.2.10 ForceB6toAM

Supervisor **ForceB6toAM** watches for the event $robot\_to\_B6$ which means there is a part in buffer B6 ready to go to AM. At this point the supervisor forces the event $B6\_to\_AM$. However, this event could currently be disabled by other supervisors or not possible in the plant. In this case, it forces event $noB6\_to\_AM\_a$ or $noB6\_to\_AM\_b$ and tries again to force the event $B6\_to\_AM$ after the tick.

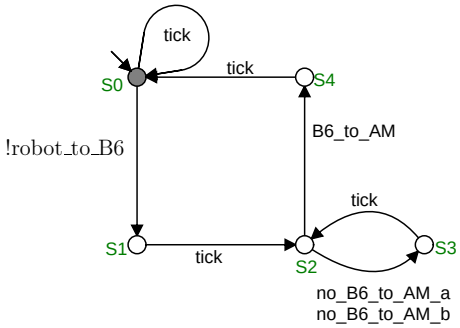


Figure 7.22: Supervisor ForceB6toAM

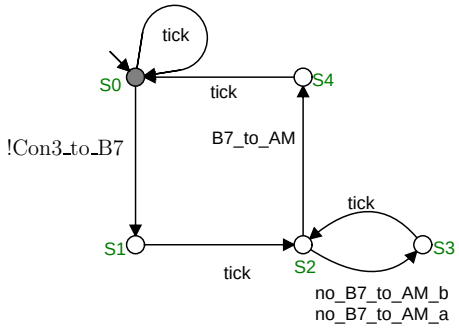


Figure 7.23: Supervsior ForceB7toAM

7.2.11 ForceB7toAM

This supervisor represents exactly the same behaviour for event $B7\_to\_AM$ as supervisor **ForceB6toAM** does for event $B6\_to\_AM$. See Figure 7.23 for details.

7.2.12 ForceInitializeAM

The supervisor **ForceInitializeAM** primarily decides when to force the event $Initialize\_AM$. The supervisor forces this event straightaway and then waits

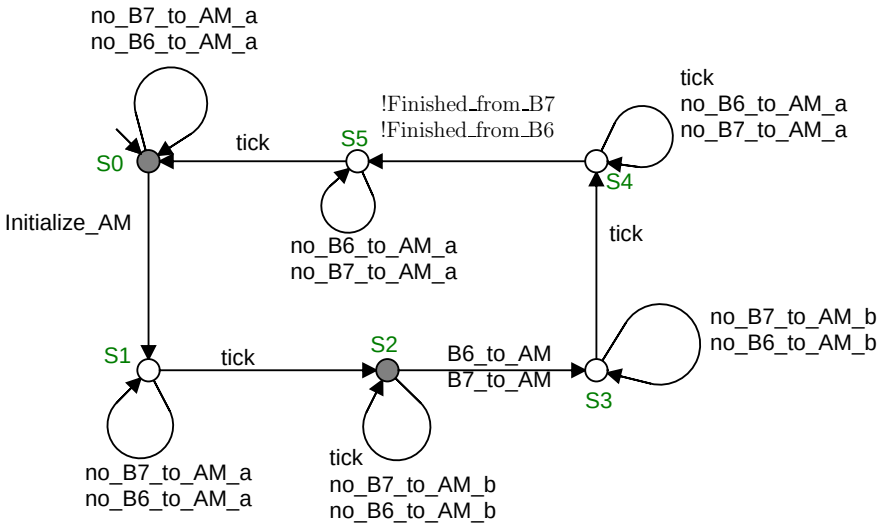


Figure 7.24: Supervisor ForceInitializeAM

for the events $Finished\_from\_B6$ or $Finished\_from\_B7$ to occur before forcing the event $Initialize\_AM$ again. The other task of this supervisor is to enable the events $noB6\_to\_AM\_a$ and $noB7\_to\_AM\_a$ when the respective events $B6\_to\_AM$ and $B7\_to\_AM$ are not eligible to occur in the plant component AM. When these events are eligible in the plant, the supervisor enables $noB6\_to\_AM\_b$ and $noB7\_to\_AM\_b$ instead. This ensures that when events $B6\_to\_AM$ and $B7\_to\_AM$ are not possible in the plant, events $noB6\_to\_AM\_a$ and $noB7\_to\_AM\_a$ will always be possible after a tick. It also ensures that the 'a' and 'b' events never become eligible at the same time.

We note that the two supervisors **ForceInitializeAM** and **AMChooser** work in harmony with each other with respect to the 'a' and 'b' events. Supervisor **AMChooser** ignores the 'a' events, therefore never disabling them, and Supervisor **ForceInitializeAM** never disables the 'b' events when events $B6\_to\_AM$ and $B7\_to\_AM$ are eligible in the plant.

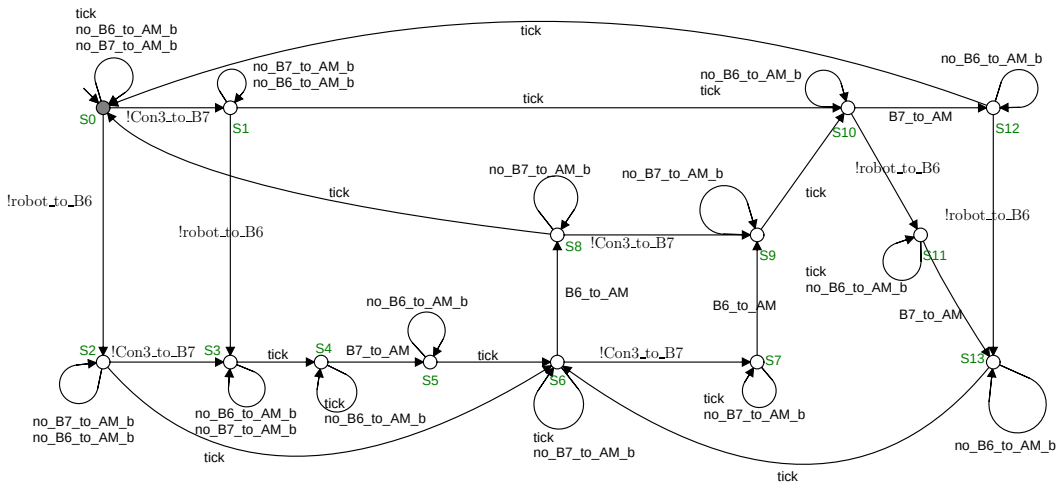


Figure 7.25: Supervisor AMChooser

7.2.13 AMChooser

This supervisor has to choose which buffer to service when both buffers B6 and B7 have a part waiting to be processed by AM. If both the buffers receive a part in the same sampling period, then the part in buffer B7 is chosen to be processed first. Then the part in buffer B6 is processed and the parts are kept on being chosen in alternating manner. If only one buffer has a part in a given sampling period, then that part is processed. In order to implement this ordering, the supervisor **AMChooser** sometimes has to disable events $B6\_to\_AM$ or $B7\_to\_AM$ and then to compensate it enable events $noB6\_to\_AM\_b$ or $noB7\_to\_AM\_b$, respectively.

7.2.14 handleSystDown

The supervisor **handleSystDown** enforces the shutdown/restart mechanism. Initially it enables the event $part\_enters\_system$ and disables the event $no\_part\_enters\_system$. Once the event shutdown has occured, it disables $part\_enters\_system$ and enables $no\_part\_enters\_system$. When the event restart occurs, it does the reverse. It also ensures the events $part\_enters\_system$ and

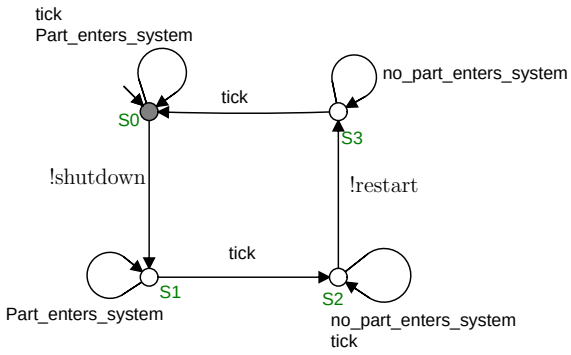


Figure 7.26: Supervisor handleSystDown

no_part_enters_system are never enabled at the same time and one of them is always possible and enabled immediately after a tick. This supervisor works in conjunction with supervisor **B2** to stop new parts from entering the system to allow the system to empty out and shut down.

7.3 Results

	Monolithic			Modular/Compositional				
	Runtime (sec)	Total States	Peak States	Runtime (sec)	Total States	Peak States	Faster runtime	Peak state difference
Plant Completeness	0.271	82,608	82,608	0.088	1,344	1,344	3 times	-81,264
Activity-loop Free	0.122	19,532	19,532	0.004	46	8	30 times	-19,524
Proper Time Behaviour	7.236	2,359,296	2,359,296	0.0642	124	16	112 times	-2,359,280
S-singular Prohibitable Behaviour	5.746	2,726,064	82,608	0.111	490	62	51.7 times	-82,546
SD Controllability i	0.637	82,608	82,608	0.407	16,212	16,212	1.5 times	-66,396
SD controllability ii.a	0.265	105,328	105,328	0.174	164,497	164,497	1.5 times	59,169
SD Controllability ii.b	0.147	165,216	165,216	7.014	193,332	41,714	-48 times	-123,502
SD Controllability iii.1	12.38	5,089,080	164,576	0.221	23,277	3,709	56 times	-160,867
SD Controllability iii.2	0.128	82,608	82,608	0.010	171	171	12.8 times	-82,437
SD Controllability iv	0.253	82,608	82,608	0.165	148,068	148,068	1.5 times	65,460

Table 7.1: Statistics from FMS Example

To test our algorithms, we applied our compositional verification algorithms on the example of SD controlled Flexible Manufacturing System. **Table 7.1** shows the results of the verification using our algorithms. For comparision purposes we have included the results of the monolithic algorithms as well. This is given to provide insight to how well the compositional approach works as com-

pared to the monolithic approach.

Table 7.1 shows the total number of states, peak number of states and average run time (in seconds) for each algorithm. Average runtimes are calculated from several runs of the algorithms. The total number of states represents the total states constructed for the synchronous product, whereas the peak number of states identify the size of the largest automaton constructed by the algorithm. The table also shows the increase factor for the runtime and the difference in peak states for each algorithm for comparison purposes. The Runtime factor shows how many times faster the compositional algorithm is as compared to the monolithic one. A negative sign shows how much slower the compositional algorithm is than the monolithic algorithms. The last column shows the difference in peak states for both the algorithms. A negative sign shows that the monolithic algorithm constructs that many fewer states than the compositional algorithm.

The algorithms for Proper Time Behaviour, S-singular prohibitable behaviour, SD Controllability point (ii) and SD controllability point (iii.1) construct and add an additional TDES to the system therefore increasing the total number of states constructed by the algorithms. The algorithms for S-singular prohibitable behaviour and SD controllability point (iii.1) invoke the checker again and again for each prohibitable event in the system, resulting in even more number of states. The peak number of states are different than total number of states for the monolithic versions of these algorithms as it represent the size of the syncronous product constructed by the algorithm for a single run of the checker. For SD controllability point (iii.1), we have merged the results for the **Algorithm 6** (SD controllability point (iii.1a)) and **Algorithm 7** (SD controllability point (iii.1b)). Therefore the peak states entry for this property represents the total number of states for one of the algorithms.

The tests were run on a 2.2GHz computer with 4GB of memory, running

under Fedora Linux 15 using Supremica version 201202221527. The verification using monolithic algorithms took about **27.18** seconds with the total number of states of the synchronous product calculated as 82,608. The verification of compositional algorithms took a total of **8.258** seconds. This represents a speedup by a factor of three. We also not that if we replaced the modular check of SD controllability (ii.b) with a monolithic check, we would have a total run time of 1.39 seconds which would represent a 14 times speedup.

As shown in Table 7.1, all the algorithms except for SD Controllability (ii.b) show improvement in terms of runtime. In Chapter 5, Section 5.1.2, we redefine SD controllability (ii.b) as a nonblocking property. The check involves the TDES T_{Υ}, which considers the entire event set Σ and it is likely that all TDES need to be taken into account to verify that the property is satisfied. In that case, the incremental verifiers cannot be of much help and thus it takes more time to perform the compositional constructions as well as verifying the property. Similar results can be seen for the properties SD Controllability (ii.a) and SD controllability (iv) for the the peak states. Since both of these properties also construct and use automata that consider the entire event set Σ, the algorithms end up using more peak states than the monolithic algorithms as they perform more constructions and verification for the compositional methods. For an example that fails the condition, it is quite likely that the modular algorithm will determine this faster than the monolithic algorithm. For most of the algorithms the results show that the compositional approach gives us better results in terms of time for this example.

CHAPTER 8

Conclusions

8.1 Conclusions

In this thesis we developed a compositional verification method for the sampled-data supervisory control approach that deals with timed DES systems to implement the supervisors as sampled-data (SD) controllers.

The sampled-data supervisory control approach identifies a set of conditions a system should satisfy when its supervisor, S, is implemented as an SD controller. These conditions are plant completeness, activity-loop free, S-singular prohibitable behaviour, proper time behaviour and SD controllability. To compositionally verify these SD properties, we redefined them in terms of other properties such as language inclusion, nonblocking and controllability. Redefining them in terms of these other properties makes it easy to apply their exisiting compositional verification algorithms to verify the SD properties.

We then presented the algorithms for the compositional verification of these properties. We implemented the algorithms in Java as a part of the project **Supremica**. Finally, we considered a flexible manufacturing system example from the literature, and applied our software to it. We demonstrated that we were able to handle this example, with better results than a monolithic approach.

8.2 Future Work

Since the sampled-data supervisory control framework is still relatively new, and there are not many timed DES examples available, we could not analyse the true potential of our algorithms and code. Further work towards verifying larger and more complex models would give a better idea of the potential of the algorithms. Also, as we ran out of time, heuristics for the the algorithm for verifying SD controllability point (iii.2) could not be explored and developed; therefore different heuristics for automata selection could be investigated.

Bibliography

[ÅFF02] Knut Åkesson, Hugo Flordal, and Martin Fabian. Exploiting modularity for synthesis and verification of supervisors. *Proceedings of the IFAC World Congress on Automatic Control, Barcelona, Spain*, 2002.

[Bal94] Silvano Balemi. Input/output discrete event processes and communication delays. *Discrete Event Dynamic Systems*, 4:41–85, 1994. 10.1007/BF01516010.

[BHG+93] S. Balemi, G.J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G.F. Franklin. Supervisory control of a rapid thermal multiprocessor. *Automatic Control, IEEE Transactions on*, 38(7):1040 –1059, Jul 1993.

[BMM04] B.A. Brandin, R. Malik, and P. Malik. Incremental verification and synthesis of discrete-event systems guided by counter examples. *Control Systems Technology, IEEE Transactions on*, 12(3):387 – 401, May 2004.

[Bra92] B. A. Brandin. *Real-Time supervisory control of automated manufacturing system*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, ON, Canada, 1992.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24:293–318, September 1992.

[BW94] B.A. Brandin and W.M. Wonham. Supervisory control of timed

discrete-event systems. *Automatic Control, IEEE Transactions on*, 39(2):329 –342, Feb 1994.

[CCMM95] S. Campos, E. Clarke, W. Marrero, and M. Minea. Verifying the performance of the PCI local bus using symbolic techniques. In *Computer Design: VLSI in Computers and Processors, 1995. ICCD '95. Proceedings., 1995 IEEE International Conference on*, pages 72 –78, Oct 1995.

[CL10] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems, 2nd ed.* Oct 2010.

[CLM89] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *Logic in Computer Science, 1989. LICS '89, Proceedings., Fourth Annual Symposium on*, pages 353 –362, June 1989.

[FM09] Hugo Flordal and Robi Malik. Compositional verification in supervisory control. *SIAM J. Control Optim.*, 48:1914–1938, June 2009.

[Fra11] Rachel Francis. An implementation of a compositional approach for verifying generalised nonblocking. Technical report, Department of Computer Science, The University of Waikato, Hamilton, New Zealand, 2011.

[GL94] Orna Grumberg and David E. Long. Model checking and modular verification. *ACM Trans. Program. Lang. Syst.*, 16:843–871, May 1994.

[GR87] C. H. Golaszewski and P. J. Ramadge. Control of discrete event processes with forced events. In *Decision and Control, 1987. 26th IEEE Conference on*, volume 26, pages 247 –251, Dec. 1987.

[HGCC96] V. Hartonas-Garmhausen, E.M. Clarke, and S. Campos. Deadlock prevention in flexible manufacturing systems using symbolic model checking. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 1, pages 527 –532 vol.1, Apr 1996.

[HGKCL95] V. Hartonas-Garmhausen, T. Kurfess, E.M. Clarke, and D. Long. Automatic verification of industrial designs. In *Industrial-Strength Formal Specification Techniques, 1995. Proceedings., Workshop on*, pages 88 –96, Apr 1995.

[Hil08] R. C. Hill. *Modular Verification and Supervisory Controller Design for Discrete Event Systems Using Abstractions and Incremental Construction.* PhD thesis, Department of Mechanical Engineering, University of Michigan, 2008.

[KM00] K.L. and McMillan. A methodology for hardware verification using compositional model checking. *Science of Computer Programming*, 37(1-3):279 – 309, 2000.

[KS94] R. Kumar and M.A. Shayman. Non-blocking supervisory control of nondeterministic discrete event systems. In *American Control Conference, 1994*, volume 1, pages 1089 – 1093 vol.1, June 1994.

[LBLW05] R.J. Leduc, B.A. Brandin, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control-part I: serial case. *Automatic Control, IEEE Transactions on*, 50(9):1322 – 1335, Sept. 2005.

[LDS09] R.J. Leduc, Pengcheng Dai, and Raoguang Song. Synthesis method

for hierarchical interface-based supervisory control. *Automatic Control, IEEE Transactions on*, 54(7):1548 –1560, July 2009.

[Led02] Ryan James Leduc. *Hierarchical Interface-Based Supervisory Control*. PhD thesis, Department of Electrical Engineering, University of Toronto, ON, Canada, 2002.

[Led09] Ryan J. Leduc. Hierarchical interface-based supervisory control with data events. *International Journal of Control*, 82(5):783–800, 2009.

[LLD06] R.J. Leduc, M. Lawford, and Pengcheng Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. *Control Systems Technology, IEEE Transactions on*, 14(4):654 – 668, July 2006.

[LLW05] R.J. Leduc, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control-part II: parallel case. *Automatic Control, IEEE Transactions on*, 50(9):1336 – 1348, Sept. 2005.

[LM10] Ryan Leduc and Robi Malik. A compositional approach for verifying hierarchical interface-based supervisory control. In *Proc. 10th International Workshop on Discrete Event Systems (WODES'10)*, pages 114–120, Berlin, Germany, Aug 2010.

[LW10] Ryan J. Leduc and Yu Wang. Sampled-data supervisory control. In *Proc. 10th International Workshop on Discrete Event Systems (WODES'10)*, pages 353–359, Berlin, Germany, Aug 2010.

[Mal03] P. Malik. *From supervisory control to nonblocking controllers for discrete event systems*. PhD thesis, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, 2003.

[McM93] K. L. McMillan. Symbolic model checking. Kluwer, 1993.

[ML08] R. Malik and R. Leduc. Generalised nonblocking. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 340 –345, May 2008.

[ML09] R. Malik and R. Leduc. A compositional approach for verifying generalised nonblocking. In *Control and Automation, 2009. ICCA 2009. IEEE International Conference on*, pages 448 –453, Dec. 2009.

[MM06] P. Malik and R. Malik. Modular control-loop detection. In *Discrete Event Systems, 2006 8th International Workshop on*, pages 119 – 124, July 2006.

[Ost89] J.S. Ostroff. *Temporal logic for real-time systems*. Advanced software development series. Research Studies Press, 1989.

[Ost90] J.S. Ostroff. Deciding properties of timed transition models. *Parallel and Distributed Systems, IEEE Transactions on*, 1(2):170 –183, Apr 1990.

[OW90] J.S. Ostroff and W.M. Wonham. A framework for real-time discrete event control. *Automatic Control, IEEE Transactions on*, 35(4):386 –397, Apr 1990.

[PCL06] P.N. Pena, J.E.R. Cury, and S. Lafortune. New results on testing modularity of local supervisors using abstractions. In *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*, pages 950 –956, Sept. 2006.

[RMR06] D. Streader R. Malik and S. Reeves. Conflicts and fair testing. *Int. J. Found. Comput. Sci.*, 17(4):797–813, 2006.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81 –98, Jan 1989.

[Sup] Supremica. [Online], Available: http://www.supremica.org.

[Wan09] Yu Wang. Sampled-data supervisory control. Master's thesis, Department of Computing and Software, McMaster University, Hamilton, ON, Canada, 2009.

[WM08] S. Ware and R. Malik. The use of language projection for compositional verification of discrete event systems. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 322 –327, May 2008.

[Won11] W.M. Wonham. Supervisory control of discrete event systems. Technical report, Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada, 2011.

[WR87] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. In *SIAM J. Control Optim.*, volume 25, pages 637 –659, 1987.

[WW96] K. C. Wong and W. M. Wonham. Hierarchical control of timed discrete-event systems. *Discrete Event Dynamic Systems*, 6:275–306, 1996. 10.1007/BF01797155.