Weak Invariant Simulation and Analysis of Parameterized Networks

by

Mohammad Hadi Zibaeenejad

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer engineering

Waterloo, Ontario, Canada, 2014

Abstract

Multi-process networks figure in many engineering applications such as communication networks, transportation networks, manufacturing and logistic systems, and computer hardware and software. Parameterized discrete event systems provide a convenient means of modeling such networks when the number of subprocesses is arbitrary, unknown or time-varying. Unfortunately, some key properties of these networks, such as nonblocking and deadlock-freedom, are undecidable. Moreover, mathematical tools supporting analysis of these networks are limited. This thesis introduces a novel mathematical notion, *weak invariant simulation* and proposes an efficient method to check whether a finite-state generator weakly invariantly simulates another finite-state generator with respect to a specific subalphabet. This new simulation relation is first used to define a tractable subclass of parameterized ring networks of isomorphic subprocesses in which deadlock-freedom is decidable. Within this framework, a procedure is given to determine the reachable deadlocked states of the network. The effectiveness of the procedure is demonstrated by the deadlock analysis of a version of the dining philosophers problem.

To generalize the results on ring networks, we consider a network consisting of several linear parameterized sections but exhibiting a branching topology. To model these networks we introduce Generalized Parameterized Discrete Event Systems (GPDES). The difficulty in analysis of a GPDES is the fact that some of the subprocesses interact with several parameterized sections of the network. Hence the analysis proposed in this thesis involves careful study of interaction among different branches of the network. Here again, we use 'weak invariant simulation' to limit the behavior of subprocesses of the network. Then we investigate interactions among different components of the network, using a *dependency graph.* The dependency graph is a directed graph developed to characterize reachable partial deadlocks caused by generalized circular waits in the proposed GPDES. Our results implicitly characterize reachable generalized circular waits as a language accepted by a finite automaton. Our framework allows for modeling and analysis of new parameterized problems. We investigated deadlock in a large-scale factory as an illustrative example.

Acknowledgements

First and foremost, I would like to thank Professor John G. Thistle for his fantastic supervision during the course of my PhD program. This thesis is the outcome of hours of fruitful discussion with him. He taught me how to deal with new and complex problems in a rigorous and formal manner. Prof. Thistle's immense knowledge and integrity were always inspiring to me.

Second, I am grateful to Professor R. S. Sreenivas of University of Illinois for insightful evaluation my dissertation as an external examiner. I also appreciate his presence in the oral defense session despite his busy schedule and traveling difficulties. I would like to express my gratitude to my advisory committee members Professors Nancy Day, Shreyas Sundaram, and Patrick Lam for their technical comments and suggestions.

Third, I send my special thanks to my friend Amirhossein Vakili for useful discussions and to my brother Ali Zibaeenejad for his help and guidance. The most sincere appreciation goes to my parents, Professor M. Javad Zibaeenejad and Khadijeh Mohammadi for their unconditional love and support in my entire life specially during my difficult life as an international student in Canada.

Last but not least, I would like to thank my dearest wife, Ensieh Mollazadeh, for her selfless support and wonderful companionship.

Dedication

This work is dedicated to the Lord.
I praise him for his guidance and blessings.

Table of Contents

List of Figures

Chapter 1

Introduction and Related Work

The evolution of digital technology has enabled the engineering of complex dynamic systems such as computer networks [27] and automated manufacturing systems [36] and transportation networks [46]. Such systems require the design of many layers of control: at low levels, traditional control theory, based on detailed differential-equation models, supports the design of relatively small-scale control loops; but higher-level problems of the coordination of multiple interacting subsystems are often better formulated in terms of more abstract discrete-event models, characterized by abrupt, instantaneous, asynchronous events.

Control theorists have worked over the last few decades toward the development of a comprehensive theory of the control of Discrete-Event Systems (DES) [35,37,51–54]. Their efforts are exemplified by the *supervisory control theory* of Ramadge and Wonham [34].

Supervisory control bears a substantial relationship to formal verification of computer-based systems, with the significant exception that it is aimed primarily at formal synthesis. It has employed similar models, tools and techniques as formal verification. These allow for a range of different representations of discrete-event systems, but for computational purposes, finite automata have furnished a convenient prototype.

Supervisory control proposes modular synthesis techniques and abstraction as means of addressing the central challenge of computational complexity [22, 31, 39]. A key stumbling block in the engineering of discrete event systems is the combinatorial explosion that occurs when models of N subprocesses are composed, yielding an overall process model of size exponential in N. This thesis identifies a class of systems which admit analysis procedures that are independent of any specific value of N. A simple example is a buffer in a manufacturing system, which can be modeled as a family of isomorphic interacting buffer cells. Control policy for avoidance of buffer overflow and underflow is essentially independent of the specific buffer capacity.

This thesis considers families of finite-state systems, in which each member corresponds to a different value of some system parameter (such as a buffer capacity). Such models are called parameterized systems. More specifically, we shall study networks consisting of a finite but arbitrarily large number of similar finite-state subprocesses. We shall suppose that subprocesses are isomorphic – each being obtained through suitable relabellings of a 'template' process – and consider the infinite family of finite state systems obtained by letting the number of subprocesses range over the natural numbers. In these networks, subprocesses interact via execution of shared events (a shared event between subprocesses can occur only if all the subprocesses are in a state that can perform the shared event). Such 'parameterized networks' furnish useful models of communication and transportation networks, manufacturing systems, etc. More generally a *parameterized network* is composed of arbitrary numbers of isomorphic subprocesses. Formally, such systems can be modeled as infinite families of finite-state systems. Practical examples of parameterized networks include wireless sensor networks, transportation networks, manufacturing systems and subprocesses in operating systems. Parameterized models are particularly useful when the number of subprocesses is unknown, time-varying, or very large.

Specific motivation for using this network model arose from prior studies of the development of call-processing services in telecommunications networks within the formal framework of discrete-event control [48]. In these networks, development of telecommunications services is considered as the design of decentralized supervisors. Using parametrized modeling is particularly useful since the number of users of such networks vary over time.

Parameterized systems have received considerable attention in the model-checking literature [2, 12, 40]. Indeed, if the underlying logic of a hardware or software system is in essence independent of specific parameter values, one might expect to be able to establish correctness without focusing on the individual finite-state models that arise when parameters are instantiated. Similarly, in control, it is to be expected that many instances of control problems give rise to control logic that is in essence independent of the values of specific parameters. Ideally, one might be able to adapt prototypical synthesis methods for finite-automaton models in such a way as to extract from the controller design the essential parameter-independent underlying logic.

It is natural to ask how much analysis can be done independently of any specific parameter value. Unfortunately, key problems such as that of checking the nonblocking property for parameterized networks are undecidable [32]. That is to say there is no algorithm that will determine whether there exists a parameter value for which the network blocks.

Behrer *et al.* have proposed a control synthesis procedure for parameterized networks [5]; but their synthesis procedure does not address blocking issues.

Our long term goal is to develop effective nonblocking supervisor synthesis methods for parameterized systems. Given the undecidability of checking nonblocking property in a parameterized system, we focus on the related problem of locating reachable deadlocks in a parameterized systems. Detecting deadlocks in distributed networks is a classic problem which is extensively studied [11]. A deadlocked state is a state from which no more transitions are possible. As will be shown below computing reachable deadlocks in a parameterized network is undecidable. Therefore, in this thesis our main aim would be investigating the existence of nontrivial classes of parameterized systems for which efficient deadlock analysis is possible.

A key feature of nontrivial deadlocks in distributed systems is the presence of a 'circular wait' or 'deadly embrace' [17]. As stated in [47], in a circular wait, the only available action by each subprocess requires a resource that is being held by another subprocess. In order to focus on this central feature, in this thesis we consider networks with the topology of a ring. In the sense that it allows us to address the central issue of circular waits, the restriction to ring topologies does not represent a substantial loss of generality. Indeed in an arbitrary topology, nontrivial deadlocks take place within subnetworks that form closed walks, and contain circular waits.

Therefore we first focus on parameterized ring networks. A ring network is a structure in which each component can only interact with its immediate neighbors. The deadlock analysis is still generally undecidable in these networks [41]. In our setting, subsystem interactions with immediate neighbors are modeled by shared events. A shared event between two neighbors can occur if both neighbors are in a state that can perform the shared event. As a part of our research we introduce the new mathematical notion of *weak invariant simulation*. Using this notion, we propose a ring network model such that each subsystem interacts with both immediate neighbors, but can only be permanently blocked by shared events with its 'counter-clockwise' neighbor. In this sense, our model generalizes that of [15]. Furthermore, our proposed model is general enough to admit realistic examples such as a version of the Dining Philosophers Problem which occurs in some concurrent systems.

After analysis of parameterized ring network models, we consider networks consisting of multiple parameterized sections, with branching topologies. To model these networks, we introduce Generalized Parameterized Discrete Event Systems (GPDES). A GPDES consist of fixed number of 'linear 'parameterized sections as well as fixed number of 'distinguished' subprocesses. Distinguished subprocesses may have an arbitrary structure, which is dissimilar to that of any other subprocess. Each linear parameterized section contains several isomorphic subprocesses: each subprocess in a linear parameterized section of the network is obtained by relabeling of a template. The number of subprocesses in each parameterized section is arbitrary.

For both ring networks and the more general branching topologies modeled by a GPDES, we show that the existence of reachable (partial) deadlocks is decidable. Our results can be interpreted in automata- or language-theoretic terms. In Chapter 3, we characterize the set of deadlock states of a parameterized ring network as a regular language over the finite alphabet of state symbols of subprocesses. This constitutes the language accepted by a finite automaton. In network topologies that incorporate branching, our results implicitly characterize deadlock states as the set of finite trees accepted by a particular finite automaton. The results are consequently related to regular model checking, in which states are represented as regular languages [6].

This thesis is organized as follows: in the following section, we will survey some of the related work reported in the literature. In Section 1.2, preliminary notions and notations from discrete events systems and graph theory are introduced.

Chapter 2 introduces the process algebraic notion of *weak invariant simulation*, compares weak invariant simulation to other simulation relations in the literature, and investigates properties of weak invariant simulation. The content of this chapter has appeared in [59, 60]: weak invariant simulation was introduced in its initial form in [59], and later in more general form (applicable to nondeterministic DES) in [60]. Properties of weak invariant simulation are also covered in [60].

Using weak invariant simulation, Chapter 3 defines a tractable subclass of parameterized ring networks of isomorphic subprocesses in which deadlock-freedom is decidable. Within this framework, this chapter gives a procedure to determine the reachable deadlocked states of the network. The content of this chapter appeared in [58]. Chapter 4 considers a network consisting of several linear parameterized sections but exhibiting a branching topology and introduces Generalized Parameterized Discrete Event Systems (GPDES) to model these networks. This chapter proposes a deadlock analysis procedure for these networks using a *dependency graph*. The dependency graph is a directed graph we developed to characterize reachable partial deadlocks caused by generalized circular waits in the proposed GPDES. An early result related to this chapter appeared in [57]. The rest of the material of this chapter is submitted for publication [55, 56].

Finally, chapter 5 concludes the thesis and gives future directions for research.

1.1 Related Work

In supervisory control theory the controlled process is described as the generator of a formal language, while the controller, or supervisor, is constructed from a recognizer for a specified target language that represents the desired closed-loop system behavior. It is generally required that supervision yield a system that is nonblocking – in the sense that a set of prespecified 'marked' states is reachable from any other reachable state. A nonblocking modular supervisor synthesis procedure is proposed in [49]. In [26], decentralized supervisory control was introduced. Decentralized supervisory control deals with the idea of local supervisors that respectively ensure the satisfaction of local specifications. In this setting, the effectiveness of distributed local supervision is compared to that of global supervision. In general the main goal of any modular or decentralized supervisor synthesis method is reducing the complexity of the design procedure [43]. However, such approaches may yield a blocking supervisor owing to the conflict among individual local supervisors.

Some algorithms for synthesis of supervisors for parametrized DES are available [3, 5]. However none of them yield nonblocking supervisors. In fact it is shown in [44] that solvability of the decentralized nonblocking synthesis problem is undecidable even when the generator is represented as a finite automaton.

Blocking states can interpreted as representing deadlock or livelock. Deadlock occurs when the system reaches a non-marking state from which any more transition is impossible. On the other hand livelock occurs when the system is trapped in endless cycles that do not lead to a marking state. Detecting deadlocks in distributed networks is a classic problem which is extensively studied. Most deadlock detection methods use exhaustive search over global system structure to find deadlocked states [8]. None of the available algorithms is suitable for finding the set of blocking states in a general parametrized network with arbitrarily large number of subsystems.

Model checking of parameterized networks is challenging and generally undecidable [1], but computer scientists have developed techniques for model checking a variety of subclasses of parameterized networks [10, 13–16]. A common approach is establishing *cut-offs* [10, 13, 14, 16]. Cut-offs are relatively small bounds on parameter values such that once the property of interest is successfully verified in every system limited by the cut-off bounds, the property is guaranteed to hold in the original parameterized network. Cut-offs are usually highly specialized to certain classes of parameterized networks and properties. Emerson and Kahlon [13] establish model-checking cut-offs for networks consisting of a finite number of classes of isomorphic subprocesses, where interaction is modeled via shared variables, transitions in one subprocess being enabled by a guard formula depending on the states of other subprocesses. But here the formal logic considered – a particular fragment of the branching-time temporal logic CTL^* minus the 'next' operator ($CTL^* \setminus X$) – cannot in general express the possibility or impossibility of deadlock of a complete network [45]. In [10], a restrictive token-passing modeling framework and linear-time temporal logic without the next operator ($LTL \setminus X$) are considered. However, $LTL \setminus X$ cannot express the (im)possibility of deadlock [14].

Again using cut-off techniques, [14–16] consider model checking of parameterized networks with the topology of a ring. Specifically, [16] studies the case of component subprocesses that interact only through the passing of a single 'unary' token, which carries no information other than its presence, and travels only in one direction around the ring. As in [13], the logic is a particular fragment of $CTL^* \setminus X$ that cannot in general express the (im)possibility of deadlock of the network. In [15], a given token can be passed only a bounded number of times, and the logic is the same as in [10] and cannot express the (im)possibility of deadlock. On the other hand, [14] addresses deadlock explicitly, but the model is once again a restrictive token-passing framework, one in which each subprocess shares a token with each neighbor in the ring, and must collect both of them before releasing either. With these restrictions, it is shown that deadlock analysis for a ring network can be reduced to that of a ring of at most five subprocesses. In contrast, our analysis is in essence based on the study of the synchronous product of just two subprocesses.

As an extension of [32], we show that the problem of checking the deadlock freedom property in parameterized networks of finite-state systems is generally undecidable [58]. Most deadlock detection methods use exhaustive search over the global system structure to find blocking states [8]. There are some algorithms available that exploit symmetry of the system for deadlock detection. For instance in [28], states with similar paths to possible deadlock are merged into a 'virtual' state. It is shown that forming virtual states reduces the complexity of deadlock detection. This method is not effective for detecting deadlock states in parameterized networks with arbitrarily large numbers of processes. There exist parameterized verification methods that use other abstraction techniques [23, 25]. These verification procedures can be used in finite state network systems with an arbitrarily large number of processes, however achieving a suitable abstracted model is a challenge and usually entails an iterative procedure called 'guard-strengthening'. The abstracted model 'over-approximates' the behaviors of an arbitrary number of processes. In the guard-strengthening procedure an insightful engineer has to modify the abstracted model at each step. If such a procedure results in satisfaction of the safety property, one may conclude that the property holds in the original parameterized network. However, there is no guarantee that a suitable abstraction can be found and this non-automated procedure may never terminate.

Synthesizing nonblocking supervisors for distributed systems has been addressed before. However designing such a supervisor for a parametrized networks raises some serious challenges that have only been partially addressed within the supervisory control and model-checking literature.

Attie and Emerson addressed the problem of synthesizing concurrent programs with arbitrary process interconnection schemes from temporal logic specifications [3]. They synthesize a system with N subsystems from a 'pair-system'. Assuming that the pair-system has the desired correctness property, they show that the global system with N subsystems also has that property if certain technical assumptions are satisfied. Therefore their method depends on the existence of synthesis method for a pair-system. The result of this synthesis method must satisfy the symmetry assumptions, (i.e. symmetrical with respect to interchanging subsystems). For deadlock freedom, they assumed that the pair-system is deadlock free, and satisfies a 'wait-for-graph' condition. In this case they guarantee that the global system with N subsystems is also deadlock free. Roughly speaking, the wait-for-graph assumption guarantees that circular wait chains cannot form.

In [5], Bherer *et al.* proposed a control scheme for parameterized discrete event systems when specifications are given in terms of predicates and are isomorphic for all subsystems. They used the synthesis idea in [3] to design a modular supervisor for each subsystem offline, and generalize it in an online procedure. Although the method reduces the complexity of design of centralized control, it does not deal with blocking or deadlock freedom at all.

The control synthesis procedure proposed by Bherer *et al.* in [5] does not address blocking issues. In fact, the general problem of checking the nonblocking property in parameterized networks is undecidable. In the special case where all subprocesses are identical, it was shown in [32] that the problem of checking blocking is decidable, if and only if there are no *broadcast* messages. Indeed, the case of identical subprocesses without broadcast messages reduces to the *home space* problem for Petri nets. However, the case where the subprocesses are isomorphic but not identical does not admit such a reduction as the corresponding undecidability result shows [32].

1.2 Preliminaries

1.2.1 Discrete Event Systems Basics

One of the conventional ways of presenting a DES is using state machines or *generators* [50]. In the following we shall use the terms generator and automaton interchangeably. A nondeterministic generator is formally defined as a 5-tuple $G = (X, \Sigma, \xi, x^0, X_m)$, where X is a state set, Σ a finite alphabet representing a finite event set, $\xi : X \times \Sigma \to 2^X$ a nondeterministic transition relation (where 2^X is the power set of X), x^0 an initial state[1], and X_m a marked state set. The condition $\xi(x, \sigma) \neq \emptyset$ means that the transition $\xi(x, \sigma)$ is defined. We denote by Σ^+ the set of all nonempty finite strings of events in Σ, and $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$, where ϵ denotes the empty string (the identity element for string concatenation). The transition function extends to $\xi : X \times \Sigma^* \to 2^X$ in a standard manner [50].

The *closed behavior* of G is the sublanguage $L(G) = \{s \in \Sigma^* | \xi(x^0, s) \neq \emptyset\}$. In some cases, it is convenient to change the initial state of a given generator to study the language generated from a different initial state. The language generated by finite transition structure G, when the initial state is x, is denoted by $L(G^{\to x})$. Formally, $L(G^{\to x}) = \{s \in \Sigma^* : \xi(x, s) \neq \emptyset\}$. The *marked behavior* of G is denoted by $L_m(G)$ and defined as $\{s \in L(G) | \xi(x^0, s) \neq \emptyset \;\&\; \xi(x^0, s) \cap X_m \neq \emptyset\}$. The language $\overline{L_m(G)}$ consists of all prefixes of strings in $L_m(G)$. The generator G (and its corresponding DES) is *nonblocking* if $L(G) = \overline{L_m(G)}$, meaning that all generated strings extend to marked strings.

The *natural projection* [50] onto event set $\hat{\Sigma} \subseteq \Sigma$ is denoted by $P_{\hat{\Sigma}}$ and defined as $P_{\hat{\Sigma}} : \Sigma^* \to \hat{\Sigma}^*$, such that

$$\begin{aligned} &P_{\hat{\Sigma}}(\epsilon) = \epsilon; \; P_{\hat{\Sigma}}(\alpha) = \begin{cases} \alpha, & \textit{if } \alpha \in \hat{\Sigma}; \\ \epsilon, & \textit{if } \alpha \notin \hat{\Sigma}; \end{cases} \\ &P_{\hat{\Sigma}}(\alpha\beta) = P_{\hat{\Sigma}}(\alpha) P_{\hat{\Sigma}}(\beta). \end{aligned}$$

Intuitively, $P_{\hat{\Sigma}}$ 'erases' event symbols that do not belong to $\hat{\Sigma}$. The natural projection inverse-image map is $P_{\hat{\Sigma}}^{-1} : \hat{\Sigma}^* \to \Sigma^*$ such that $P_{\hat{\Sigma}}^{-1}(s) = \{t \in \Sigma^* : P_{\hat{\Sigma}}(t) = s\}$.

The language generated by the *synchronous product* of two given languages L_1 and L_2 is defined by $L_1 \| L_2 := P_{\Sigma_1}^{-1}(L_1) \cap P_{\Sigma_2}^{-1}(L_2)$.

$$L(G_1 \| G_2) = P_{\Sigma_1}^{-1}(L(G_1)) \cap P_{\Sigma_2}^{-1}(L(G_2))$$

The converse of a binary relation $R \subseteq X_1 \times X_2$ is given by $R^c := \{(x_2, x_1) \in X_2 \times X_1 : (x_1, x_2) \in R\}$. For a given set S, $|S|$ denotes the cardinality of that set. For any language L, $[L]_n$, $0 \leq n$ denotes the set of elements of L with length n. $\mathbb{N}$ is the set of natural numbers. For two binary relations R_1 and R_2, $R_2 \circ R_1$ is the composition of the two relations. Formally, $R_2 \circ R_1 = \{(x_3, x_1) : \exists (x_3, x_2) \in R_2 \;\&\; (x_2, x_1) \in R_1\}$.

[1]Our notation x^0 for the initial state is not the typical one: we write 0 as a superscript because we reserve subscripts on state symbols to represent components of tuples of states – e.g. $x = (x_1, x_2)$.

1.2.2 Graphs

For the purpose of this thesis, a directed graph $\mathscr{D}$ is an ordered pair (V, A), where V is the vertex set and A is the set of ordered pairs of vertices called arcs. Considering an arc (u_1, u_2), u_2 is a direct successor of u_1, and u_1 is a direct predecessor of u_2. For two vertices $u_0, u_k \in V$, a $u_0 - u_k$ walk is an alternating sequence $u_0, a_1, u_1, a_2, ..., a_k, u_k$ of vertices and arcs such that $a_i = (u_{i-1}, u_i)$, for $1 \leq i \leq k$. A nontrivial walk contains at least one arc. Direct successors of a vertex u_0 are vertices that can be reached from u_0 by a walk containing exactly one arc. A directed graph $\mathscr{D}$ is strongly connected if for every pair $u, v \in V$, $\mathscr{D}$ contains both $u - v$ and $v - u$ walks. For more information on graph theory see [9].

Notation: the converse of a binary relation $R \subseteq X_1 \times X_2$ is given by $R^c := \{(x_2, x_1) \in X_2 \times X_1 : (x_1, x_2) \in R\}$. For two binary relations R_1 and R_2, $R_2 \circ R_1$ is the composition of the two relations. Formally, $R_2 \circ R_1 = \{(x_3, x_1) : \exists (x_3, x_2) \in R_2 \; \& \; (x_2, x_1) \in R_1\}$. For any language L, L_n, $0 \leq n$ denotes the set of elements of L with length n.

Chapter 2

Weak invariant simulation: Properties and algorithms

Process algebra was initially developed by Milner in the form of the Calculus of Communicating Systems (CSS) [30]. It represents a widely accepted framework for modeling and analysis of concurrent systems [4, 19]. Its semantics defined in terms of simulation relations that describe similarities in the behavior of different processes. For the purpose of this thesis, we introduce the new process relation *weak invariant simulation*.

In this chapter we define the notion of weak invariant simulation for a nondeterministic PDES and give some insight into its properties. It should be emphasized that our simulation relation is not intended to provide a useful new semantics for concurrent systems. Rather, it is adapted to a particular task of analyzing synchronous products, as presented in Chapter 3. We nevertheless compare weak invariant simulation to other simulation relations reported in the literature. Moreover, we propose an efficient procedure to check whether a process invariantly weakly simulates another process with respect to a specific subalphabet. The greatest lower bound of all weak invariant simulations between two subprocesses is also introduced. As an application of weak invariant simulation, we define a nontrivial class of PDES for which the deadlock freedom property (see Chapters 3 and 4.)

2.1 Weak invariant Simulation

2.1.1 Invariant simulation: a comparative perspective

In this section we define a new simulation relation: *Invariant Simulation.* This relation is a key element in our characterization of a PDES network model with a decidable deadlock-freedom property.

Simulation relations originated in [30]. In addition, [19] and [4] report *completed simulation*, *ready simulation*, *two-nested simulation* and *bisimulation* as distinct simulation relations. We first state the definitions of these relations from [4], and then define *invariant simulation*, which originates here. Finally we relate all types of simulation by comparing them with respect to subset inclusion.

Consider generators $G_i = (X_i, \Sigma_i, \xi_i, x_{0i}, X_{mi})$, $i = 1, 2$. We assume no relationship between the alphabets Σ_1 and Σ_2, but if their intersection is empty, then the only simulation relations between G_1 and G_2 will be trivial ones.

Definition 1. A *simulation* is a binary relation $\mathcal{S} \subseteq X_1 \times X_2$ between states of the two generators G_1 and G_2 such that for each $(x_1, x_2) \in \mathcal{S}$ and every $\alpha \in \Sigma_2$, if $\hat{x}_2 \in \xi_2(x_2, \alpha)$, then there exists $\hat{x}_1 \in \xi_1(x_1, \alpha)$ and $(\hat{x}_1, \hat{x}_2) \in \mathcal{S}$. Intuitively this means by induction that any sequence of events undertaken by G_2 from x_2 can also be executed by G_1 from x_1.

– A simulation $\mathcal{C}$ is a *completed simulation* if for each $(x_1, x_2) \in \mathcal{C}$, x_1 is a deadlocked state if and only if x_2 is a deadlocked state.

– A simulation $\mathcal{R}$ is a *ready simulation* if for each $(x_1, x_2) \in \mathcal{R}$, we have

$$(\forall \alpha \in \Sigma_1)[\xi_1(x_1, \alpha) \neq \emptyset \Rightarrow \xi_2(x_2, \alpha) \neq \emptyset]. \tag{2.1}$$

This additional condition means that the sets of immediately executable events in x_1 and x_2 are exactly the same.

– A simulation $\mathcal{T}$ is a *two-nested simulation* if $\mathcal{T}^c$ is contained in a simulation. Two-nested simulation of x_2 by x_1 inductively means that x_1 can imitate execution of events by x_2 and vice-versa.

– A simulation $\mathcal{B}i$ is a *bisimulation* if its converse $\mathcal{B}i^c$ is also a simulation.

Next, we formally define invariant simulation as a new and distinct simulation relation.

Definition 2. A simulation relation $\mathcal{IS} \subseteq X_1 \times X_2$ is *invariant* if for any pair $(x_1, x_2) \in \mathcal{IS}$ and for all $\alpha \in \Sigma_2$, we have

$$(\forall \hat{x}_1 \in \xi_1(x_1, \alpha))(\forall \hat{x}_2 \in \xi_2(x_2, \alpha))[(\hat{x}_1, \hat{x}_2) \in \mathcal{IS}]. \tag{2.2}$$

This 'invariance' property simply asserts that, as long as the two generators execute the same events, the simulation relation is preserved. Note that if the second universal quantifier in (2.2) is replaced with an existential quantifier, the result is simply the definition of an ordinary simulation relation.

By virtue of their 'invariance' property, the particular invariant simulation relations employed in our application will be preserved whenever shared events between the respective generators are synchronized. For this property, it will suffice to use a weak version of invariant simulation which will be introduced shortly.

We use infix notation for simulation relations. Assume $\mathcal{H}$ is one of the aforementioned simulation relations. $G_1 \mathcal{H} G_2$ denotes simulation of generator G_2 by G_1. Generator G_1 simulates G_2 if there exists simulation $\mathcal{H}$ such that $(x_{0_1}, x_{0_2}) \in \mathcal{H}$.

Generators G_1 and G_2 in Figure 2.1(b) demonstrate the difference between two-nested simulation and bisimulation. In this example which was taken from [38], G_1 is two-nested simulating G_2 and G_2 is two-nested simulating G_1, but there is no bisimulation relation between the two generators.

The following proposition helps us to compare the newly defined invariant simulation to other simulation relations by means of the partial ordering of subset inclusion.

Proposition 1. The following relationships among different types of simulation relations exist:

(a) Every ready simulation is a completed simulation.

(b) Every two-nested simulation is a ready simulation.

(c) An invariant simulation need not be a two-nested simulation, a ready simulation or a completed simulation and vice versa.

(d) An invariant simulation need not be a bisimulation and vice versa.

Proof. (a) By definition of ready simulation and completed simulation.

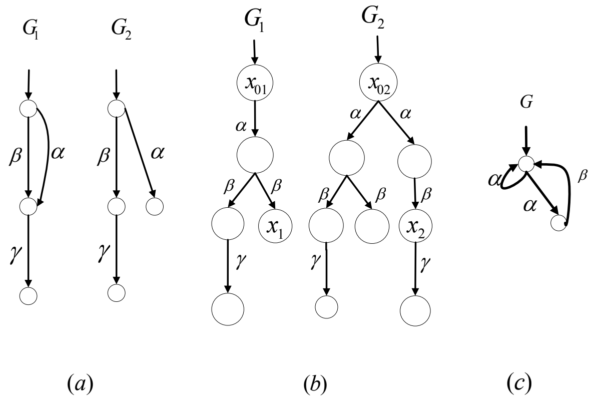

Figure 2.1: Examples demonstrating incomparability of invariant simulation with two-nested simulation, completed simulation and bisimulation: (a) Subprocess G_1 is invariantly simulating G_2, but there is no completed simulation of G_2 by G_1 (and no simulation of G_1 by G_2). (b) Process G_1 is two-nested simulating G_2, but there is no invariant simulation of G_2 by G_1. Furthermore, the pair (x_{01}, x_{02}) does not belong to a bisimulation. (c) An example showing that bisimulation does not imply invariant simulation: The identity relation on the states of generator G is a bisimulation, but not an invariant simulation.

(b) Let $\mathcal{T}$ be a two-nested simulation. Assume $G_1 \mathcal{T} G_2$, or equivalently $(x_{01}, x_{02}) \in \mathcal{T}$. By definition of two-nested simulation, (x_{02}, x_{01}) is contained in a simulation, therefore for all $\alpha \in \Sigma_1$, by the definition of simulation, if $\xi_1(x_{01}, \alpha) \neq \emptyset$, iff $\xi_2(x_{02}, \alpha) \neq \emptyset$. Therefore G_1 is ready simulating G_2.

(c) The proof is done by counterexamples. Consider the generators presented in Figure 2.1(a). Process G_1 invariantly simulates G_2, but there is no completed simulation of G_2 by G_1. Therefore, invariant simulation is not stronger than completed simulation. According to this, using parts (a) and (b) of this Proposition, we conclude that invariant simulation is not stronger than two-nested or ready simulation.
Conversely, consider the generators in Figure 2.1(b), which was originally presented in [38]. It easy to check that G_1 is two-nested simulating G_2. Furthermore, G_1 is completed simulating G_2. On the other hand, if (x_{01}, x_{02}) belongs to an invariant simulation relation then by the definition of invariant simulation, state pair (x_1, x_2) should also belong to an invariant simulation, which is not the case in this example. Therefore G_1 is not invariantly simulating G_2.

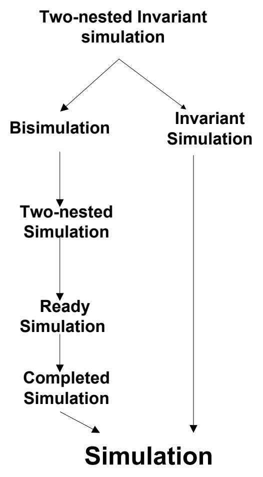

Figure 2.2: The relationships among our notion of invariant simulation, bisimulation and other types of simulation relations.

(d) By the example of Figure 2.1(a), the converse of an invariant simulation need not be a simulation. Hence invariant simulation does not imply bisimulation. On the other hand, the identity relation for the generator presented in Figure 2.1(c) is a bisimulation, but this generator does not invariantly simulate itself.

□

To complete the picture of the relationship of invariant simulation to other simulation relations, it is convenient to define a *two-nested invariant simulation* as an invariant simulation whose converse is contained in another invariant simulation. It turns out that such a relation is also a bisimulation:

Proposition 2. Consider invariant simulations $\mathcal{IS}_1 \subseteq X_1 \times X_2$ and $\mathcal{IS}_2 \subseteq X_2 \times X_1$, then $\mathcal{IS}_1 \cap \mathcal{IS}_2^c \subseteq X_1 \times X_2$ is a bisimulation. Consequently, if $\mathcal{IS}_1$ is a two-nested invariant simulation, with $\mathcal{IS}_1^c \subseteq \mathcal{IS}_2$, then $\mathcal{IS}_1$ is a bisimulation.

Proof. First we show that $\mathcal{IS}_1 \cap \mathcal{IS}_2^c$ is a simulation. Suppose $(x_1, x_2) \in \mathcal{IS}_1 \cap \mathcal{IS}_2^c$ and $\xi_2(x_2, \alpha) \neq \emptyset$. Then since $\mathcal{IS}_1$ is a simulation we have

$$\xi_1(x_1, \alpha) \neq \emptyset, \qquad (\because \mathcal{IS}_1 \text{ is a simulation})$$

and due to invariance of $\mathcal{IS}_1$ and $\mathcal{IS}_2$,

$$(\forall x_1' \in \xi_1(x_1, \alpha))(\forall x_2' \in \xi_2(x_2, \alpha))[(x_1', x_2') \in \mathcal{IS}_1 \cap \mathcal{IS}_2^c]$$

By the symmetric argument, $(\mathcal{IS}_1 \cap \mathcal{IS}_2^c)^c = \mathcal{IS}_1^c \cap \mathcal{IS}_2$ is also a simulation. Hence $\mathcal{IS}_1 \cap \mathcal{IS}_2^c$ is bisimulation.

Now, if $\mathcal{IS}_1$ is a two-nested invariant simulation, then for some such $\mathcal{IS}_2$, $\mathcal{IS}_1^c \subseteq \mathcal{IS}_2$; that is, $\mathcal{IS}_1 \subseteq \mathcal{IS}_2^c$. Hence $\mathcal{IS}_1 = \mathcal{IS}_1 \cap \mathcal{IS}_2^c$, a bisimulation. $\square$

The above proposition implies that two-nested invariant simulation is stronger than bisimulation. The Hasse diagram of Figure 2.2 displays the relationships among all the afore-mentioned types of simulation.

The next proposition states that the property of invariant simulation is preserved under relational composition with a bisimulation.

Proposition 3. Let $\mathcal{B}i \subseteq X_1 \times X_1$ be a bisimulation and $\mathcal{IS} \subseteq X_1 \times X_2$ be an invariant simulation. The composition $\mathcal{B}i \circ \mathcal{IS} \subseteq X_1 \times X_2$ is also an invariant simulation.

Proof. By Proposition 1, bisimulation is stronger than all types of simulation relations except invariant simulation. The proof for these relations is immediate. For the case that $\mathcal{I}$ is an invariant simulation the proof is as follows:

It is easy to check that $\mathcal{B}\circ\mathcal{I}$ is a simulation. We will show that it is in fact invariant . For this purpose, consider three subprocesses G_0, G_1 and G_2. Let x_0, x_1 and x_2 be states of the respective subprocesses such that $(x_0, x_1) \in \mathcal{B}$ and $(x_1, x_2) \in \mathcal{I}$. Then $(x_0, x_2) \in \mathcal{B} \circ \mathcal{I}$. If $\Sigma_1 \cap \Sigma_2 = \emptyset$, the proof is trivial. Suppose therefore that for some $\sigma \in \Sigma_1 \cap \Sigma_2$, $\hat{x}_0 \in \xi_0(x_0, \sigma)$ and $\hat{x}_2 \in \xi_1(x_2, \sigma)$. Because $\mathcal{B}$ is a bisimulation, there must exist $\hat{x}_1 \in \xi_1(x_1, \sigma) \neq \emptyset$, such that

$$(\hat{x}_0, \hat{x}_1) \in \mathcal{B}.$$

But $\hat{x}_2 \in \xi_1(x_2, \sigma)$, and $(x_1, x_2) \in \mathcal{I}$, therefore $(\hat{x}_1, \hat{x}_2) \in \mathcal{I}$. On the other hand, $(\hat{x}_0, \hat{x}_1) \in \mathcal{B}$, hence $(\hat{x}_0, \hat{x}_2) \in \mathcal{B} \circ \mathcal{I}$. In other words, simulation $\mathcal{B} \circ \mathcal{I}$ is invariant. Hence for all $s_0 \in \Sigma_0^*$ and $s_2 \in \Sigma_2^*$, if $\xi_0(x_0, s_0) \neq \emptyset$ and $\xi_2(x_2, s_2) \neq \emptyset$, we have

$$\begin{aligned}&(P_{\hat{\Sigma}}(s_2) = P_{\hat{\Sigma}}(s_0))\\ &\qquad\qquad \Rightarrow (\xi_0(x_0, s_0), \xi_2(x_2, s_2)) \in \mathcal{B} \circ \mathcal{I}.\end{aligned}$$

$\square$

2.1.2 Weak invariant simulation: definition

In this subsection we define a weak version of invariant simulation, which proves to be useful in our subsequent analysis. Consider generators $G_i = (X_i, \Sigma_i, \xi_i, x_{0i}, X_{mi})$, $0 < i \leq 2$ and a natural projection $P_{\hat{\Sigma}} : \Sigma^* \to \hat{\Sigma}^*$ with $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\hat{\Sigma} \subseteq \Sigma$.

Definition 3. A *weak simulation* of G_2 by G_1 with respect to $\hat{\Sigma}$ is a binary relation $\mathcal{WS} \subseteq X_1 \times X_2$ between states of the two generators G_1 and G_2 such that for each $(x_1, x_2) \in \mathcal{WS}$ and every $l_2 \in \Sigma_2^*$, if $\hat{x}_2 \in \xi_2(x_2, l_2) \neq \emptyset$, there exists $l_1 \in \Sigma_1^*$ such that $\hat{x}_1 \in \xi_1(x_1, l_1)$ and the following hold

1. $P_{\hat{\Sigma}}(l_2) = P_{\hat{\Sigma}}(l_1)$
2. $(\hat{x}_1, \hat{x}_2) \in \mathcal{WS}$

A *weak bisimulation* w.r.t. $\hat{\Sigma}$ is a weak simulation w.r.t. $\hat{\Sigma}$ whose converse is also a weak simulation w.r.t. $\hat{\Sigma}$.

Definition 4. Let $\mathcal{I}$ be a weak simulation relation of G_2 by G_1 with respect to $\hat{\Sigma}$. The weak simulation relation $\mathcal{I}$ is a *weak invariant simulation* w.r.t. $\hat{\Sigma}$ if for any pair $(x_1, x_2) \in \mathcal{I}$ and for all $l_1 \in \Sigma_1^*$ and $l_2 \in \Sigma_2^*$ and all $\hat{x}_1 \in \xi_1(x_1, l_1)$ and $\hat{x}_2 \in \xi_2(x_2, l_2)$, we have

$$P_{\hat{\Sigma}}(l_1) = P_{\hat{\Sigma}}(l_2) \Rightarrow (\hat{x}_1, \hat{x}_2) \in \mathcal{I}.$$

Generator G_1 *weakly invariantly simulates* generator G_2 if $(x_{0_1}, x_{0_2}) \in \mathcal{I}$, where $\mathcal{I}$ is a weak invariant simulation w.r.t. $\hat{\Sigma}$.

Remark 1. The weak versions of the simulation relations can be considered to be the same as the original versions, but applied to the following modified (generally nondeterministic) generator, in which event labels have intuitively been projected onto the subalphabet $\hat{\Sigma}$:

$$G_{wi} := (X_i, \hat{\Sigma} \cap \Sigma_i \cup \{\epsilon\}, \xi_{wi}, x_{0i}, X_{mi}),$$

where

$$\begin{aligned} \xi_{wi} :& X_i \times (\hat{\Sigma} \cap \Sigma_i) \cup \{\epsilon\} \to 2^{X_i} \\ (x, \sigma) &\mapsto \bigcup \{\xi_i(x, s) : P_{\hat{\Sigma}}(s) = \sigma\}. \end{aligned}$$

In other words, the G_{wi} features the same state set as G_i; but whenever G_i has a sequence of transitions from x_1 to x_2 labelled by a string $s \in \Sigma_i^*$, if $P_{\hat{\Sigma}}(s) = \sigma \in (\hat{\Sigma} \cap \Sigma_i) \cup \epsilon$, then G_{wi} has a state transition from x_1 to x_2 labelled by σ.

Remark 2. For deadlock analysis of interacting generators G_1 and G_2, weak invariant simulation relations with respect to $\hat{\Sigma}$ are particularly useful when $\Sigma_1 \cap \Sigma_2 \subseteq \hat{\Sigma}$. In the case where $\hat{\Sigma}$ does not contain all the shared events between G_1 and G_2, simulation of G_2 by G_1 does not give us enough information regarding possible deadlocks caused by interaction between these two subprocesses.

The next corollary is an extension of Proposition 1. It states that with respect to a specific set, for instance $\hat{\Sigma}$, the composition of a weak bisimulation and any type of weak simulation relation is a simulation of the same type with respect to $\hat{\Sigma}$.

Corollary 1. Let $\mathcal{I}$ be a weak invariant simulation w.r.t. $\hat{\Sigma}$ and $\mathcal{B}$ be a weak bisimulation relation with respect to $\hat{\Sigma}$. Composition $\mathcal{B} \circ \mathcal{I}$ is a weak invariant simulation w.r.t. $\hat{\Sigma}$.

Proof. By Proposition 3 and Remark 1. □

2.1.3 Properties of weak invariant simulation

In this section, we investigate properties of weak invariant simulation in greater detail. In particular, we introduce an algorithm to decide weak invariant simulation of a given generator by another.

The results of this section will be used extensively in the analysis of parameterized networks in the second part of the thesis (see Chapters 3 and 4). To prepare for these applications, we introduce some of our notation for parameterized networks here. In particular, we denote generators G_i, where i is an integer subscript; we shall similarly index our weak invariant simulations and the corresponding subalphabets. For example, $\mathcal{I}_i$ will denote a weak invariant simulation relation between G_{i-1} and G_i with respect to a subalphabet denoted $\hat{\Sigma}_i$.

Under synchronization of shared events, a weak invariant simulation of G_i by G_{i-1} with respect to $\hat{\Sigma}_i \subseteq \Sigma_{i-1}$ is preserved by transitions that do not involve events in the set $\hat{\Sigma}_i \setminus \Sigma_i$. The following proposition aims to show this.

Proposition 4. Consider two arbitrary generators $G_j = (X_j, \Sigma_j, \xi_j, x_j^0, X_{mj})$, $j \in \{i-1, i\}$, that are components of a synchronous product $G = \Big\|_{k=1}^{N} G_k = (X, \Sigma, \xi, x^0, X_m)$, with $N \geq i$. For any state $x \in X$ of the synchronous product, let x_{i-1} and x_i respectively denote the corresponding states of G_{i-1} and G_i. If $\mathcal{I}_i$ is a weak invariant simulation relation w.r.t.

$\hat{\Sigma}_i \subseteq \Sigma_{i-1}$, then

$$(\forall x \in X)(\forall s \in ((\Sigma \setminus \hat{\Sigma}_i) \cup \Sigma_i)^*) \\ [(x_{i-1}, x_i) \in \mathcal{I}_i \;\&\; \hat{x} \in \xi(x, s) \Rightarrow (\hat{x}_{i-1}, \hat{x}_i) \in \mathcal{I}_i]. \tag{2.3}$$

Proof. The proof is evident from the definition of weak invariant simulation. From $s \in ((\Sigma \setminus \hat{\Sigma}_i) \cup \Sigma_i)^*$, we have $P_{\hat{\Sigma}_i}(s) \in (\Sigma_i \cap \hat{\Sigma}_i)^*$. Therefore $P_{\hat{\Sigma}_i}(P_{\Sigma_i}(s)) = P_{\Sigma_i}(P_{\hat{\Sigma}_i}(s)) = P_{\hat{\Sigma}_i}(s)$. On the other hand, $\hat{\Sigma}_i \subseteq \Sigma_{i-1}$; hence $P_{\hat{\Sigma}_i}(P_{\Sigma_{i-1}}(s)) = P_{\Sigma_{i-1}}(P_{\hat{\Sigma}_i}(s)) = P_{\hat{\Sigma}_i}(s)$. Accordingly, $P_{\hat{\Sigma}_i}(P_{\Sigma_{i-1}}(s)) = P_{\hat{\Sigma}_i}(P_{\Sigma_i}(s))$ and (2.3) is satisfied by the definition of weak invariant simulation. □

To further analyze properties of weak invariant simulation of one generator by another, we define the following structure.

Definition 5. For a given generator G_i, $G_i \upharpoonright \Delta_i$ is the restriction of the generator to a subalphabet Δ_i and is formed by erasing transitions with events that belong to the set $\Sigma_i \setminus \Delta_i$. Formally, $G_i \upharpoonright \Delta_i = (X_i, \Delta_i, \hat{\xi}_i, x_i^0, X_{mi})$ and

$$\hat{\xi}_i(x_i, \sigma) = \begin{cases} \xi_i(x_i, \sigma), & \text{if } \sigma \in \Delta_i; \\ \emptyset, & \text{if } \sigma \in \Sigma_i \setminus \Delta_i. \end{cases}$$

The next result applies whenever the condition $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i \subseteq \Sigma_{i-1}$ is satisfied (see Remark 2). For two finite-state generators G_{i-1} and G_i such that G_{i-1} weakly invariantly simulates G_i w.r.t. $\hat{\Sigma}_i$, this lemma gives a procedure to calculate a binary relation which is the greatest lower bound of all weak invariant simulation relations w.r.t. $\hat{\Sigma}_i$ between G_{i-1} and G_i that include the initial pair (x_{i-1}^0, x_i^0). Specifically, the relation is the state set of the synchronous product of G_i and $G_{i-1} \upharpoonright \Delta_{i-1}$, where Δ_{i-1} is chosen so as to exclude events in $\hat{\Sigma}_i$ that are not common to Σ_{i-1} and Σ_i.

Lemma 1. Consider two arbitrary generators $G_{i-1} = (X_{i-1}, \Sigma_{i-1}, \xi_{i-1}, x_{i-1}^0, X_{mi-1})$ and $G_i = (X_i, \Sigma_i, \xi_i, x_i^0, X_{mi})$ and assume that $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i \subseteq \Sigma_{i-1}$. Let $\Delta_{i-1} = \Sigma_{i-1} \setminus (\hat{\Sigma}_i \setminus \Sigma_i)$ and R_i be the state set of the synchronous product $(G_{i-1} \upharpoonright \Delta_{i-1}) \| G_i$. We have:

(a) The binary relation R_i is contained in all weak invariant simulation relations of G_i by G_{i-1} w.r.t. $\hat{\Sigma}_i$ that include the pair of initial states (x_{i-1}^0, x_i^0).

(b) If R_i is a weak simulation w.r.t. $\hat{\Sigma}_i$, then it is a weak invariant simulation w.r.t. $\hat{\Sigma}_i$.

(c) If generator G_{i-1} weakly invariantly simulates G_i w.r.t. $\hat{\Sigma}_i$, then R_i is a weak invariant simulation.

Proof. *(a)* Consider an arbitrary weak invariant simulation $\mathcal{I}_i$ w.r.t. $\hat{\Sigma}_i$ with $(x^0_{i-1}, x^0_i) \in \mathcal{I}_i$. We will show that R_i is contained in $\mathcal{I}_i$. Consider an arbitrary pair $(x_{i-1}, x_i) \in R_i$. For some $l \in (\Sigma_{i-1} \cup \Sigma_i)^*$, we have $(x_{i-1}, x_i) \in \tilde{\xi}_i((x^0_{i-1}, x^0_i), l)$, where $\tilde{\xi}_i$ is the transition function of automaton $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$. Let $P_{\Sigma_j}(l) = l_j,\ j = i-1, i$. Since $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i \subseteq \Sigma_{i-1}$, by the definition of the transition function $\hat{\xi}_{i-1}$ of $G_{i-1} \upharpoonright \Delta_{i-1}$, we have

$$(\forall t \in L(G_{i-1} \upharpoonright \Delta_{i-1} \| G_i))(P_{\Sigma_{i-1} \cap \Sigma_i}(t) = P_{\hat{\Sigma}_i}(t)). \tag{2.4}$$

Again, from the fact that $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i \subseteq \Sigma_{i-1}$, we have $P_{\hat{\Sigma}_i}(l_i) = P_{\Sigma_{i-1} \cap \Sigma_i}(l_i) = P_{\Sigma_{i-1} \cap \Sigma_i}(l)$; but

$$\begin{aligned} P_{\Sigma_{i-1} \cap \Sigma_i}(l) &= P_{\hat{\Sigma}_i}(l) && \text{(by (2.4))} \\ &= P_{\hat{\Sigma}_i}(l_{i-1}) && (\because \hat{\Sigma}_i \subseteq \Sigma_{i-1}). \end{aligned}$$

Hence, $P_{\hat{\Sigma}_i}(l_i) = P_{\hat{\Sigma}_i}(l) = P_{\hat{\Sigma}_i}(l_{i-1})$. Therefore, by the definition of weak invariant simulation and the fact that $(x^0_{i-1}, x^0_i) \in \mathcal{I}_i$, we have $(x_{i-1}, x_i) \in \mathcal{I}_i$. This shows that $R_i \subseteq \mathcal{I}_i$.

(b) Suppose R_i is a weak simulation w.r.t. $\hat{\Sigma}_i$ and let $(x_{i-1}, x_i) \in R_i$. Consider strings $s_i \in L(G_i^{\to x_i})$ and $s_{i-1} \in L(G_{i-1}^{\to x_{i-1}})$. If $P_{\hat{\Sigma}_i}(s_i) = P_{\hat{\Sigma}_i}(s_{i-1})$, s_{i-1} contains no events in $\hat{\Sigma}_i \setminus \Sigma_i$. Hence $\hat{\xi}_{i-1}(x_{i-1}, s_{i-1}) \neq \emptyset$ (where, again, $\hat{\xi}_{i-1}$ is the transition function of $G_{i-1} \upharpoonright \Delta_{i-1}$). By the structure of the synchronous product $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$, we have

$$(\forall \hat{x}_{i-1} \in \hat{\xi}_{i-1}(x_{i-1}, s_{i-1}))(\forall x'_i \in \xi_i(x_i, s_i))[(\hat{x}_{i-1}, x'_i) \in R_i]. \tag{2.5}$$

Since s_{i-1} and s_i are arbitrary elements of $L(G_{i-1}^{\to x_{i-1}})$ and $L(G_i^{\to x_i})$ that have the same $\hat{\Sigma}_i$ projection, it follows from (2.5) that R_i has the invariance property. Therefore R_i is a weak invariant simulation w.r.t. $\hat{\Sigma}_i$.

(c) If generator G_{i-1} weakly invariantly simulates G_i w.r.t. $\hat{\Sigma}_i$, then there exists a weak invariant simulation w.r.t. $\hat{\Sigma}_i$, containing (x^0_{i-1}, x^0_i). Let H be such a relation. As was shown in part (a) of the proof, $R_i \subseteq H$. Consider an arbitrary pair $(x_{i-1}, x_i) \in R_i \subseteq H$. Fix an arbitrary string $s_i \in \Sigma_i^*$ such that $\xi_i(x_i, s_i) \neq \emptyset$ (note that the empty string is such a string). Since H is a weak simulation w.r.t. $\hat{\Sigma}_i$, and $(x_{i-1}, x_i) \in H$, by the definition of weak simulation there exists $s_{i-1} \in \Sigma_{i-1}^*$ such that $\xi_{i-1}(x_{i-1}, s_{i-1}) \neq \emptyset$ and

$$P_{\hat{\Sigma}_i}(s_i) = P_{\hat{\Sigma}_i}(s_{i-1}) \tag{2.6}$$

$$\Rightarrow P_{\Sigma_{i-1} \cap \Sigma_i}(s_i) = P_{\Sigma_{i-1} \cap \Sigma_i}(s_{i-1}). \tag{2.7}$$

Hence, by the structure of synchronous product $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$, we conclude

$$(\forall x''_{i-1} \in \xi_{i-1}(x_{i-1}, s_{i-1}))(\forall x'_i \in \xi_i(x_i, s_i))[(x''_{i-1}, x'_i) \in R_i]. \tag{2.8}$$

From (2.6) and (2.8), we conclude that R_i is a weak simulation. Therefore by part (b), it is a weak invariant simulation w.r.t. $\hat{\Sigma}_i$. □

Note that if there does exist a weak invariant simulation of G_i by G_{i-1} w.r.t. $\hat{\Sigma}_i$, then, by (c), R_i is such a relation; and by (a), it is the smallest such relation. Hence it is indeed the greatest lower bound of the set of all such relations.

Based on Lemma 1, the next theorem gives an algorithm to check whether a given finite-state generator weakly invariantly simulates another, with respect to a specific subalphabet.

Theorem 1. Consider two arbitrary generators $G_{i-1} = (X_{i-1}, \Sigma_{i-1}, \xi_{i-1}, x^0_{i-1}, X_{mi-1})$ and $G_i = (X_i, \Sigma_i, \xi_i, x^0_i, X_{mi})$ and assume that $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i \subseteq \Sigma_{i-1}$. Let $\Delta_{i-1} = \Sigma_{i-1} \backslash (\hat{\Sigma}_i \backslash \Sigma_i)$ and R_i be the state set of synchronous product $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$. Generator G_{i-1} weakly invariantly simulates G_i w.r.t. $\hat{\Sigma}_i$ if and only if

$$\begin{aligned}&(\forall (x_{i-1}, x_i) \in R_i)\\ &\quad [P_{\hat{\Sigma}_i}(L(G_i^{\rightarrow x_i})) = P_{\hat{\Sigma}_i}(L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\rightarrow (x_{i-1}, x_i)}))].\end{aligned} \tag{2.9}$$

Proof. *(If)* Suppose (2.9) holds; we will show that R_i is a suitable weak invariant simulation. Consider the automaton $(G_{i-1} \upharpoonright \Delta_{i-1}) \| G_i = (R_i, \Sigma_{i-1} \cup \Sigma_i, \tilde{\xi}_{i-1}, x^0_{i-1} \times x^0_i, X_{mi-1} \times X_{mi})$ and let $(x_{i-1}, x_i) \in R_i$. According to (2.9), for any $s_i \in \Sigma_i^*$, we have

$$\begin{aligned}&(\xi_i(x_i, s_i) \neq \emptyset)\\ &\quad \Rightarrow (P_{\hat{\Sigma}_i}(s_i) \in P_{\hat{\Sigma}_i}(L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\rightarrow (x_{i-1}, x_i)}))).\end{aligned}$$

Let w be a string in $L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\rightarrow (x_{i-1}, x_i)})$ such that $P_{\hat{\Sigma}_i}(s_i) = P_{\hat{\Sigma}_i}(w)$. Set $s_{i-1} = P_{\Sigma_{i-1}}(w)$. We have $P_{\hat{\Sigma}_i}(s_i) = P_{\hat{\Sigma}_i}(w)$; therefore, because $\hat{\Sigma}_i \subseteq \Sigma_{i-1}$,

$$\begin{aligned}&P_{\hat{\Sigma}_i}(s_i) = P_{\hat{\Sigma}_i}(P_{\Sigma_{i-1}}(w))\\ \Leftrightarrow &P_{\hat{\Sigma}_i}(s_i) = P_{\hat{\Sigma}_i}(s_{i-1}).\end{aligned} \tag{2.10}$$

So $s_{i-1} \in \Delta^*_{i-1}$. Let $\hat{\xi}_{i-1}$ be the transition function of $G_{i-1} \upharpoonright \Delta_{i-1}$. Since $\hat{\xi}_{i-1}(x_{i-1}, s_{i-1}) \neq \emptyset$, according to the definition of $\hat{\xi}_{i-1}$, we have $\xi_{i-1}(x_{i-1}, s_{i-1}) \neq \emptyset$. We conclude that for all $(x_{i-1}, x_i) \in R_i$ and $s_i \in \Sigma_i^*$, if $\xi_i(x_i, s_i) \neq \emptyset$, then there exists $s_{i-1} \in \Sigma_{i-1}^*$ such that $\xi_{i-1}(x_{i-1}, s_{i-1}) \neq \emptyset$ and $P_{\hat{\Sigma}_i}(s_{i-1}) = P_{\hat{\Sigma}_i}(s_i)$. By the fact that $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i$, we have

$$P_{\Sigma_{i-1}\cap\Sigma_i}(s_i) = P_{\Sigma_{i-1}\cap\Sigma_i}(s_{i-1}). \qquad \textit{(by (2.10))}$$

Again using (2.10), we conclude that s_{i-1} contains no event in $\hat{\Sigma}_i \setminus \Sigma_i$. Therefore, by the structure of the synchronous product $G_{i-1} \restriction \Delta_{i-1} \| G_i$, we have $\{(\xi_{i-1}(x_{i-1}, s_{i-1}) \times \xi_i(x_i, s_i))\} \subseteq R_i$. Hence the binary relation R_i is a weak simulation w.r.t. $\hat{\Sigma}_i$. Indeed, by Lemma 1(b), R_i is a weak invariant simulation w.r.t. $\hat{\Sigma}_i$. Since $(x_{i-1}^0, x_i^0) \in R_i$, the generator G_{i-1} weakly invariantly simulates G_i w.r.t. $\hat{\Sigma}_i$.
(*Only if*) Consider an arbitrary weak invariant simulation $\mathcal{I}_i$ w.r.t. $\hat{\Sigma}_i$, with $(x_{i-1}^0, x_i^0) \in \mathcal{I}_i$. According to Lemma 1(a), R_i is contained in $\mathcal{I}_i$. If $R_i = \emptyset$, then (2.9) holds vacuously. Consider then an arbitrary pair $(x_{i-1}, x_i) \in R_i$. Fix a string $l_i \in L(G_i^{\to x_i})$. Since $(x_{i-1}, x_i) \in \mathcal{I}_i$, there exists $l_{i-1} \in L(G_{i-1}^{\to x_{i-1}})$ such that $P_{\hat{\Sigma}_i}(l_{i-1}) = P_{\hat{\Sigma}_i}(l_i)$. This means that $l_{i-1} \in \Delta_{i-1}^*$, so $l_{i-1} \in L((G_{i-1} \restriction \Delta_{i-1})^{\to x_{i-1}})$. Moreover, because $P_{\hat{\Sigma}_i}(l_{i-1}) = P_{\hat{\Sigma}_i}(l_i)$ and $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i$,

$$P_{\Sigma_{i-1}\cap\Sigma_i}(l_{i-1}) = P_{\Sigma_{i-1}\cap\Sigma_i}(l_i).$$

Hence, according to the restriction of G_{i-1} to event set Δ_{i-1}, and the structure of the synchronous product $G_{i-1} \restriction \Delta_{i-1} \| G_i$, we have

$$(\exists l \in L(G_{i-1} \restriction \Delta_{i-1} \| G_i)^{\to(x_{i-1}, x_i)})(P_{\hat{\Sigma}_i}(l_i) = P_{\hat{\Sigma}_i}(l)).$$

Because $l_i \in L(G_i^{\to x_i})$ was chosen arbitrarily,

$$\begin{aligned}&(\forall l_i \in L(G_i^{\to x_i}))\\ &\quad(\exists l \in L((G_{i-1} \restriction \Delta_{i-1} \| G_i)^{\to(x_{i-1}, x_i)}))[P_{\hat{\Sigma}_i}(l_i) = P_{\hat{\Sigma}_i}(l)].\end{aligned}$$

Hence

$$P_{\hat{\Sigma}_i}(L(G_i^{\to x_i})) \subseteq P_{\hat{\Sigma}_i}(L((G_{i-1} \restriction \Delta_{i-1} \| G_i)^{\to(x_{i-1}, x_i)})). \tag{2.11}$$

Conversely, no string $l \in L((G_{i-1} \restriction \Delta_{i-1} \| G_i)^{\to(x_{i-1}, x_i)})$ contains any symbols in $\hat{\Sigma}_i \setminus \Sigma_i$. Therefore for any such string, $P_{\hat{\Sigma}_i}(l) = P_{\hat{\Sigma}_i \cap \Sigma_i}(l) = P_{\hat{\Sigma}_i}(P_{\Sigma_i}(l))$. Let $P_{\Sigma_i}(l) = l_i$. Because $l_i \in L(G_i^{\to x_i})$, we have

$$\begin{aligned}&(\forall l \in L((G_{i-1} \restriction \Delta_{i-1} \| G_i))^{\to(x_{i-1}, x_i)})\\ &\qquad\qquad(\exists l_i \in L(G_i^{\to x_i}))[P_{\hat{\Sigma}_i}(l) = P_{\hat{\Sigma}_i}(l_i)].\end{aligned}$$

Hence

$$\begin{aligned} &P_{\hat{\Sigma}_i}(L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\rightarrow(x_{i-1},x_i)})) \\ &\qquad\qquad\qquad \subseteq (P_{\hat{\Sigma}_i}(L(G_i^{\rightarrow x_i})). \end{aligned} \tag{2.12}$$

From (2.11) and (2.12), relation (2.9) follows. This completes the proof. □

Whenever both generators are finite, Theorem 1 gives us a method to check whether or not one generator weakly invariantly simulates the other. However, direct checking of condition (2.9) by standard automata-theoretic means is computationally expensive, in the sense that deciding inequivalence of finite-state automata is PSPACE-complete [18]. The next corollary gives an alternative condition which is equivalent to (2.9).

Corollary 2. Let $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i \subseteq \Sigma_{i-1}$, $\Delta_{i-1} = \Sigma_{i-1} \setminus (\hat{\Sigma}_i \setminus \Sigma_i)$, and R_i be the state set of synchronous product $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$ as defined in Theorem 1. Generator G_{i-1} weakly invariantly simulates G_i w.r.t. $\hat{\Sigma}_i$ if and only if the following condition holds for $n = 1$:

$$\begin{aligned} &(\forall (x_{i-1}, x_i) \in R_i)([P_{\hat{\Sigma}_i}(L(G_i^{\rightarrow x_i}))]_n \\ &\qquad\qquad \subseteq [P_{\hat{\Sigma}_i}(L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\rightarrow(x_{i-1},x_i)}))]_n). \end{aligned} \tag{2.13}$$

(Recall that $[.]_n$ denotes the subset of strings of length n)

Proof. By Theorem 1, it suffices to show that the condition (2.13) is satisfied for any $n \geq 0$ if and only if (2.13) is satisfied for $n = 1$.

(Only if) Immediate.

(If) In this part we assume (2.13) holds for $n = 1$, and prove by induction that it holds for all $n \geq 0$. For $n = 0$, condition (2.13) holds trivially. For $n = 1$, (2.13) holds by assumption. This forms the base of the induction. For the induction step, suppose that (2.13) holds for n. Fix $(x_{i-1}, x_i) \in R_i$. Let $s \in \hat{\Sigma}_i^*$ and $\sigma_i \in \hat{\Sigma}_i$ such that $s\sigma_i \in [P_{\hat{\Sigma}_i}(L(G_i^{\rightarrow x_i}))]_{n+1}$ (if no such string exists, the result holds vacuously). Then there exists $t\sigma_i \in L(G_i^{\rightarrow x_i})$ such that $P_{\hat{\Sigma}_i}(t) = s$. By the inductive hypothesis,

$$\begin{aligned} s &\in [P_{\hat{\Sigma}_i}(L(G_i^{\rightarrow x_i}))]_n \\ &\qquad \subseteq [P_{\hat{\Sigma}_i}(L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\rightarrow(x_{i-1},x_i)}))]_n. \end{aligned}$$

This means that there exists $\hat{t} \in L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\rightarrow(x_{i-1},x_i)})$ such that $P_{\hat{\Sigma}_i}(\hat{t}) = s = P_{\hat{\Sigma}_i}(t)$. Now, because $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i$, it follows by the structure of the synchronous product that there exists $\tilde{t} \in L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\to (x_{i-1}, x_i)})$ such that $P_{\Sigma_i}(\tilde{t}) = t$. Note that $\tilde{t}$ contains no symbols in $\hat{\Sigma}_i \setminus \Sigma_i$; therefore

$$P_{\hat{\Sigma}_i}(\tilde{t}) = P_{\hat{\Sigma}_i}(P_{\Sigma_i}(\tilde{t})) = P_{\hat{\Sigma}_i}(t) = s. \tag{2.14}$$

Let $x'_i \in \xi_i(x_i, t) \cap \chi_i(\sigma_i)$ and x'_{i-1} be such that $(x'_{i-1}, x'_i) \in \tilde{\xi}_i((x_{i-1}, x_i), \tilde{t})$. Since $(x'_{i-1}, x'_i) \in R_i$, by assumption we have $\sigma_i \in [P_{\hat{\Sigma}_i}(L(G_i^{\to x'_i}))]_1 \subseteq [P_{\hat{\Sigma}_i}(L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\to (x'_{i-1}, x'_i)}))]_1$. Let $r \in L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\to (x'_{i-1}, x'_i)})$ such that $P_{\hat{\Sigma}_i}(r) = \sigma_i$ and consider therefore a string

$$\tilde{t}r \in L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\to (x_{i-1}, x_i)}).$$

Then $P_{\hat{\Sigma}_i}(\tilde{t}r) \in P_{\hat{\Sigma}_i}(L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\to (x_{i-1}, x_i)}))$. Because $P_{\hat{\Sigma}_i}(r) = \sigma_i$, we have

$$\begin{aligned} P_{\hat{\Sigma}_i}(\tilde{t}r) = P_{\hat{\Sigma}_i}(\tilde{t}) P_{\hat{\Sigma}_i}(r) &= P_{\hat{\Sigma}_i}(t)\sigma_i \\ &= s\sigma_i \qquad\qquad (\textit{by } (2.14)). \end{aligned}$$

Therefore $s\sigma_i \in P_{\hat{\Sigma}_i}(L((G_{i-1} \upharpoonright \Delta_{i-1} \| G_i)^{\to (x_{i-1}, x_i)}))$. Because $s\sigma_i \in [P_{\hat{\Sigma}_i}(L(G_i)^{\to x_i})]_{n+1}$ is arbitrary, (2.13) holds for $n + 1$. This completes the induction. □

Computation of the finite event set on either side of the inclusion of (2.13) requires only the polynomial-time computation of an unobservable reach [20] (with $\hat{\Sigma}_i$ considered to be the observable subalphabet) and computation of the set of all $\hat{\Sigma}_i$ events that may occur within that unobservable reach. Once these events are known, it remains only to check the inclusion. The computation of an unobservable reach in the deterministic case is linear in the product of the cardinalities of the state set and the alphabet [20]. For nondeterministic automata, by Tarjan's depth-first search algorithm [42], it is of order at most the cardinality of the set of transitions, as in Tarjan's procedure each distinct transition is traversed exactly twice. This estimate dominates the complexity of computing the set of $\hat{\Sigma}_i$ events that can occur within a given unobservable reach. Thus, computation of the subalphabet on the right-hand side of the inclusion is $O(|X_{i-1}|^2|X_i|^2|\Sigma_{i-1} \cup \Sigma_i|)$. But computation of all such subalphabets can be performed in a single depth-first search of the synchronous product, so the computation of all the necessary subalphabets is $O(|X_{i-1}|^2|X_i|^2|\Sigma_{i-1} \cup \Sigma_i|)$. Checking inclusion of two event sets is at worst dominated by the complexity of sorting them: $O(\hat{\Sigma}_i)log(|\hat{\Sigma}_i|)$. So the overall time complexity of checking (2.13) is at most $O(|X_{i-1}|^2|X_i|^2|\Sigma_{i-1} \cup \Sigma_i| + |X_{i-1}||X_i||\hat{\Sigma}_i|log(|\hat{\Sigma}_i|))$.

2.2 Conclusion

Our goal is to develop methods to analyze global system properties of parameterized discrete event systems and, in the first instance, of parameterized networks of isomorphic subprocesses. To this end a new mathematical notion called weak invariant simulation has been introduced in this chapter. We also investigated the relationships among weak invariant simulation and existing comparative semantics in process algebra and established the distinctness of weak invariant simulation. Furthermore, we proposed a computationally efficient method to check existence of weak invariant simulation with respect to a specific alphabet between two given generators. Other properties of weak invariant simulation between two given generators are also studied. We established a procedure to decide existence of weak invariant simulation between two generators and calculated the greatest lower bound of such relation between two generators.

Chapter 3

Analysis of parameterized ring discrete event systems

In this chapter we introduce a class of parameterized networks for which the deadlock-freedom property is decidable. The network considered in this chapter has the topology of a ring (each subprocess directly interacts with its two 'neighbors' in the ring.) To motivate the study, we show that the deadlock analysis of parameterized ring networks is undecidable.

Then, to achieve decidability, we restrict the interactions between subprocesses. The structural assumptions are formulated in terms of the new mathematical relation introduced in Chapter 2: weak invariant simulation of one subprocess by another. We first establish some properties of the network model. In particular, we show that our assumptions serve to ensure that while both immediate neighbors of a subprocess may prevent it from executing a shared events, only one neighbor can permanently prevent an event from occurring; in that sense, control only flows around the ring in one direction. Then we propose a procedure for the deadlock analysis of the network. This procedure enables us to determine all the reachable deadlocked states of our parameterized ring network. The effectiveness of the proposed framework is demonstrated by analysis of a version of the dining philosophers problem.

3.1 The network model

3.1.1 Parameterized discrete event systems

For the purposes of this thesis, a PDES $\mathcal{G}$, with parameter $N > 2$, is a set of synchronous products of $M \in \mathbb{N}$ isomorphic finite-state subprocesses modeled by generators. Formally,

$$\mathcal{G} = \{\|_{i=1}^{M} G_i : \ M \in \mathbb{N}\}, \tag{3.1}$$

where

$$G_i = (X_i, \Sigma_i, \xi_i, x_i^0, X_{mi}), \tag{3.2}$$

with $X_1 = X_2 = ... = X_M$. Subalphabet $\Sigma_i = \Sigma_{L_i} \cup \Sigma_{S_i}$, where subsets Σ_{L_i} and Σ_{S_i} contain local (unshared) event and shared event symbols respectively. For a fixed N, let $G^N = \|_{i=1}^{N} G_i = (X, \Sigma, \xi, x^0, X_m)$ be an instance of PDES $\mathcal{G}$ with N subprocesses. In this instance, $\Sigma = \bigcup_{i=1}^{N} \Sigma_i$ is the global alphabet set and $\Sigma_S = \bigcup_{i=1}^{N} \Sigma_{S_i}$ and $\Sigma_L = \Sigma \setminus \Sigma_S$ are shared event and local event subsets respectively. Shared events are shared between exactly two subprocesses: thus a given event symbol σ belongs to at most two of the Σ_{S_i}. Shared events can only occur simultaneously in both such subprocesses. The initial state of the i^{th} subprocess is the i^{th} component of x^0 and denoted by x_i^0. The states of G^N take the form of N-tuples $x = (x_1, x_2, ..., x_N)$, where x_i is the state of the i^{th} subprocess G_i. In this chapter we refer to these N-tuples as 'global states'. The set of all global states is denoted X.

The system parameter N is an arbitrary natural number (greater than 2) that denotes the total number of subprocesses in an instance of a PDES.

The problem of checking the nonblocking property for a PDES consisting of an arbitrary number N of finite-state subprocesses is algorithmically undecidable [32]. In other words, the nonblocking property cannot be mechanically checked for a general network of an arbitrary number of interacting isomorphic finite-state subprocesses. Similarly, the closely related property of deadlock-freedom is undecidable (see Corollary 3 below). In this chapter, we focus on the formulation of a decidable subproblem of checking the deadlock-freedom of PDES.

A shared event between two subprocesses G_{i-1} and G_i, $1 < i \leq N$, can occur only if both subprocesses are in states that allow the shared event. If for instance G_i is in a state that can reach marking states only by executing a shared event which cannot be executed by G_{i-1}, then subprocess G_i is blocked. The following definition helps us to address this issue.

Definition 6. In a PDES, for a shared event σ_i of the i^{th} subprocess, $1 \leq i \leq N$, *companion states* of σ_i in the j^{th} subprocess are states x_j in G_j, for which $\xi_j(x_j, \sigma_i) \neq \emptyset$. The set of such companion states is $\chi_j(\sigma_i)$.

Remark 3. Weak invariant simulation can be used to formulate useful assumptions about the network behavior. Consider PDES $\mathcal{G}$ and an instance G^N of this PDES. For an arbitrary i, j, $1 \leq i, j \leq N$, assume that for a reachable global state $x \in X$, we have $(x_i, x_j) \in \mathcal{I}_i$, where $\mathcal{I}_i$ is a weak invariant simulation with respect to all the shared events of G_i with any other subprocess. By the definition of weak invariant simulation, it can be shown that for any $\sigma_i \in \Sigma_i \cap \Sigma_j$, if $\xi_j(x_j, \sigma_i) \neq \emptyset$, then G_i can reach $\chi_i(\sigma_i)$ by a string l of local events; hence $l\sigma_i$ is executable from x in G^N. In other words, in state x, G_i does not block G_j from executing an event that is shared between them.

Note that if G_i transitions from x_i to another state $\hat{x}_i$ via a shared event that is not in $\Sigma_i \cap \Sigma_j$, then by the definition of weak invariant simulation, $(\hat{x}_i, x_j)$ is not necessarily a member of $\mathcal{I}_i$, and G_i may block G_j. In the next subsection, we propose a tractable parameterized ring network in which for any i, $1 \leq i < N$, and any reachable state, G_i does not permanently block its neighbor with higher index, namely G_{i+1}. To achieve this property, we introduce network assumptions to ensure that regardless of the evolution of the global network, weak invariant simulation of G_{i+1} by G_i w.r.t. all the shared events of G_i can always be (re-)established between states of the two neighbors in the ring (Theorem 2 below proves this property for our proposed ring network model).

3.1.2 The ring network model

In this thesis, a parameterized ring network is a PDES $\mathcal{G}$ that is a set of synchronous products of $N \in \mathbb{N}$ isomorphic finite-state subprocesses arranged in a ring topology, with each subprocess sharing events only with its immediate neighbors in the ring. Indices of subprocesses of the ring network start with one and increase 'clockwise' over the ring till they reach N. From this point on in the chapter terms $i + j$ and $i - j$ are calculated using *modulo-N* arithmetic over the complete residue system $\{1, 2, ..., N\}$. This is the same as standard modulo-N arithmetic, except that the equivalence class of the integers modulo N that contains zero is represented by N instead of zero. Thus in a ring PDES, $\Sigma_i \cap \Sigma_j \neq \emptyset$ only if $i - j = 1$ *or* $i - j = N - 1$. For subprocess G_i with $1 \leq i \leq N$, we assume that the local event symbols in set Σ_{L_i} have no index. Furthermore, shared event symbols in set Σ_{S_i} either have index $i - 1$ or index i: symbols in $\Sigma_{i-1} \cap \Sigma_i$ (shared events between generators G_{i-1} and G_i) have index $i - 1$, while the symbols in $\Sigma_i \cap \Sigma_{i+1}$ (shared events between G_i and G_{i+1}) have index i. In a ring network, the companion states of a shared event of the i^{th} subprocess are states of the neighbor subprocesses G_{i+1} and G_{i-1}.

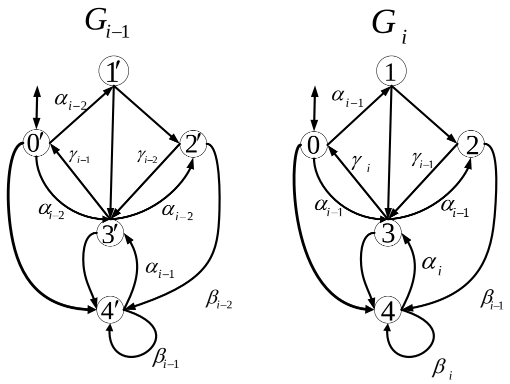

Figure 3.1: The $i-1^{th}$ and i^{th} instance of a PDES that satisfies proposed ring network model. Transitions with local events are unlabeled in this picture. In this PDES, the state set of the synchronous product of G_{i-1} and G_i can be taken to be $\mathcal{V}_i$, a weak invariant simulation of G_i by G_{i-1} w.r.t. $\Sigma_{i-1} \cap \Sigma_i$. $\mathcal{W}_i = \{(0,0),(0,3),(0,4),(3,0),(3,1),(3,2),(3,3),(3,4),(4,0),(4,1),(4,2),$
$(4,3),(4,4)\}$ is a weak invariant simulation of G_i by G_{i-1} w.r.t. the set of all shared events of G_{i-1}. Note that state 1 of G_{i-1} weakly simulates state 1 of G_i w.r.t. $\Sigma_{S_{i-1}}$, but $(1,1)$ is not a member of $\mathcal{W}_i$ since the invariance property fails.

For instance G^N of PDES $\mathcal{G}$ with ring structure, we define a cyclic permutation κ to relate the structures of subprocesses. There exists a bijection $\kappa : \Sigma \to \Sigma$ of order N such that $\forall \alpha \in \Sigma_i, \kappa(\alpha) \in \Sigma_{i+1}$ and the following properties hold:

$$\begin{aligned}
&(\forall \sigma \in \Sigma_{L_i})(\kappa(\sigma) = \sigma);\\
&(\forall \sigma_i \in \Sigma_i \cap \Sigma_{i+1})(\kappa(\sigma_i) = \sigma_{i+1});\\
&(\forall \alpha \in \Sigma_i)(\forall x \in X)(\xi_i(x,\alpha) = \xi_{i+1}(x,\kappa(\alpha))),
\end{aligned}$$

where X_s is the common state set of all the subprocesses of G^N. The first two of these assumptions respectively specify how κ acts on local and shared events. The third specifies the exact sense in which the subprocesses are isomorphic: they are identical up to this permutation of the indices of shared events. To get from a subprocess to its neighboring subprocess with larger index, the cyclic permutation κ should be applied once. This implies that in a ring network G^N, to get to G_j from G_i, it suffices to add $j-i$ to every index of any shared event of subprocess G_i. Subprocesses displayed in Figure 3.1 have this property.

The next corollary states that checking the deadlock-freedom property in a parameterized network with ring topology is undecidable.

Corollary 3. It is undecidable whether the reachable states of a parameterized ring network include deadlocked states.

Proof. By the proof of Theorem 3 of [32], given an arbitrary Turing machine M, a parameterized ring network can be constructed so that, if M halts on the empty input, a sufficiently large instance of the parameterized network can faithfully simulate M's entire run on the empty input. Moreover, the network can be programmed so that it deadlocks if and only if a faithful simulation of M on the empty input shows the Turing machine entering a halting state. (Indeed, in the construction used in [32], when a subprocess detects that the simulation has entered a halting state, that subprocess deadlocks. It is simple to modify the construction so that the deadlocked subprocess also signals all other subprocesses to deadlock.)

It follows that the (undecidable) halting problem reduces to that of checking the possibility of deadlock either of a single subprocess in a parameterized ring network or of an entire parameterized ring network. Both the latter problems are therefore undecidable. □

Given the above result, in order to achieve decidability, we must restrict the generality of our class of problems by introducing some structural assumptions. The rest of the chapter is devoted to showing that the following assumptions suffice.

Any instance G^N, $N > 2$, of our parameterized ring network model $\mathcal{G}$, will be assumed to satisfy the following properties for $1 \leq i \leq N$:

$$(\forall x_i \in X_i)(\forall x_i' \in X_i)(\exists t \in \Sigma_i^*)[\xi_i(x_i, t) = x_i'], \tag{3.3}$$

$$(\forall \sigma_{i-1} \in \Sigma_{i-1} \cap \Sigma_i)(|\chi_{i-1}(\sigma_{i-1})| \leq 1), \tag{3.4}$$

$$G_{i-1}\mathcal{V}_i G_i, \tag{3.5}$$

$$G_{i-1}\mathcal{W}_i G_i. \tag{3.6}$$

where $\mathcal{V}_i$ and $\mathcal{W}_i$ are weak invariant simulation relations of G_i by G_{i-1} w.r.t. $\Sigma_{i-1} \cap \Sigma_i$ and $\Sigma_{S_{i-1}}$ respectively.

Assumptions (3.3) and (3.4) are conditions on the structure of individual subprocesses, while assumptions (3.5) and (3.6) restrict the way subprocesses interact.

Assumption (3.3) ensures that in any individual subprocess there exists a path from any state to any target state. In other words, the transition graph of each subprocess of the network is strongly connected. In the context of nonterminating systems this technical restriction is arguably mild: it simply rules out states that could become permanently inaccessible as the subprocess evolves (even in the absence of synchronization with other subprocesses). Condition (3.4) expresses that in generator G_{i-1}, each shared event in the subset $\Sigma_{i-1} \cap \Sigma_i$ has at most one companion state. This means that potential interactions of G_{i-1} with G_i via a specific shared event in $\Sigma_{i-1} \cap \Sigma_i$ can only occur if G_{i-1} is in that specific state. If (3.4) is not satisfied, suitable enrichment of the event alphabet would make it hold (by distinguishing occurrences of the same event that can occur in distinct states), but this alphabet enrichment could make the remaining assumptions (3.5), (3.6) more stringent.

Assumption (3.5) states that G_{i-1} weakly invariantly simulates G_i with respect to $\Sigma_{i-1} \cap \Sigma_i$ and assumption (3.6) means that G_{i-1} weakly invariantly simulates G_i with respect to $\Sigma_{S_{i-1}}$. These assumptions ensure that the behaviors of G_{i-1} and G_i over the respective subalphabets are in some sense similar. Two weak invariant simulations with respect to different subalphabets are not necessarily comparable; the significance of weak invariant simulation depends critically on the subalphabet $\hat{\Sigma}_i$. In our network model the relations $\mathcal{W}_i$ and $\mathcal{V}_i$ are in general incomparable w.r.t. set inclusion. As will be seen, these assumptions add sufficient structure to make deadlock decidable.

Deadlock often arises through resource contention, and it may be helpful to interpret (3.5) and (3.6) in that context. Events shared between G_{i-1} and G_i can be thought of as transfers of shared resources between G_{i-1} and G_i. By Proposition 5 (below) and the definition of weak invariant simulation, assumption (3.5) ensures that from all the reachable states of the network, if interaction between G_{i-1} and the rest of the network is ignored, subprocess G_{i-1} can always reach companion states of events shared between G_{i-1} and G_i; thus G_{i-1} can always potentially provide the resources necessary for subprocess G_i. Violation of assumption (3.5) means that even if G_{i-1} and G_i are isolated from the network, G_{i-1} may never be able to provide resources needed by G_i. This might indicate a 'design flaw' in network architecture that can easily be identified by forming the synchronous product of G_{i-1} and G_i. On the other hand, if the states of G_{i-1} and G_i are related by $\mathcal{W}_i$, subprocess G_{i-1} can eventually provide any resources requested by G_i. Now, it need not be true that, in every reachable state of the network and for any i, the states of G_{i-1} and G_i are in fact related by $\mathcal{W}_i$; but, as shown in Theorem 2 below, regardless of network interactions the weak invariant simulation relation $\mathcal{W}_i$ can be eventually reestablished between G_{i-1} and G_i.

Remark 4. Checking properties (3.3 - 3.6) for all i, $1 \leq i \leq N$, is not necessary. By symmetry and the fact that the state sets of the subprocesses are identical, the $\mathcal{V}_i$ and $\mathcal{W}_i$ can respectively be chosen to be identical subsets of state pairs for every i. The isomorphic structure of the network then guarantees satisfaction of (3.3 - 3.6) for all i, $1 \leq i \leq N$, if these properties are satisfied for any i. Properties (3.3) and (3.4) can be verified by examining a single subprocess. Properties (3.5) and (3.6) can be checked by Corollary 2.

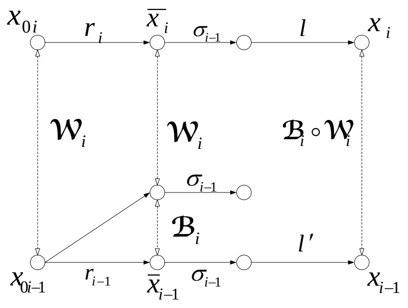

Figure 3.2: The illustration of proof of Lemma 2.

There are practical ring networks exemplified by the dining philosophers problem [24] that can be modeled to comply with the above assumptions. See Section 3.4 for more detail. Figure 3.1 illustrates the $i-1^{th}$ and i^{th} instances of another network that satisfies properties (3.3 - 3.6).

3.2 Properties of The Proposed Network Model

The following results shed light on the properties of the proposed network model and on weak invariant simulation. The first states that relation $\mathcal{V}_i$ in our proposed network model is preserved throughout the evolution of the network:

Proposition 5. For any instance G^N of $\mathcal{G}$ with (3.3 - 3.6), for every $x \in X$ and every i, $1 \le i \le N$, $(x_{i-1}, x_i) \in \mathcal{V}_i$.

Proof. From (3.5), we have for all i, $(x^0_{i-1}, x^0_i) \in \mathcal{V}_i$, where $\mathcal{V}_i$ is a weak invariant simulation w.r.t. $\Sigma_{i-1} \cap \Sigma_i$. Since $(\Sigma_{i-1} \cap \Sigma_i) \subseteq \Sigma_i$, the result follows from Proposition 4. □

On the other hand, because $\mathcal{W}_i$ of assumption (3.6) are weak invariant simulations with respect to $\Sigma_{S_{i-1}}$, rather than $\Sigma_{i-1} \cap \Sigma_i$, they need not be preserved throughout the evolution of the network. However, the following lemma states that, regardless of the evolution of G^N, for some i, $1 \le i \le N$, subsystems G_{i-1} and G_i are in states x_{i-1} and x_i respectively, such that $(x_{i-1}, x_i) \in \mathcal{W}_i$.

Lemma 2. Consider an instance G^N of $\mathcal{G}$ satisfying (3.3 - 3.6). For all $x \in X$ there exists i such that the pair $(x_{i-1}, x_i) \in \mathcal{W}_i$.

Proof. See appendix 3.A □

Corollary 4. Consider an instance G^N of $\mathcal{G}$ satisfying (3.3 - 3.6). For any $s \in L(G^N)$ with $P_{\Sigma_S}(s) \neq \epsilon$, let $x \in \xi(x^0, s)$ and i be such that the last occurrence of any shared event in string s belongs to the set $\Sigma_{i-1} \cap \Sigma_i$. We then have $(x_{i-1}, x_i) \in \mathcal{W}_i$, where x_i is the i^{th} component of $x \in X$.

Proof. By the proof of Lemma 2 and the invariance of $\mathcal{W}_i$. □

If G_{i-1} weakly invariantly simulates G_i w.r.t. $\hat{\Sigma}_i \subseteq \Sigma_{S_{i-1}}$, the next proposition establishes a relationship between any state pair $(x_{i-1}, x_i) \in X_{i-1} \times X_i$ that belongs to a weak invariant simulation w.r.t. $\hat{\Sigma}_i$, and the weak invariant simulation R_i defined in Lemma 1.

Proposition 6. Consider two arbitrary generators $G_{i-1} = (X_{i-1}, \Sigma_{i-1}, \xi_{i-1}, x^0_{i-1}, X_{m_{i-1}})$ and $G_i = (X_i, \Sigma_i, \xi_i, x^0_i, X_{mi})$ in a ring network such that conditions (3.3), (3.4) and (3.6) are satisfied. Let $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i \subseteq \Sigma_{S_{i-1}}$ and $\Delta_{i-1} = \Sigma_{i-1} \setminus (\hat{\Sigma}_i \setminus \Sigma_i)$. Furthermore, let R_i be the state set of synchronous product $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$. For an arbitrary pair $(x_{i-1}, x_i) \in X_{i-1} \times X_i$, if there exists a weak invariant simulation $\mathcal{I}_i \in X_{i-1} \times X_i$ w.r.t. $\hat{\Sigma}_i$, such that $(x_{i-1}, x_i) \in \mathcal{I}_i$, then

$$\begin{aligned}(\exists s_{i-1} \in & (\Sigma_{i-1} \setminus \hat{\Sigma}_i)^*) \\ & (\exists \tilde{x}_{i-1} \in \xi_{i-1}(x_{i-1}, s_{i-1}))[(\tilde{x}_{i-1}, x_i) \in R_i]. \qquad (3.7)\end{aligned}$$

Proof. Suppose that, for all shared events $\sigma_j \in \Sigma_{S_i}$, $j \in \{i-1, i\}$, we have $\chi_i(\sigma_j) = \emptyset$. Then, the restriction of G_{i-1} to $\Delta_{i-1} = \Sigma_{i-1} \setminus (\hat{\Sigma}_i \setminus \Sigma_i)$ has no effect, for there are no transitions defined in G_i for shared events, and, by isomorphism, there are none defined in G_{i-1} either. The synchronous product is therefore a shuffle product, so, for any pair $(x_{i-1}, x_i) \in X_{i-1} \times X_i$, $(x_{i-1}, x_i) \in R_i$. Suppose therefore that for some $\sigma_j \in \Sigma_{S_i}$, $j \in \{i-1, i\}$, $\chi_i(\sigma_j) \neq \emptyset$. Consider a string $s_i \in (\Sigma_i \setminus \hat{\Sigma}_i)^*$ such that for some $\sigma_j \in \Sigma_{S_i}$, $\xi_i(x_i, s_i\sigma_j) \neq \emptyset$ and $P_{\hat{\Sigma}_i}(s_i) = \epsilon$. The existence of such a string s_i is implied by assumption (3.3): if $\chi_i(\sigma_j) \neq \emptyset$, then there is a state in G_i where a σ_j transition is defined; that state is reachable from x_i, by (3.3); either it is reachable by means of local events of G_i or there is a state that is so reachable where some other shared event of G_i can occur. If (x_{i-1}, x_i) belongs to some weak invariant simulation w.r.t. $\hat{\Sigma}_i$, there exists $s_{i-1} \in \Sigma^*_{i-1}$ such that $\xi_{i-1}(x_{i-1}, s_{i-1}\sigma_j) \neq \emptyset$ and $P_{\hat{\Sigma}_i}(s_{i-1}) = \epsilon$. Therefore let $\tilde{x}_{i-1} \in \xi_{i-1}(x_{i-1}, s_{i-1}) \cap \chi_{i-1}(\sigma_j)$. Consider string $r_i s_i \sigma_j \in \Sigma^*_i$ such that $\xi_i(x^0_i, r_i s_i \sigma_j) \neq \emptyset$ and $x_i \in \xi_i(x^0_i, r_i)$. By assumption

(3.6), G_{i-1} simulates G_i w.r.t. $\Sigma_{S_{i-1}}$. Therefore there exists string $t_{i-1} \in \Sigma_{i-1}^*$ such that $\xi_{i-1}(x_{i-1}^0, t_{i-1}\sigma_j) \neq \emptyset$ and

$$\begin{aligned} P_{\Sigma_{S_{i-1}}}(t_{i-1}) &= P_{\Sigma_{S_{i-1}}}(r_i s_i) \\ &= P_{\Sigma_{i-1}\cap\Sigma_i}(r_i s_i) \qquad\qquad (\because r_i s_i \in \Sigma_i^*). \end{aligned}$$

But $s_i \in (\Sigma_i \setminus \hat{\Sigma}_i)^*$ and $\Sigma_{i-1} \cap \Sigma_i \subseteq \hat{\Sigma}_i$; therefore

$$P_{\Sigma_{S_{i-1}}}(t_{i-1}) = P_{\Sigma_{i-1}\cap\Sigma_i}(r_i). \tag{3.8}$$

Hence $P_{\Sigma_{S_{i-1}}}(t_{i-1}) \subseteq (\Sigma_{i-1} \cap \Sigma_i)^*$; therefore $P_{\Sigma_{S_{i-1}}}(t_{i-1}) = P_{\Sigma_{i-1}\cap\Sigma_i}(t_{i-1})$ and by (3.8), $P_{\Sigma_{i-1}\cap\Sigma_i}(t_{i-1}) = P_{\Sigma_{i-1}\cap\Sigma_i}(r_i)$. Let $\hat{x}_{i-1} \in \xi_{i-1}(x_{i-1}^0, t_{i-1}) \cap \chi_i(\sigma_j)$. Then, by the structure of the synchronous product, we have $(\hat{x}_{i-1}, x_i) \in R_i$. On the other hand, by assumption (3.4), $\tilde{x}_{i-1}$ and $\hat{x}_{i-1}$ are one and the same; therefore $(\tilde{x}_{i-1}, x_i) \in R_i$.

$\square$

Lemma 1 showed that, if there exists a weak invariant simulation relation between G_{i-1} and G_i, then R_i is the smallest one. In contrast, while R_i need not be the largest weak invariant simulation between G_{i-1} and G_i, the above proposition states that, under our network assumptions, it is in a technically precise sense 'nearly that large': if x_{i-1} weakly invariantly simulates x_i w.r.t. $\hat{\Sigma}_i$, then x_{i-1} has a successor state reachable via events in $\Sigma_{i-1} \setminus \hat{\Sigma}_i$ that is related to x_i by R_i.

The following theorem expresses the most important property of our proposed ring network: if x_{i-1} does not weakly invariantly simulate x_i w.r.t. $\Sigma_{S_{i-1}}$, then there exists a path in the global model from x to another global state $\hat{x}$ such that $(\hat{x}_{i-1}, \hat{x}_i)$ belongs to the weak invariant simulation relation $\mathcal{W}_i$ w.r.t. $\Sigma_{S_{i-1}}$. This means that regardless of the evolution of the global network, the weak invariant simulation relation $\mathcal{W}_i$ can always be established between any two neighboring subprocesses in our proposed network model.

Theorem 2. Consider an instance G^N of $\mathcal{G}$ satisfying (3.3 - 3.6). For all $x \in X$ and all i, we have

$$(\exists s \in \Sigma^*)[(\xi(x,s) \neq \emptyset) \;\&\; (\exists \hat{x} \in \xi(x,s))(\hat{x}_{i-1}, \hat{x}_i) \in \mathcal{W}_i]. \tag{3.9}$$

Proof. For any string $l \in \Sigma^*$ and any i, let l_i denote $P_{\Sigma_i}(l)$. Fix $x \in X$. The proof is by induction on index i. According to Lemma 2, there exists m, $1 \leq m \leq N$ such that $(x_{m-1}, x_m) \in \mathcal{W}_m$. Therefore by setting $s = \epsilon$, we see that (3.9) holds for $i = m$. This forms the base case of the induction.

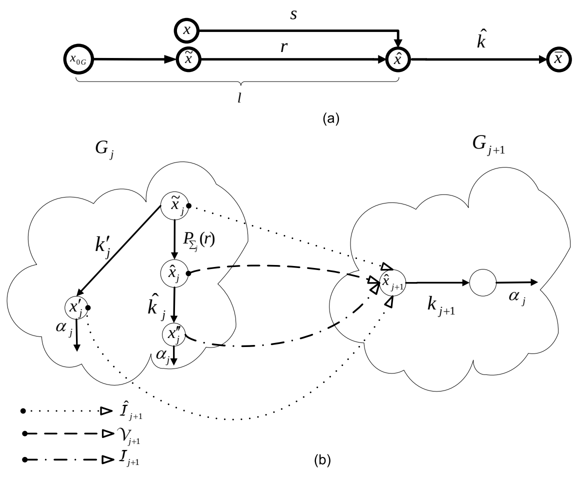

Figure 3.3: The illustration of relations between states in the induction step of the proof of Theorem 2: (a) Transitions with global strings l, $\tilde{l}$, $\hat{k}$, s and r. String r is not explicitly mentioned in the proof, but by the proof of the Claim, it does not contain any events shared between G_j and G_{j+1}. Existence of global string $\hat{k}$ is established in the last part of the induction. (b) Weak invariant simulation relations between states of G_j and G_{j+1}. Solid lines are transitions of subprocesses. String $P_{\Sigma_j}(r)$ is the projection of string r in part (a) of this figure. $\mathcal{V}_{j+1}$ and $\mathcal{W}_{j+1}$ are a weak invariant simulation w.r.t. $\Sigma_j \cap \Sigma_{j+1}$ and w.r.t. Σ_{S_j} respectively.

Now we assume that for some j, we have

$$(\exists s \in \Sigma^*)[(\xi(x,s) \neq \emptyset) \ \& \ (\exists \hat{x} \in \xi(x,s))(\hat{x}_{j-1}, \hat{x}_j) \in \mathcal{W}_j] \tag{3.10}$$

This forms the induction hypothesis.

Before the induction step, we prove a useful fact. According to the induction hypothesis, there exists a string $s \in \Sigma^*$ such that for some $\hat{x} \in \xi(x,s) \neq \emptyset$, $(\hat{x}_{j-1}, \hat{x}_j) \in \mathcal{W}_j$. We choose such an s and $\hat{x}$. Consider string $l \in \Sigma^*$ with $\hat{x} \in \xi(x^0, l)$; such a transition exists because the synchronous product G^N is by definition reachable. To facilitate the induction step, we invoke the following:

Claim. There exists a string $\tilde{l} \leq l$ such that

$$(\forall \tilde{x}_j \in \xi_j(x_j^0, \tilde{l}_j))((\tilde{x}_j, \hat{x}_{j+1}) \in \mathcal{W}_{j+1}) \tag{3.11}$$

Proof. By Corollary 4; see Appendix 3.B for details. □

To paraphrase, it is possible, by means of a prefix of l, for G_j to enter a state that is in a weak invariant simulation relation with $\hat{x}_{j+1}$.

It remains to complete the induction step. The weak invariant simulation relations established between states of neighboring subprocesses G_j and G_{j+1} in this part of the proof are displayed in Figure 3.3.

The case where for all $\alpha_j \in \Sigma_j \cap \Sigma_{j+1}$, $\chi_j(\alpha_j) = \emptyset$ or $\chi_{j+1}(\alpha_j) = \emptyset$ is trivial, for in this case, no shared event between G_j and G_{j+1} can occur - and indeed, by isomorphism, no shared event can occur anywhere in the network. Therefore we assume that there exists $\alpha_j \in \Sigma_j \cap \Sigma_{j+1}$ such that $\chi_j(\alpha_j) \neq \emptyset$ and $\chi_{j+1}(\alpha_j) \neq \emptyset$. Moreover, by assumption (3.3), the state transition graph of each component subprocess of the network is strongly connected. We accordingly fix $\alpha_j \in \Sigma_j \cap \Sigma_{j+1}$ such that there exists $k_{j+1} \in (\Sigma_{j+1} \setminus (\Sigma_j \cap \Sigma_{j+1}))^* = (\Sigma_{j+1} \setminus \Sigma_{S_j})^*$ with $\xi_{j+1}(\hat{x}_{j+1}, k_{j+1}\alpha_j) \neq \emptyset$. By the claim, we choose an $\tilde{l} \leq l$ and an $\tilde{x}_j \in \xi_j(x_j^0, \tilde{l}_j)$ such that $(\tilde{x}_j, \hat{x}_{j+1}) \in \mathcal{W}_{j+1}$. By the definition of weak invariant simulation, there exists $k_j' \in (\Sigma_j \setminus \Sigma_{S_j})^*$ such that

$$\xi_j(\tilde{x}_j, k_j'\alpha_j) \neq \emptyset, \text{ and} \tag{3.12}$$

$$(\forall y_j \in \xi_j(\tilde{x}_j, k_j'))(y_j, \hat{x}_{j+1}) \in \mathcal{W}_{j+1}. \tag{3.13}$$

According to (3.12) there exists $x_j' \in \xi_j(\tilde{x}_j, k_j') \cap \chi_j(\alpha_j)$. By (3.13),

$$(x_j', \hat{x}_{j+1}) \in \mathcal{W}_{j+1}. \tag{3.14}$$

At this point, it suffices to show that x_j' and $\hat{x}_{j+1}$ are components of some global state reachable from $\hat{x}$. For this, note that by Proposition 5, we have $(\hat{x}_j, \hat{x}_{j+1}) \in \mathcal{V}_{j+1}$. Since $\xi_{j+1}(\hat{x}_{j+1}, k_{j+1}\alpha_j) \neq \emptyset$, there must exist $\hat{k}_j \in \Sigma_j^*$ such that $(P_{\Sigma_j \cap \Sigma_{j+1}}(\hat{k}_j) = \epsilon)$ & $\xi_j(\hat{x}_j, \hat{k}_j\alpha_j) \neq \emptyset$. Fix such a string $\hat{k}_j$ and a state $x_j'' \in \xi_j(\hat{x}_j, \hat{k}_j) \cap \chi_j(\alpha_j)$. By assumption (3.4), x_j' and x_j'' are the same state. Now $\hat{k}_j \in \Sigma_j^*$ contains only local events and events in $\Sigma_{j-1} \cap \Sigma_j$. By the inductive hypothesis, we have $(\hat{x}_{j-1}, \hat{x}_j) \in \mathcal{W}_j$. Because $\mathcal{W}_j$ is a weak simulation with respect to $\Sigma_{S_{j-1}}$, there exists a $\hat{k}_{j-1} \in \Sigma_{j-1}^*$ with the same $\Sigma_{S_{j-1}}$ projection as $\hat{k}_j$, and therefore containing only local events and events in $\Sigma_{j-1} \cap \Sigma_j$. There must therefore exist a global string $\hat{k} \in L((G^N)^{\rightarrow \hat{x}})$ such that $P_{\Sigma_{j-1}}(\hat{k}) = \hat{k}_{j-1}$ and $P_{\Sigma_j}(\hat{k}) = \hat{k}_j$ and, for all $i \notin \{j-1, j\}$, $P_{\Sigma_i}(\hat{k}) = \emptyset$. It follows that there exists $\bar{x} \in X$ such that $\bar{x} \in \xi(x, s\hat{k})$ and

$$\begin{aligned} \bar{x}_j &= x_j' \in \xi_j(\hat{x}_j, \hat{k}_j), \text{ and} \\ \bar{x}_{j+1} &= \hat{x}_{j+1} \qquad\qquad (\because P_{\Sigma_{j+1}}(\hat{k}) = \epsilon). \end{aligned}$$

Therefore by (3.14), $(\bar{x}_j, \bar{x}_{j+1}) \in \mathcal{W}_{j+1}$. This completes the induction.

In the next subsection, we apply the above fundamental network properties to the analysis of deadlock. □

3.3 The Deadlock Analysis Procedure

The next definition aims to characterize occurrence of a circular wait in our ring network. It will be used in the next theorem to determine all the deadlocked states of the proposed network.

Definition 7. Consider an instance G^N of $\mathcal{G}$ satisfying (3.3 - 3.6). Fix i, $1 \leq i \leq N$, and let $\hat{\Sigma}_i = \Sigma_{S_{i-1}}$. Consider the synchronous product $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i = (R_i, \Sigma_{i-1} \cup \Sigma_i, \tilde{\xi}_i, x^0_{i-1} \times x^0_i, X_{m_{i-1}} \times X_{m_i})$, where $\Delta_{i-1} = \Sigma_{i-1} \setminus (\hat{\Sigma}_i \setminus \Sigma_i) = \Sigma_{i-1} \setminus \Sigma_{i-2}$. We define, for any i, $R_{d_i} \subseteq R_i$ to be the subset of state pairs that satisfy the following property:

$$(\forall \sigma_i \in \Sigma_{i-1} \cup \Sigma_i)[(\tilde{\xi}_i((x_{i-1}, x_i), \sigma_i) \neq \emptyset) \Rightarrow (\sigma_i \in \Sigma_{i+1})]. \tag{3.15}$$

Property (3.15) of state pair subset R_{d_i} expresses that whenever G^N is in a state $x \in X$ such that for all i, $(x_{i-1}, x_i) \in R_{d_i}$, all the subprocesses are waiting for execution of a shared event in their respective immediate neighbors with 'larger' index. These dependencies in the direction of increasing indices result in a circular wait, and consequently deadlock.

The next theorem implies that checking deadlock-freedom in the proposed network model is decidable. Furthermore it enables us to find the deadlocked states.

Theorem 3. Consider an instance G^N of $\mathcal{G}$ satisfying (3.3 - 3.6). Let $\hat{\Sigma}_i = \Sigma_{S_{i-1}}$, $\Delta_{i-1} = \Sigma_{i-1} \setminus \Sigma_{i-2}$ and $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i = (R_i, \Sigma_{i-1} \cup \Sigma_i, \tilde{\xi}_i, x^0_{i-1} \times x^0_i, X_{m_{i-1}} \times X_{m_i})$. Furthermore let $R_{d_i} \subseteq R_i$ be the set of state pairs that satisfy (3.15). The instance G^N of PDES $\mathcal{G}$ contains a (reachable) deadlocked state if and only if there exists state $x \in X_1 \times X_2 \times \ldots \times X_N$ such that

$$(\forall i)((x_{i-1}, x_i) \in R_{d_i}). \tag{3.16}$$

Proof. (*If*) We will first show that $x \in X_1 \times X_2 \times \ldots \times X_N$ is reachable, and therefore a state of G^N. According to (3.16), for all i, $1 \leq i \leq N$, $(x_{i-1}, x_i) \in R_{di}$. But $R_{di} \subseteq R_i$, therefore

$$(x_{i-1}, x_i) \in R_i, \tag{3.17}$$

and R_i is a weak invariant simulation w.r.t $\Sigma_{S_{i-1}}$, according to Lemma 1(c) with $\hat{\Sigma}_i = \Sigma_{S_{i-1}}$. By assumption (3.3), for some $\sigma_{i-1} \in \Sigma_{i-1} \cap \Sigma_i$, there exists a string $s_i \in (\Sigma_i \setminus \Sigma_{i-1})^*$ and an $x_i' \in X_i$ such that $x_i' \in \xi_i(x_i, s_i) \cap \chi_i(\sigma_{i-1})$ (by the non-emptiness of R_{di}). Hence by the weak invariant simulation w.r.t. $\Sigma_{S_{i-1}}$ in (3.17), a state in $\chi_{i-1}(\sigma_{i-1})$ must be reachable from x_{i-1} by executing only local events. But by the definition of R_{di}, no local event is defined from x_{i-1}; therefore $x_{i-1} \in \chi_{i-1}(\sigma_{i-1})$.

On the other hand, since condition (3.16) holds for all i, the state pair (x_i, x_{i+1}) is reachable in $G_i \upharpoonright \Delta_i \| G_{i+1}$. Accordingly, owing to the restriction of G_i to $\Delta_i = \Sigma_i \setminus \Sigma_{i-1}$, x_i is reachable from x_i^0 (the initial state of G_i) by a string r_i that contains no event shared with G_{i-1}. By (3.6), $(x_{i-1}^0, x_i^0) \in \mathcal{W}_i$, where $\mathcal{W}_i$ is a weak invariant simulation w.r.t. $\Sigma_{S_{i-1}}$. By existence of string r_i and the definition of weak invariant simulation, we have

$$(x_{i-1}^0, x_i) \in \mathcal{W}_i. \tag{3.18}$$

But as stated earlier, there exists a path $s_i \in (\Sigma_i \setminus \Sigma_{i-1})^*$ such that $x_i' \in \xi_i(x_i, s_i) \cap \chi_i(\sigma_{i-1})$. Therefore by (3.18) there exists a path of local events from x_{i-1}^0 to a state $\hat{x}_{i-1} \in \chi_{i-1}(\sigma_{i-1})$. Since $x_{i-1}, \hat{x}_{i-1} \in \chi_{i-1}(\sigma_{i-1})$, by (3.4) we conclude that x_{i-1} and $\hat{x}_{i-1}$ are one and the same. Because x_{i-1} is reachable from the initial state of G_{i-1} by local events, we can use the same argument for reachability of such states in all other subprocesses, and conclude that x is a reachable global state in G^N; that is, $x \in X$.

Now suppose G^N contains no deadlocked state, but (3.16) is satisfied. By the absence of deadlock, there must exist some event for which a transition is defined in the global state x. By assumption, such an event must be a shared event $\alpha_j \in \Sigma_S$. Let i be such that $\alpha_j \in \Sigma_{i-1} \cap \Sigma_i \subseteq \Sigma_{S_{i-1}}$. Given that $\xi(x, \alpha_j) \neq \emptyset$, by (3.16), $\tilde{\xi}_i((x_{i-1}, x_i), \alpha_j) \neq \emptyset$. But this contradicts (3.15).

(*Only if*) Consider an arbitrary deadlocked state $x \in X$ of G^N. We will show that this state satisfies property (3.16). For this, we shall apply Proposition 6; but that requires us first to establish that, in a deadlocked state such as x, for all i, (x_{i-1}, x_i) belongs to a weak invariant simulation with respect to $\hat{\Sigma}_i = \Sigma_{S_{i-1}}$. According to Theorem 2, for any i, if x_{i-1} is not in a weak invariant simulation relation with x_i w.r.t. $\Sigma_{S_{i-1}}$, then there exists a string $s \in \Sigma^+$ such that for some $\hat{x} \in \xi(x, s)$, $(\hat{x}_{i-1}, \hat{x}_i) \in \mathcal{W}_i$. By assumption $x \in X$ is a deadlocked state; therefore, for every i, we must have $(x_{i-1}, x_i) \in \mathcal{W}_i$.

We are now in a position to apply Proposition 6, for any i, with $\hat{\Sigma}_i = \Sigma_{S_{i-1}}$. Accordingly, there exists a string s_{i-1} of local events of G_{i-1} and an $\tilde{x}_{i-1} \in \xi_{i-1}(x_{i-1}, s_{i-1})$ such that $(\tilde{x}_{i-1}, x_i) \in R_i$. But since x is a deadlocked state, this string of local events can only be the empty string; hence $(x_{i-1}, x_i) \in R_i$.

Now suppose $(x_{i-1}, x_i) \notin R_{di}$; i.e., for some $\sigma_j \in (\Sigma_{i-1} \cup \Sigma_i) \setminus \Sigma_{i+1}$ the transition $\tilde{\xi}_i((x_{i-1}, x_i), \sigma_j)$ is defined. But σ_j is not shared with G_{i-2} (owing to the restriction $G_{i-1} \upharpoonright \Delta_{i-1}$) or G_{i+1}. Therefore $\xi(x, \sigma_j) \neq \emptyset$. This contradicts the assumption that state x is deadlocked. Hence $(x_{i-1}, x_i) \in R_{d_i}$. This means that property (3.16) holds for the deadlocked state x. □

The above theorem reduces deadlock analysis of the global system to polynomial-time analysis of the synchronous product $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$. This analysis yields the set of state pairs R_{di}; the state pairs in turn determine whether it is possible for a given PDES instance to deadlock.

Remark 5. Indeed, by isomorphism, the relation R_{di} is the same for all i. According to the above theorem, to check for potential deadlocks in our subclass of parameterized ring networks, it suffices to check for cycles of states of individual subprocesses such that successive pairs in a given cycle are related by R_{di}. Given the synchronous product $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$, the number of computational steps required to compute R_{di} and to check for such cycles is $O(|X_i|^2 |\Sigma_i|)$.

For instance, consider $R_{di} = \{(x_a, x_b), (x_b, x_c)\}$. Then it is impossible to satisfy (3.16) for all i, because no cycle made up of states x_a, x_b and x_c has the property that all pairs of successive states are related by R_{di}. However, if the pair (x_c, x_a) also belongs to R_{di}, then x_a, x_b and x_c form a cycle of length 3 such that all pairs of successive states in the cycle (namely $(x_a, x_b), (x_b, x_c)$ and (x_c, x_a)) belong to R_{di}. Therefore, condition (3.16) can be satisfied for all i provided N is a multiple of 3. This example also shows that for some parameterized ring networks, the network size plays a critical role in deadlock susceptibility. For these PDES, Theorem 3 also provides information about specific size instances of a PDES that are deadlock-prone.

3.4 Illustrative example: parameterized dining philosophers

Dining philosophers is a classic multi-process example featuring isomorphic subprocesses. Moreover, it is easily understood intuitively. Here we illustrate our methods by considering the parameterized version: the problem of dining philosophers for an arbitrary number of philosophers. This problem can be modeled using a PDES. The parameter of interest is the number of philosophers and the problem corresponding to a specific number of philosophers is a particular instance of the PDES. Our goal is to compute accessible deadlocked states

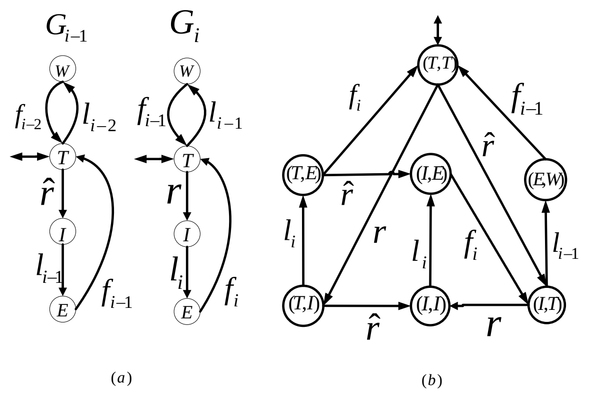

Figure 3.4: (a) The model of two neighbor philosophers in the dining philosopher example. (b) The structure of $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$, as described in Lemma 1, for the dining philosophers example.

of all instances of the PDES; for a specific instance, the possibility of deadlock can be checked using known methods [25].

Assume that $N > 2$ philosophers are seated around a dining table. There is only one fork between any two adjacent philosophers. Our model of individual philosophers is similar to that of [21]. In order to eat, a philosopher must hold both adjacent forks. Each philosopher is initially in a Thinking state. As a philosopher gets hungry, he first picks up the fork on his right-hand side, and enters an Idle state. Then he picks up the fork on his left and enters his Eating state. He relinquishes both forks when he has finished eating and returns to the Thinking state. The Thinking state is the sole marking state. The question is whether, whenever a philosopher picks up his right-hand side fork, he will be able eventually to eat and to return to the Thinking state. A model of the i^{th} and $i-1^{th}$, $1 \leq i \leq N$, philosophers are depicted in Figure 3.4(a) (philosophers face inward and are numbered clockwise from above). States T are E are Thinking and Eating states. State W represents the case where a philosopher is thinking and waits for his right-hand side fork to become available. State I is the Idle state. In the model of the i^{th} philosopher, events r and l_i respectively mean that philosopher i takes his right-hand side and left-hand side fork (the left and right directions are from the viewpoint of the philosopher). Event f_i means philosopher i finishes eating and returns both forks. Consistently with the notation of this chapter, in subprocess G_i events with subscript i (namely, l_i and f_i) are shared between G_i and G_{i+1}, events with subscript $i-1$ (l_{i-1} and f_{i-1}) between G_{i-1} and G_i. Events $\hat{r}$

and r are local events; however they nevertheless model interaction between subprocesses, by preempting, or being preempted by, shared events: For instance, consider the shared resource (fork) between G_{i-1} and G_i. When subprocess G_i executes event r (philosopher i takes his right-hand side fork), it transitions from state T (thinking state) to state I (idle state). Note that shared event l_{i-1} which represents taking the left-hand side fork by philosopher $i-1$, is available from state T of G_i, but unavailable from state I of G_i. Therefore as long as G_i is in state I (philosopher i is idle and holds his right-hand fork), shared event l_{i-1} cannot be executed by G_{i-1} (philosopher $i-1$ cannot take his left-hand side fork). Similarly, if philosopher $i-1$ takes his left-hand side fork (executes event l_{i-1}), philosopher i moves to state W from state T, where he cannot pick up the same fork (cannot execute event r).

It is easy to check that G^N with G_i presented in Figure 3.4(a) complies with conditions (3.3 - 3.6). Therefore, according to Theorem 3, we can locate the deadlocked states in this problem by constructing the synchronous product $G_{i-1} \upharpoonright \Delta_{i-1} \| G_i$ depicted in Figure 3.4(b). As can be seen in the figure, the set R_{di} of Definition 7 consists of pairs (I, I) and (I, E). A given state $x \in X$ is deadlocked if and only if (3.16) is satisfied for all i. The only possible setting in which each pair of neighboring subprocesses is in (I, I) or (I, E) is that in which all subprocesses are in state I (see Remark 5). This result can easily be confirmed with intuitive reasoning.

3.5 Conclusion

In this chapter, we used the mathematical tool of 'weak invariant simulation' to define a subclass of parameterized networks in which the deadlock-freedom property is decidable. On the basis of the findings of Chapter 2, a method for deadlock analysis of a class of parameterized ring networks was proposed. The analysis is based on a pair of subprocesses and exploits the symmetry of a parameterized network in order to reason about global system properties. The present chapter highlights application of weak invariant simulation to deadlock analysis of parameterized networks with the topology of a ring; but most nontrivial deadlocks in more general topologies arise as a result of a 'circular wait' or a 'deadly embrace' within a subnetwork having the topology of a closed circuit. In the next chapter we extend the deadlock analysis to a class of parameterized networks with more general topologies.

Appendix

3.A Proof of Lemma 2

For initial state x^0, the proof is immediate by (3.6). For an arbitrary state $x \in X$, for some string $s \in \Sigma^*$, we have $x \in \xi(x^0, s)$. If string s has no shared event, the desired property holds for any i, according to Proposition 4 with $\hat{\Sigma}_i = \Sigma_{S_{i-1}}$. In case the string s contains any shared event, let i be such that $\sigma_{i-1} \in \Sigma_{i-1} \cap \Sigma_i$ is the last shared event symbol in s and let $r\sigma_{i-1}$ be the longest prefix of s ending in σ_{i-1}, $r_i = P_{\Sigma_i}(r)$ and $r_{i-1} = P_{\Sigma_{i-1}}(r)$.

Let $\bar{x}_i \in \xi_i(x_i^0, r_i)$ and $l_i \in \Sigma_i^*$ be such that $x_i \in \xi_i(\bar{x}_i, \sigma_{i-1}l_i)$ and $P_{\Sigma_{S_{i-1}}}(l_i) = \emptyset$. Similarly, let $\bar{x}_{i-1} \in \xi_{i-1}(x_{i-1}^0, r_{i-1})$ such that $x_{i-1} \in \xi_{i-1}(\bar{x}_{i-1}, \sigma_{i-1}l_{i-1})$ where $l_{i-1} \in \Sigma_{L_{i-1}}^*$. We shall show that $(\bar{x}_{i-1}, \bar{x}_i) \in \mathcal{W}_i$, which will lead to the result. For this, note that because $x_i \in \xi_i(x_i^0, s_i)$, where $s_i = P_{\Sigma_i}(s)$, by (3.6) and the definition of weak invariant simulation there exists $\hat{s}_{i-1} \in L(G_{i-1})$ such that

$$P_{\Sigma_{S_{i-1}}}(\hat{s}_{i-1}) = P_{\Sigma_{S_{i-1}}}(s_i).$$

Let $\hat{r}_{i-1} \in \Sigma_{i-1}^*$ be such that $\hat{r}_{i-1}\sigma_{i-1}$ is the longest prefix of $\hat{s}_{i-1}$ ending in σ_{i-1}. Then $P_{\Sigma_{S_{i-1}}}(\hat{r}_{i-1}\sigma_{i-1}) = P_{\Sigma_{S_{i-1}}}(r_i\sigma_{i-1})$ and therefore $P_{\Sigma_{S_{i-1}}}(\hat{r}_{i-1}) = P_{\Sigma_{S_{i-1}}}(r_i)$.

Let $\hat{x}_{i-1} \in \xi_{i-1}(x_{i-1}^0, \hat{r}_{i-1})$ be such that $\xi_{i-1}(\hat{x}_{i-1}, \sigma_{i-1}) \neq \emptyset$. Then by definition of weak invariant simulation $(\hat{x}_{i-1}, \bar{x}_i) \in \mathcal{W}_i$. Since $\bar{x}_{i-1}$ and $\hat{x}_{i-1}$ are in $\chi_{i-1}(\sigma_{i-1})$, by assumption (3.4), they are one and the same. We therefore have $(\bar{x}_{i-1}, \bar{x}_i) \in \mathcal{W}_i$; but this implies that $(x_{i-1}, x_i) \in \mathcal{W}_i$.

3.B Proof of Claim

We consider two cases. First, if $P_{\Sigma_j \cap \Sigma_{j+1}}(l) = \epsilon$, we have $P_{\Sigma_{S_j}}(P_{\Sigma_{S_{j+1}}}(l)) = \epsilon$ and therefore $P_{\Sigma_{S_j}}(l_{j+1}) = \epsilon$. Let $\tilde{l} = \epsilon$. By assumption (3.6) of the network and the invariance of $\mathcal{W}_{j+1}$, we have $(x_j^0, \hat{x}_{j+1}) \in \mathcal{W}_{j+1}$.

Second, if $P_{\Sigma_j \cap \Sigma_{j+1}}(l) \neq \epsilon$, assume that σ_j is the last event of l that belongs to the set $\Sigma_j \cap \Sigma_{j+1}$. Consider string $\tilde{l} = t\sigma_j \leq l$ of maximal length. Let $\tilde{x} \in \xi(x^0, \tilde{l})$. By Corollary 4, we have $(\tilde{x}_j, \tilde{x}_{j+1}) \in \mathcal{W}_{j+1}$.

Therefore by the invariance property and the fact that σ_j is the last event of l_{j+1} that belongs to the set Σ_{S_j}, we conclude that $(\tilde{x}_j, \hat{x}_{j+1}) \in \mathcal{W}_{j+1}$.

Hence in both cases, (3.11) holds. This proves the claim.

Chapter 4

Analysis of generalized PDES

In this chapter, we consider networks consisting of multiple parameterized sections, and more general topologies. To model these networks, we introduce Generalized Parameterized Discrete Event Systems (GPDES). A GPDES consist of fixed number of 'linear 'parameterized sections as well as fixed number of 'distinguished' subprocesses. Distinguished subprocesses may have an arbitrary structure, which is dissimilar to that of any other subprocess. Each linear parameterized section contains several isomorphic subprocesses: each subprocess in a linear parameterized section of the network is obtained by relabeling of a template. The number of subprocesses in each parameterized section is arbitrary. In our setting, subsystems interact with each other via execution of shared events. A shared event between two subprocesses can occur if both subprocesses are in states that allow the shared event. We assume that each event is shared between at most two subprocesses.

A GPDES can be represented by a directed graph, where arcs correspond to linear parameterized sections and nodes are the distinguished subprocesses. The graph representation of a GPDES may contain several cycles. We characterize the dependencies in each cycle of the network: for an arbitrary cycle, we first disable the events that are shared with subprocesses outside of the cycle to get a ring network. In the resulting ring network, we calculate state pairs with the *forward dependency* property. After carrying out this calculation for all cycles of the network, we form the *dependency graph.* We show that the specific subgraphs of the dependency graph represent reachable *local deadlocks* in our proposed GPDES network. The proposed deadlock analysis procedure has polynomial-time complexity. We illustrate our proposed method by deadlock analysis of a large-scale factory.

4.1 The network model

4.1.1 Linear parameterized discrete event systems

Recall from Chapter 3 that a PDES $\mathcal{P}$ is an infinite set of synchronous products of M isomorphic finite-state subprocesses, where M ranges over the set of natural numbers $\mathbb{N}$. Formally,

$$\mathcal{P} = \{\|_{i=1}^{M} P_i : \; M \in \mathbb{N}\}, \tag{4.1}$$

where

$$P_i = (X_i, \Sigma_i, \xi_i, x_i^0, X_{mi}), \tag{4.2}$$

with $X_1 = X_2 = ...$, and M is the unspecified parameter. Subprocesses P_i, $1 < i < M$, are formed by appropriate relabellings of a template.

We are particularly interested in PDES with *linear* topology. PDES $\mathcal{P}$ has linear topology if for any member $\|_{i=1}^{M} P_i \in \mathcal{P}$, subprocesses P_i, $1 < i < M$ has shared transitions with both P_{i-1} and P_{i+1}, but there is no event shared between P_1 and P_M. We assume that shared events are shared between at most two subprocesses. For subprocess P_i of a linear PDES, we assume that symbols in Σ_{L_i} (local events) have no index, and shared event symbols in set Σ_{S_i} either have index $i-1$ or index i: symbols in $\Sigma_{i-1} \cap \Sigma_i$ (shared events between generators P_{i-1} and P_i) have index $i-1$, while the symbols in $\Sigma_i \cap \Sigma_{i+1}$ (shared events between P_i and P_{i+1}) have index i.

For a linear PDES with M subprocesses, we set P_1 as the template, and define permutation κ to form the rest of the subprocesses:

$$(\forall \alpha \in \Sigma_i)(\forall x \in X_s)(\xi_i(x, \alpha) = \xi_{i+1}(x, \kappa(\alpha))).$$

where X_s is the common state set of all the subprocesses of $\mathcal{P}$. Permutation $\kappa : \Sigma \to \Sigma$ is a bijection of order M such that for all i,

$$(\forall \sigma \in \Sigma_{L_i})(\kappa(\sigma) = \sigma);$$
$$(\forall \sigma_i \in \Sigma_i \cap \Sigma_{i+1})(\kappa(\sigma_i) = \sigma_{i+1});$$

An event shared between two subprocesses P_{i-1} and P_i, $1 < i \leq M$, can occur only if both subprocesses are in states that allow the shared event. If for instance P_i is in a state that can reach marking states only by executing a shared event which cannot be executed by P_{i-1}, then subprocess P_i is blocked. The following definition helps us to address this issue.

Definition 8. [60] Consider a network $\|_{i=1}^{M} P_i$, with $P_i = (X_i, \Sigma_i, \xi_i, x_i^0, X_{mi})$, $M \in \mathbb{N}$. For a shared event σ_i of subprocess P_i , $1 \leq i \leq M$, *companion states* of σ_i in subprocess P_j are states x_j, for which $\xi_j(x_j, \sigma_i) \neq \emptyset$. The set of such companion states is $\chi_j(\sigma_i)$. The notion of companion states can be extended to subalphabets: in the j^{th} subprocess, companion states of a given subalphabet $\hat{\Sigma}$ are states x_j in P_j, for which for some $\sigma_i \in \hat{\Sigma}$, $\xi_j(x_j, \sigma_i) \neq \emptyset$.

4.1.2 Generalized parameterized discrete event systems

A GPDES consists of a fixed number of ‘distinguished’ subprocesses and fixed number of linear parameterized sections. A distinguished subprocess can have a structure distinct from those of other subprocesses. Each *linear* parameterized section consist of an arbitrary number of isomorphic subprocesses uniformly indexed from 1 to parameter of the linear parameterized section. We assume that each event symbol in the alphabet of the GPDES is at most shared between two subprocesses. We refer to subprocesses that have shared events between them as neighboring subprocesses.

In order to characterize the network topology, we use a strongly connected directed graph: each vertex in the directed graph represents a distinguished subprocess and each arc of the graph represents a linear parameterized section. The direction of each arc in the graph representation is that of increasing indices of the linear parameterized section they represent.

In the graph representation of the network, we name the vertices with multiple incoming arcs *input vertices*, and the nodes with multiple outgoing arcs *output vertices*. In this chapter, we consider networks with one input vertex and multiple output vertices; and assume that the input vertex is not also an output vertex. The subprocesses correspond to input vertex and output vertices are called input subprocess and output subprocess respectively. As an example of this branching topology, see the graph representation of large-scale factory in Figure 4.4.(a) of Section 3.4.

Checking the deadlock freedom property in the general network is undecidable [32]. Thus, in order to be able to carry out our analysis, we set assumptions on all cycles of the network. Consider an instance of the GPDES network described above. Within this instance, consider an arbitrary cycle. For simplicity, we relabel subprocesses of this cycle from G_1 to G_N starting with the input vertex (note that any cycle must include the unique input vertex.) We denote such cycle

$$G^N = \big\|_{i=1}^{N} G_i = (X, \Sigma, \xi, x^0, X_m), \tag{4.3}$$

with $G_i = (X_i, \Sigma_i, \xi_i, x_i^0, X_{mi})$, $1 \leq i \leq N$. From this point on in the chapter, when we refer to cycles, terms $i+j$ and $i-j$ are calculated using *modulo-N* arithmetic over the complete residue system $\{1, 2, ..., N\}$. This is the same as standard modulo-N arithmetic, except that the equivalence class of the integers modulo N that contains zero is represented by N instead of zero. Thus in cycle G^N, $\Sigma_i \cap \Sigma_j \neq \emptyset$ only if $i - j = 1$ *or* $i - j = N - 1$. We set $\Sigma_i = \Sigma_{L_i} \cup \Sigma_{S_i}$, were Σ_{S_i} is the set of shared events of G_i and Σ_{L_i} is the set of local (unshared) events of G_i.

For cycle G^N, for all i, we assume the following:

$$(\forall i)(\forall x_i \in X_i)(\forall x_i' \in X_i)(\exists t \in \Sigma_i^*)[x_i' \in \xi_i(x_i, t)], \tag{4.4}$$
$$(\forall i)(\forall \sigma_i \in \Sigma_i \cap \Sigma_{i+1})(|\chi_i(\sigma_i)| \leq 1), \tag{4.5}$$
$$(\forall i)(G_i \mathcal{V}_{i+1} G_{i+1}), \tag{4.6}$$

where $\mathcal{V}_{i+1}$ is a weak invariant simulation of G_{i+1} by G_i w.r.t. $\Sigma_i \cap \Sigma_{i+1}$.

In Chapter 3, we proposed a framework for deadlock analysis of parameterized ring networks that are subject to four assumptions. Three of those assumptions are the same as conditions (4.4) to (4.6). The fourth – and most restrictive – assumption was removed here, which allows modeling of more complex parameterized networks.

Assumptions (4.4) and (4.5) are conditions on the structure of an individual subprocess; assumption (4.6) restricts the way subprocesses interact. Assumption (4.4) states that the transition graph of each subprocess of the network is strongly connected. By (4.5), each shared event in the subset $\Sigma_{i-1} \cap \Sigma_i$ has at most one companion state in subprocess G_{i-1}. In other words, interactions between G_{i-1} with G_i via a specific shared event in $\Sigma_{i-1} \cap \Sigma_i$ can only occur if G_{i-1} is in that specific state. Assumption (4.6) states that G_{i-1} weakly invariantly simulates G_i with respect to $\Sigma_{i-1} \cap \Sigma_i$. from all the reachable states of the network, if interaction between G_{i-1} and the rest of the network is ignored, subprocess G_{i-1} can always reach companion states of events shared between G_{i-1} and G_i. If this condition is not satisfied, then G_{i-1} may never be able to provide resources needed by G_i, which may indicate a design flaw in the network structure. All three assumptions on the linear parameterized sections of the network are arguably mild. For more discussion of these three conditions see Chapter 3.

To make the analysis tractable we restrict the structure of the distinguished subprocesses. For the unique input vertex G_1, we assume

$$(\forall \alpha \in \Sigma_N \cap \Sigma_1)(\forall \beta \in \Sigma_1 \cap \Sigma_2)[(\chi_1(\alpha)) \cap (\chi_1(\beta)) = \emptyset], \tag{4.7}$$

$$(\forall x \in R_2)(x_1 \mathcal{W}_2 x_2), \tag{4.8}$$

where R_2 is the state set of synchronous product $G_1 \| G_2$ and $\mathcal{W}_2$ is a weak simulation w.r.t. Σ_{S_1}. Assumption (4.7) expresses that if there is an event defined from any state of G_1 that is shared with G_2 then there is no event shared with G_N defined from that state. If a subprocess does not satisfy this assumption, we can simply move one of the transition with shared events to a new state and a add a local transition from the original state to the new one. Assumption (4.8), expresses that all the state pairs in synchronous product of G_1 and G_2 are in relation $\mathcal{W}_2$. This means that from any reachable global state, if G_2 is in a state in which a shared event with G_1 is defined, G_1 can always reach the companion state of that shared event without executing any shared event. In other words, G_1 can always provide resources requested by G_2. This is a relatively strong assumption, however according to proposed graph representation of network, there is only one input node in the network. Therefore this restriction is only applied to a single subprocess. This assumption allows us to relax other assumptions on the parameterized sections of the network.

Let J be the index set of output vertices of G^N in the network graph. For any output subprocess G_j, $j \in J$, we have

$$(G_j \mathcal{Q}_{j+1} G_{j+1}), \tag{4.9}$$

where $\mathcal{Q}_{j+1}$ is a weak invariant simulation w.r.t. $\Sigma_{S_j} \setminus \Sigma_{j-1}$.

Assumption (4.9) expresses that as long as G_j executes no event shared with subprocesses outside of G^N, then execution of events shared between G_j and G_{j+1} cannot be blocked by subprocesses outside of G^N. Indeed, subprocess G_j may execute a shared event with the rest of the network, after which the simulation relation need not hold. In other words, G_j can always provide resources for G_{j+1} until a resource is requested from G_j by subprocesses outside of cycle G^N. This interpretation arguably shows the mildness of this assumption.

For each parameterized section of the network, properties (4.4) and (4.5) can be verified by examining a single subprocess, then by isomorphism the property extends to all subprocesses of the parameterized section. Similarly, (4.6) needs to be checked for only two neighboring subprocesses for each parameterized section using Corollary 1 of Chapter 3. For the distinguished subprocesses, checking (4.8) and (4.9) respectively can also be done by the procedure in Corollary 1 of Chapter 3. Checking (4.7) is trivial.

One of the important properties of $\mathcal{V}_i$ in assumption (4.6) of our proposed network model is that it is preserved throughout the evolution of the network:

Proposition 7. Let G^N be the cycle defined in (4.3). Then for all $(x_{i-1}, x_i) \in R_i$, where R_i is the state set of synchronous product $G_{i-1} \| G_i$,

$$(x_{i-1}, x_i) \in \mathcal{V}_i. \tag{4.10}$$

Proof. By Proposition 1 of Chapter 3: specifically, by assumption, $(x_{i-1}^0, x_i^0) \in \mathcal{V}_i$, where $\mathcal{V}_i$ is a weak invariant simulation w.r.t. $\Sigma_{i-1} \cap \Sigma_i$. Since $(\Sigma_{i-1} \cap \Sigma_i) \subseteq \Sigma_i$, the result follows from the definition of weak invariant simulation. □

4.2 The deadlock analysis

In order to locate circular waits among the subprocesses, we initially focus on the individual cycles in the network graph. For that analysis, we disable transitions of certain distinguished subprocesses in a cycle of the network graph to yield a subgraph with ring structure. The next function will be used to restrict the transitions of subprocesses.

Definition 9. For a given generator G_i, $G_i(\xrightarrow{\Delta_i})$ is the restriction of the generator to a transition function $\hat{\xi} : X_i \times \Delta_i \to X_i$ and is formed by erasing transitions with events that belong to the set $\Sigma_i \setminus \Delta_i$, and unreachable states. Formally, $G_i(\xrightarrow{\Delta_i}) = (\hat{X}_i, \Sigma_i, \hat{\xi}_i, x_i^0, X_{m_i})$ and

$$\hat{\xi}_i(x_i, \sigma) = \begin{cases} \xi_i(x_i, \sigma), & \text{if } \sigma \in \Delta_i; \\ \emptyset, & \text{if } \sigma \in \Sigma_i \setminus \Delta_i. \end{cases},$$

and $\hat{X}_i \subseteq X_i$ such that for all $\hat{x}_i \in \hat{X}_i$, $\exists l \in \Sigma_i^*$ such that $\hat{x}_i \in \hat{\xi}_i(x_i^0, l)$.

Note that the above operation does not alter the alphabet set of G_i; it prevents the occurrence of any events in $\Sigma_i \setminus \Delta_i$ by altering the transfer function of the generator.

Next, we define an *isolated cycle* in the GPDES network. For any cycle in the network, the isolated version is calculated by disabling shared events of distinguished subprocesses with the rest of the network:

Definition 10. Let G^N be the cycle defined in (4.3). The *isolated cycle* $\hat{G}^N = (\hat{X}, \Sigma, \hat{\xi}, x^0, X_m)$ is the ring network formed by restriction of all subprocesses of G^N to transitions that are not shared with subprocesses outside of G^N. Formally, let $\Delta_i = (\Sigma_{i-1} \cap \Sigma_i \cap \Sigma_{i+1}) \cup \Sigma_{L_i}$. Then $\hat{G}^N = \|_{i=1}^{N} (G_i(\xrightarrow{\Delta_i}))$.

Note that in G^N, the only subprocesses that have events shared with subprocesses outside of G^N are G_1 and G_j, $j \in J$, where J is the index set of output vertices of G^N. Therefore, in order to achieve the isolated cycle $\hat{G}^N$, we only need to restrict these subprocesses.

We imposed restrictions on input and output vertices of each cycle by (4.7-4.9). The next proposition expresses two properties of vertices of an isolated cycle that are input and output vertices in the original network graph. Part (a) of the proposition expresses that whenever states of two neighbors in the original network graph are in weak invariant simulation relation of assumption (4.9), then these states belong to another weak invariant simulation relation in the isolated cycle; however, part (b) indicates that assumption (4.8) is essentially preserved for the restricted subprocesses of the isolated cycle.

Proposition 8. Consider cycle G^N defined in (4.3), and isolated cycle $\hat{G}^N$.

(a) Let $j \in J$, where J is the index set of output vertices. Assume $(x_j, x_{j+1}) \in \mathcal{Q}_{j+1}$, where $\mathcal{Q}_{j+1}$ is the weak invariant simulation of G_{j+1} by G_j w.r.t. $\Sigma_j \setminus \Sigma_{j-1}$. We then have $(x_j, x_{j+1}) \in \hat{\mathcal{V}}_{j+1}$, where $\hat{\mathcal{V}}_{j+1}$ is the weak invariant simulation of $\hat{G}_{j+1}$ by $\hat{G}_j$ w.r.t. $\Sigma_{G_i} \cap \Sigma_{G_{i+1}}$.

(b) Let $\hat{R}_2$ be the state set of the synchronous product $\hat{G}_1 \| \hat{G}_2$. For all $(x_1, x_2) \in \hat{R}_2$, $(x_1 \hat{\mathcal{W}}_2 x_2)$, where $\hat{\mathcal{W}}_2$ is the weak invariant simulation of $\hat{G}_2$ by $\hat{G}_1$ weak invariant simulations w.r.t. all shared events of G_1.

Proof. (Part (a)) By assumption $(x_j, x_{j+1}) \in \mathcal{Q}_{j+1}$. Therefore by the definition of weak invariant simulation, for any $l_{j+1} \in \Sigma_{j+1}^*$ with $\xi_{j+1}(x_{j+1}, l_{j+1}) \neq \emptyset$, there exists string $l_j \in \Sigma_j^*$ such that $\xi_j(x_j, l_j) \neq \emptyset$,

$$P_{\Sigma_j \setminus \Sigma_{j-1}}(l_j) = P_{\Sigma_j \setminus \Sigma_{j-1}}(l_{j+1}), \tag{4.11}$$

and for all $x'_j \in \xi_j(x_j, l_j)$ and $x'_{j+1} \in \xi_{j+1}(x_{j+1}, l_{j+1})$, $(x'_j, x'_{j+1}) \in \mathcal{Q}_{j+1}$. But $l_{j+1} \in \Sigma_{j+1}^*$, therefore

$$P_{\Sigma_j \setminus \Sigma_{j-1}}(l_{j+1}) = P_{\Sigma_j \cap \Sigma_{j+1}}(l_{j+1}). \tag{4.12}$$

This means that l_j contains no event shared with P_{j+1}; therefore $\hat{\xi}_j(x_j, l_j) \neq \emptyset$ and $P_{\Sigma_j \setminus \Sigma_{j-1}}(l_j) = P_{\Sigma_j \cap \Sigma_{j+1}}(l_j)$. By (4.11) and (4.12), $P_{\Sigma_j \cap \Sigma_{j+1}}(l_j) = P_{\Sigma_j \cap \Sigma_{j+1}}(l_j)$. Hence by the definition of weak invariant simulation, $(x_j, x_{j+1}) \in \hat{\mathcal{V}}_{j+1}$. Note that because the pair (x'_j, x'_{j+1}) is also member of $\mathcal{Q}_{j+1}$, we can use the same argument for this pair to conclude it belongs to $\hat{\mathcal{V}}_{j+1}$.

(Part(b)) By the fact that $\hat{R}_2 \in R_2$, (4.8) and the definition of weak invariant simulation. Details are omitted due to similarity to proof of part(a).

□

The following lemma expresses an important property of our proposed network: consider cycle G^N defined in (4.3) and let $\hat{G}^N$ be the isolated version of G^N. Then regardless of the evolution of the network, all the shared events of a given subprocess $\hat{G}_i$, $1 < i \leq N$, with the neighbor of 'lower' index, namely $\hat{G}_{i-1}$, can eventually be executed via a string whose execution does not change the states of subprocesses $\hat{G}_{i+1}$ to $\hat{G}_N$.

Lemma 3. Consider G^N defined in (4.3) and let $\hat{G}^N = (\hat{X}, \Sigma, \hat{\xi}, x^0, X_m)$ be the isolated version of G^N. For all $x \in \hat{X}$ and all $i \neq 1$, we have

$$(\forall \sigma_{i-1} \in \Sigma_{i-1} \cap \Sigma_i)[(\xi_i(x_i, \sigma_{i-1}) \neq \emptyset) \\ \Rightarrow (\exists s \in (\Sigma \setminus \tilde{\Sigma})^*)(\hat{\xi}(x, s\sigma_{i-1}) \neq \emptyset)], \tag{4.13}$$

where $\tilde{\Sigma} = \bigcup_{k=i}^N \Sigma_k$.

Proof. $\hat{G}^N$ has the topology of a ring network. Let $j \in J$, where J is the index set of output vertices. According to Proposition 8.(a), $\hat{G}_j \mathcal{V}_i \hat{G}_{j+1}$ and by Proposition 8.(b), for all $x \in \hat{R}_2$, $x_j \mathcal{W}_2 x_{j+1}$, where $\hat{R}_2$ is the state set of the synchronous product $\hat{G}_1 \| \hat{G}_2$. Let $x \in \hat{X}$ be such that for some $i \neq 1$ and $\sigma_{i-1} \in \Sigma_{i-1} \cap \Sigma_i$,

$$\xi_i(x_i, \sigma_{i-1}) \neq \emptyset. \tag{4.14}$$

According to Proposition 7, $(x_{i-1}, x_i) \in \mathcal{V}_i$, therefore by (4.14) and the definition of weak invariant simulation, there exists a string $l_{i-1} \in (\Sigma_{i-1} \setminus \Sigma_i)^*$ such that $\chi_{i-1}(\sigma_{i-1}) \in \xi_{i-1}(x_{i-1}, l_{i-1})$. If l_{i-1} has no events shared with G_{i-2} (consists of local events only); $\chi_{i-1}(\sigma_{i-1})$ can be reached in the global model by a local string of G_{i-1}. This satisfies (4.13).

For the case that l_{i-1} contains shared events with G_{i-2}, the proof of reachability of $\chi_{i-1}(\sigma_{i-1})$ in G_{i-1} within the global model is by induction on subprocess indices. To form the base case of the induction, let $i = 2$. By assumption (4.8), for any $x \in X$, $x_1 \mathcal{W}_2 x_2$. By (4.14), we have $\xi_2(x_2, \sigma_1) \neq \emptyset$; then, according to the definition of weak invariant simulation, there exists string $l_1 \in (\Sigma_{L_1})^*$ such that $\xi_1(x_1, l_1\sigma_1) \neq \emptyset$. Since l_1 consists of only local events of G_1, we have $l_1 \in (\Sigma \setminus \bigcup_{j=2}^N \Sigma_j)^*$ and satisfies (4.13). This forms the base case of the induction.

For the induction hypothesis, assume (4.13) holds when i is replaced with some $k > 1$; we will show that (4.13) holds for $k+1$. Suppose that for some event $\sigma_k \in \Sigma_k \cap \Sigma_{k+1}$, we have $\xi_{k+1}(x_{k+1}, \sigma_k) \neq \emptyset$. Then by the same reasoning as above, there exists a string $l_k \in (\Sigma_k \setminus \Sigma_{k+1})^*$ such that $\chi_k(\sigma_k) \in \xi_k(x_k, l_k)$. The only possible shared events of l_k are with G_{k-1}. Let α_{k-1} be the first shared event of l_k with G_{k-1}. According to the induction hypothesis, there exists string $s \in (\Sigma \setminus \bigcup_{j=k}^{N} \Sigma_j)^*$ such that $\xi(x, s\alpha_{k-1}) \neq \emptyset$. Therefore shared event α_{k-1} can be executed within the global model by a string s that contains no event in $\bigcup_{j=k}^{N} \Sigma_j$. The proof for reachability of the rest of the shared events of l_k within the global network is similar. Since $(\Sigma \setminus \bigcup_{j=k}^{N} \Sigma_j) \subseteq (\Sigma \setminus \bigcup_{j=k+1}^{N} \Sigma_j)$, we conclude that there exists a string $\hat{s} \in (\Sigma \setminus \bigcup_{j=k+1}^{N} \Sigma_j)^*$ such that $P_{\Sigma_k}(\hat{s}) = l_k$ and $\hat{s}$ can be executed within the global model. Since $\chi_k(\sigma_k) \in \xi_k(x_k, l_k)$ and $\xi_{k+1}(x_{k+1}, \sigma_k) \neq \emptyset$, σ_k can be executed and this completes the proof. □

4.2.1 Forward dependency property

Here we define a *forward dependency property* based on synchronous products of neighboring subprocesses in an isolated cycle of the network. This property aims to characterize the occurrence of a circular wait. According to Lemma 3, in an isolated cycle all the events of a subprocess shared with the neighbor of 'lower' index, can eventually be executed. Therefore the only shared events that may be blocked in an isolated cycle are the ones that are shared with the neighbor of 'larger' index. A state pair in a synchronous product of two neighboring subprocesses is forward dependent if the only event defined from it in the synchronous product is an event shared with the larger-index neighbor.

Definition 11. Consider cycle $G^N = \|G_i,\ 1 \leq i \leq N$ defined (4.3) and let $\hat{G}^N = (\hat{X}, \Sigma, \hat{\xi}, x^0, X_m)$ be the isolated version of G^N. For any i, $1 \leq i \leq N$, let $\hat{R}_i$ be the state set of the synchronous product $\hat{G}_{i-1}\|\hat{G}_i = (\hat{R}_i, \Sigma_{i-1} \cup \Sigma_i, \delta_i, x^0_{i-1} \times x^0_i, X_{m_{i-1}} \times X_{m_i})$. For a state pair $(x_{d_{i-1}}, x_{d_i}) \in \hat{R}_i$, we define the following property

$$(\forall \sigma_i \in \Sigma_{i-1} \cup \Sigma_i)[(\delta_i((x_{d_{i-1}}, x_{d_i}), \sigma_i) \neq \emptyset) \Rightarrow (\chi_{i+1}(\sigma_i) \neq \emptyset)]. \tag{4.15}$$

A state $x_d \in X_1 \times X_2 \times ... \times X_N$, is forward dependent if for all i, $(x_{d_{i-1}}, x_{d_i}) \in \hat{R}_i$ and $(x_{d_{i-1}}, x_{d_i})$ satisfies (4.15). We denote by $X_d \subseteq X$ the set of all forward dependent states of cycle G^N.

If a state pair $(x_{i-1}, x_i) \in \hat{R}_i$ satisfies (4.15), it means that the only transitions available from this pair in the synchronous product $\hat{G}_{i-1}\|\hat{G}_i$ are shared with the neighbor of 'larger' index in the isolated cycle, namely $\hat{G}_{i+1}$. For a reachable state x in $\hat{G}^N$, if property (4.15) holds for all i, then all the subprocesses of $\hat{G}^N$ are waiting for execution of an event shared with their respective immediate neighbors with larger index. Note that for such a state x, there can be events shared with subprocesses outside of cycle G^N defined from x_1 or x_j, $i \in J$, where J is the index set of output vertices of G^N. Execution of these shared events may break the circular wait within the cycle. Therefore existence of a circular wait in a cycle of the network may not necessarily cause a deadlock. We introduce the *dependency graph* in Section 4.2.2 to locate *generalized circular waits* among multiple cycles of the network which cause a local deadlock.

Reachability of forward dependent states

For deadlock analysis, we initially consider cycle $G^N = (X, \Sigma, \xi, x^0, X_m)$ in (4.3) and its isolated version $\hat{G}^N$. We shall show that any state $x_d \in X_d$ that satisfies the forward dependency property (4.15) is reachable in $\hat{G}^N$. This in turn means that x_d is reachable in G^N. The next proposition is the first step in proving this reachability. It states that in G^N, if there exists $x_d \in X_d$ and a reachable state $x \in X$ of G^N such that for some k, $1 \leq k \leq N$, x_{d_k} and x_k are one and the same, then there exists an executable string l that takes G_{k-1} from x_{k-1} to $x_{d_{k-1}}$ and contains no event from alphabets of subprocesses G_k to G_N.

Proposition 9. Consider $G^N = \|G_i = (X, \Sigma, \xi, x^0, X_m)$, $1 \leq i \leq N$, defined in (4.3). Let $\hat{G}^N = \|\hat{G}_i = (\hat{X}, \Sigma, \hat{\xi}, x^0, X_m)$, $1 \leq i \leq N$, be the restricted version of G^N. Consider state $x_d \in X_d$ of Definition 11, and a state $x \in \hat{X}$. For any $k \in \{2, 3, ..., N\}$, if x_k and x_{d_k} are one and the same, then there exists a string $l \in (\Sigma \setminus \bigcup_{r=k}^{N} \Sigma_r)^*$ such that $\hat{\xi}(x, l) \neq \emptyset$ and $x_{d_{k-1}} \in \hat{\xi}_{k-1}(x_{k-1}, P_{\Sigma_{k-1}}(l))$, where $\hat{\xi}_{k-1}$ is the transition function of $\hat{G}_{k-1}$.

Proof. See the appendix. □

Now, using the above proposition, the next lemma shows that any state in the state set of an arbitrary cycle of our proposed network that satisfies forward dependency property of Definition 11 is reachable within the isolated cycle, and hence in the global GPDES network.

Lemma 4. Consider cycle $G^N = (X, \Sigma, \xi, x^0, X_m)$ in (4.3), and let $\hat{G}^N = (\hat{X}, \Sigma, \hat{\xi}, x^0, X_m)$ be the isolated version of G^N. If the state subset X_d in Definition 11 is nonempty, then all of its members are reachable in $\hat{G}^N$.

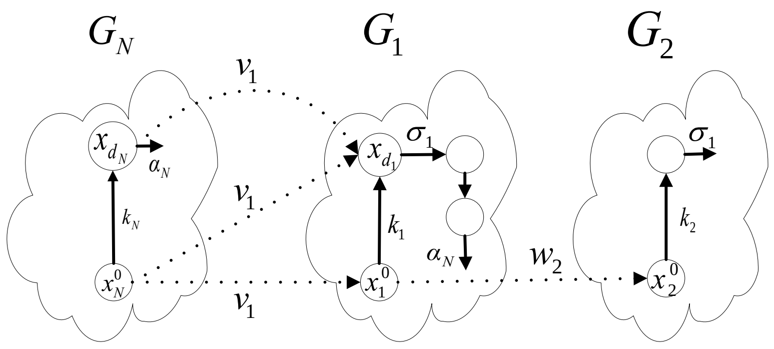

Figure 4.1: Subprocesses G_N and G_1 and the weak invariant simulation relation between them in the proof of Lemma 4.

Proof. The transitions and weak invariant simulations that appear in this part of the proof are shown in Figure 4.1. Consider an arbitrary $x_d \in X_d$. For any i, $1 \leq i \leq N$, by Definition 11, we have $(x_{d_{i-1}}, x_{d_i}) \in R_i$. Therefore by (4.6) and Proposition 7, we have

$$(\forall i)(x_{d_{i-1}}, x_{d_i}) \in \mathcal{V}_i, \tag{4.16}$$

where $\mathcal{V}_i$ is a weak invariant simulation w.r.t. $\Sigma_{i-1} \cap \Sigma_i$. By (4.8) and reachability of (x_{d_1}, x_{d_2}) in $G_1 \| G_2$,

$$(x_{d_1}, x_{d_2}) \in \mathcal{W}_2. \tag{4.17}$$

where $\mathcal{W}_2$ is a weak invariant simulation w.r.t. Σ_{S_1}.

According to the definition of isolated cycle $\hat{G}^N$, the only subprocesses affected by isolation are G_1 and G_j, $j \in J$, where J is the index set of output vertices. Therefore $\hat{G}_2$ and G_2 are one and the same. This means that $\hat{G}_2$ satisfies (4.4). By this assumption, there must exist a path in $\hat{G}_2$ from x_{d_2} to an event in $\Sigma_1 \cap \Sigma_2$. Therefore by (4.17), and the definition of weak invariant simulation, there exists a transition defined from x_{d_1}. But according to (4.15), the only transitions defined from x_{d_1} are via events that are shared transitions with $\hat{G}_2$. Let $\sigma_1 \in \Sigma_1 \cap \Sigma_2$ be such that $\xi_1(x_{d_1}, \sigma_1) \neq \emptyset$. By the definition of X_d, $\chi_2(\sigma_1)$ is nonempty. Again by (4.4), there exists a string $k_2 \in \Sigma_2^*$ such that $\chi_2(\sigma_1) \cap \xi_2(x_2^0, k_2) \neq \emptyset$. By (4.8), $(x_1^0, x_2^0) \in \mathcal{W}_2$; therefore by the definition of weak invariant simulation there exists string $k_1 \in ((\Sigma_1 \cap \Sigma_2) \cup \Sigma_{L_1})^*$ such that $\chi_1(\sigma_1) \cap \xi_1(x_1^0, k_1) \neq \emptyset$. Therefore by (4.5) and the fact that $k_1 \in ((\Sigma_1 \cap \Sigma_2) \cup \Sigma_{L_1})^*$, we have $x_{d_1} \in \hat{\xi}_1(x_1^0, k_1) \neq \emptyset$. In other words, x_{d_1} is reachable in $\hat{G}_1$. On the other hand, by setting $i = 1$,

$$(x_N^0, x_1^0) \in \mathcal{V}_1. \qquad \text{(by (4.6))}$$

Therefore by the definition of weak invariant simulation, and the fact that $\hat{k}_1$ contains no event shared with G_N,

$$(x_N^0, x_{d_1}) \in \mathcal{V}_1. \tag{4.18}$$

By (4.4), there exists a shared event $\alpha_N \in \Sigma_N \cap \Sigma_1$ whose companion state in G_1 is accessible from x_{d_1} via strings in $(\Sigma_1 \setminus \Sigma_N)^*$. According to (4.16), we also have $(x_{d_N}, x_{d_1}) \in \mathcal{V}_1$, and by (4.15) the only transitions defined from x_{d_N} are shared transitions with G_1. Hence

$$\xi_N(x_{d_N}, \alpha_N) \neq \emptyset. \tag{4.19}$$

But $(x_N^0, x_{d_1}) \in \mathcal{V}_1$, therefore there must exist a string $k_N \in (\Sigma_N \setminus \Sigma_1)^*$ such that

$$\chi_N(\alpha_N) \in \xi_N(x_N^0, k_N) \qquad \text{(by (4.5) and (4.19))}$$

But isolation of G_N has no effect on G_N (because according to network structure, G^N is not a distinguished subprocess); therefore $(x_{d_N} \in \hat{\xi}_N(x_N^0, k_N))$. String k_N belongs to the set $(\Sigma_N \setminus \Sigma_1)^*$, in other words it contains no event shared with G_1. By Lemma 3, all the shared events of k_N with $\hat{G}_{N-1}$ can eventually be executed, therefore $\hat{G}_N$ can reach x_{d_N} within the global system.

By Proposition 9, $(x_{d_{N-1}}, x_{d_N})$ can be reached in the isolated cycle $\hat{G}^N$. But again, we use Proposition 9 and the fact that $\hat{G}_{N-1}$ can reach $x_{d_{N-1}}$, to show that $x_{d_{N-2}}$,$x_{d_{N-1}}$, and x_{d_N} are simultaneously reachable in $\hat{G}^N$. With the same reasoning and after $N-1$ times application of Proposition 9, it can be shown that state $x_d \in X_d$ is reachable $\hat{G}^N$. Since $\hat{G}^N$ is the restricted version of G^N.

□

4.2.2 Dependency graph

The forward dependent states of isolated cycles need not represent partial or global deadlocks in the global GPDES network. For the purpose of this chapter, we define *local deadlocks*. A local deadlock refers to the case where the subprocesses of one or multiple cycles in the GPDES network cannot execute any event regardless of the evolution of the rest of the network.

Definition 12. Let X' be the state set of subprocesses of a strongly connected subgraph of the network graph (that is, the Cartesian product of the state sets of these subprocesses); then $x \in X'$ is a *local deadlocked* state if no transition is possible from x within the global network.

Our global analysis is a generalization of an analysis of individual isolated cycles of the network. We relate the local deadlocks to forward dependent states by construction of a directed graph called the *dependency graph*.

The procedure to calculate arcs (A_D) and vertices (V_D) of the dependency graph is given in Algorithm 1: to build the graph, we first calculate the set of state pairs that satisfy (4.15) for each isolated cycle of the network graph (set *Depend*). All states x' or x'' such that $(x', x'') \in$ *Depend* are the vertices of the dependency graph. All pairs (x', x'') such that $(x', x'') \in$ *Depend* are the arcs of the dependency graph.

Construction of the dependency graph involves the following operations. First locating all cycles of the network graph; given the fact that the network has only one input vertex, the time complexity of this operation is the same as depth first traversal of a tree: $O(|V|)$, where V is the vertex set of network graph. Second, for each cycle, calculation of isolated cycles of the network graph with time complexity $O(|A||V|M_1^2)$, where A is the arc set of the network graph and M_1 is the maximum number of events in all the subprocess of the network. Finally execution of Algorithm 1 with time complexity $O(({M_2}^4 M_1)(|A| + |V|^2))$, where M_2 is the maximum numbers of states in all subprocess of the network. Therefore the overall complexity of the construction of the dependency graph is $O(|A||V|M_1^2 + ({M_2}^4 M_1)(|A| + |V|^2))$.

The next definition states the *consistency* property of a subgraph of the dependency graph and explains how a consistent subgraph of the dependency graph *represents* a state of a part of the network – specifically of a strongly connected subgraph of the network graph.

Definition 13. A strongly connected subgraph $\bar{\mathscr{D}}$ of the dependency graph $\mathscr{D}$, is *consistent* if it contains a state of the input vertex and does not contain two states of any distinguished subprocess.

Algorithm 1 The creation of dependency graph $\mathscr{D}$.

```
for all linear PDES of the network graph do
  $S$ =synchronous products of two neighboring subprocesses
  $Depend$ = pairs in state set of $S$ that satisfy (4.15) $\cup$ $Depend$
end for
for all isolated cycles in the network graph do
  Uniformly label subprocesses from $\hat{G}_1$ to $\hat{G}_N$
  for all $k \in K$, where $K$ is the index set distinguished subprocesses do
    $S_k = \hat{G}_{k-1} \| \hat{G}_k$, $S_{k+1} = \hat{G}_k \| \hat{G}_{k+1}$.
    $Depend$ = pairs in state set of $S_k$ and $S_{k+1}$ that satisfy (4.15) $\cup$ $Depend$
  end for
end for
$V_D$ = the set of all states $x'$ or $x''$ such that $(x', x'') \in Depend$
$A_D$ = all pairs $(x', x'')$ such that $(x', x'') \in Depend$.
```

A consistent subgraph $\bar{\mathscr{D}}$ *represents* a set of states; each member of the set is the state of subprocesses of a strongly connected subgraph of the network graph: if a vertex is the state of a distinguished subprocess, the vertex represents the state of that distinguished subprocess. The direct successors of that vertex in $\bar{\mathscr{D}}$ represent the states of adjacent subprocesses in parameterized sections of the network. In turn, a direct successor of one of the latter vertices represents the state of a possible neighboring subprocess whether it be a distinguished subprocess or another subprocess in the same parameterized section.

For an example of consistent subgraphs and the states they represent see the case study of the chapter (Section 4.3).

The next proposition expresses that for any component x_i of a forward dependent state x, there exists an event defined from x_i in G_i that is shared with G_{i+1}. Then it shows that any state represented by a cycle in a consistent subgraph of the dependency graph is reachable in the corresponding isolated cycle of the network.

Proposition 10. Consider cycle $G^N = (X, \Sigma, \xi, x^0, X_m)$ in (4.3), and let

$$\hat{G}^N = (\hat{X}, \Sigma, \hat{\xi}, x^0, X_m)$$

be the isolated version of G^N. Let $x \in X$ be represented by a cycle in a consistent subgraph of the dependency graph $\mathscr{D}$ and J be the index set of output vertices of G^N in the network graph. (a) For any $j \in J$, $(x_j, x_{j+1}) \in \mathcal{Q}_{j+1}$, where $\mathcal{Q}_{j+1}$ is the weak invariant simulation of G_{j+1} by G_j w.r.t. $\Sigma_{S_j} \setminus \Sigma_{j-1}$. (b) For any x_j, $i \in J$, $1 \leq j \leq N$, there exists an event $\sigma \in \Sigma_j \cap \Sigma_{j+1}$ such that $\hat{\xi}_j(x_j, \sigma_j) \neq \emptyset$. (c) Any such state x is reachable in $\hat{G}^N$.

Proof. See the appendix. □

To formulate an appropriate generalization of circular waits, we define *full* subgraphs. This property deals with the issue of output subprocesses in deadlock analysis: a state of an output subprocess may have events shared with subprocesses of different parameterized sections of the network. In a deadlock, none of these shared events can be executed. Therefore, to locate deadlocks we need to consider subgraphs of the dependency graph that take account of all such events that are possible in a given state of the output subprocess.

Definition 14. Let $\bar{\mathscr{D}}$ be a subgraph of the dependency graph $\mathscr{D}$. In $\bar{\mathscr{D}}$, consider vertex x_j of an output subprocess G_j in the network graph. For every event σ defined from x_j in G_j that is shared with a direct successor of G_j in the network graph, let Y^σ be the state set of that direct successor. Subgraph $\bar{\mathscr{D}}$ is *full* if for every such x_j and σ, and some $y^\sigma \in Y^\sigma$, $\bar{\mathscr{D}}$ contains the arc (x_j, y^σ).

By Proposition 10, the existence of an arc in the dependency graph indicates the forward dependency of the corresponding state pair in the arc.

Consistent subgraphs are strongly connected by definition. It follows that full, consistent subgraphs represent sets of states that constitute generalized circular waits. In support of this interpretation, we have the following claim.

Claim 1. Any full, consistent subgraph of the dependency graph represents a set of locally deadlocked states in the proposed GPDES network.

Our goal is to detect all *reachable* generalized circular waits in the global network. The *output-reachability* property defined below provides a necessary and sufficient condition for the reachability of states represented by a full, consistent subgraph of the dependency graph.

Definition 15. Consider a full, consistent subgraph of the dependency graph $\mathscr{D}$, and let x be a state that it represents. Let J be the index set of output vertices. This subgraph is *output-reachable* if every state x_j of G_j, $j \in J$, is reachable in G_j by local events and events shared with its unique direct predecessor in the network graph.

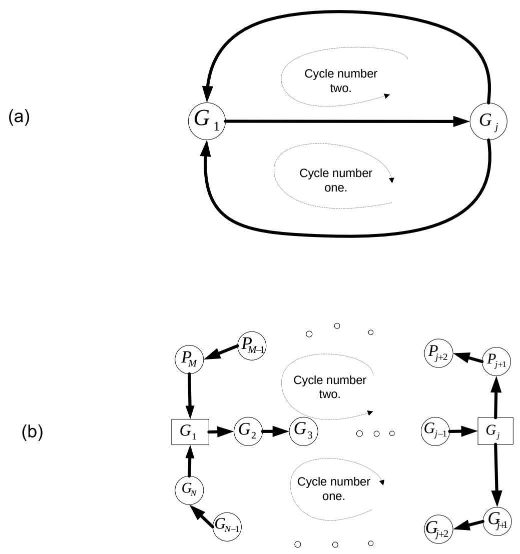

Figure 4.2: (a) Graph representation of a network with branching which has two cycles and three parameterized sections (b) Equivalent presentation of the network in part (a).

While a consistent subgraph represents a set of states each corresponding to specific instance of the GPDES, it determines a unique state x_j, $j \in J$ of any output subprocess. To check output-reachability of a subgraph, it suffices to check the appropriate reachability of all states x_j, $j \in J$, within the respective G_j.

The next claim deals with the reachability of the set of locally deadlocked states calculated by Claim 1.

Claim 2. A state represented by a full, consistent subgraph of $\mathscr{D}$ is reachable in global GPDES $\mathcal{G}$ if and only if that subgraph is output-reachable.

Claims 1 and 2 will be proved as Theorem 5 below. The rest of the chapter concerns technical details that lead to the proof of the theorem.

4.2.3 Technical details of the analysis

The proof of the above claim in the proposed network topology is involved. To aid readability, we therefore first consider the particular network structure of Figure 4.2 and carry

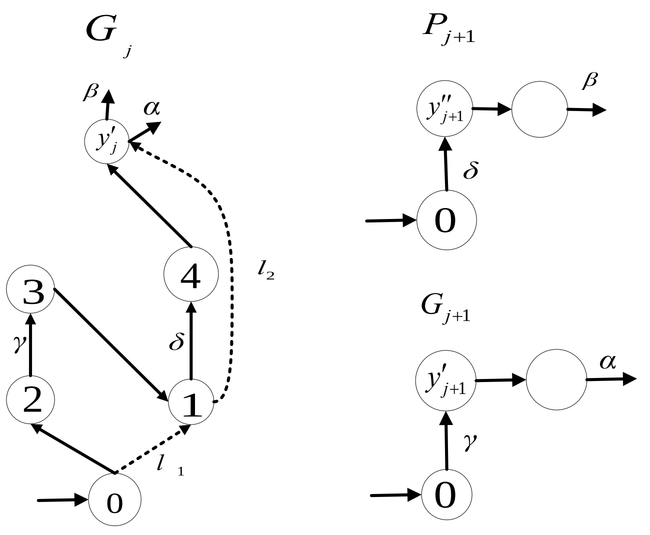

Figure 4.3: Subprocesses G_j, G_{j+1} and P_{j+1} in the proof of Proposition 11. The unlabeled transitions in G_j contain only local events and events shared with G_{j-1}. In order for y'_j,y'_{j+1},y''_{j+1} to be simultaneously reachable in the global network, there must a path in G_j from state 0 to y'_j that contains both γ and δ. This path is shown by solid transitions. The dotted transition l_1l_2 in G_j consist of local events and events shared with G_{j-1}.

out the deadlock analysis (Theorem 4). Then we generalize the results to proposed network topology by induction on the structure of the network (Theorem 5).

Let $\mathscr{D}$ be the dependency graph of the network structure of Figure 4.2 and y be a state represented by a full, consistent subgraph of $\mathscr{D}$. In y, assume y_j is the state of output subprocess G_j. The following proposition establishes a relation between reachability of y in the global network and the path defined from y_j^0 to y_j in G_j.

Proposition 11. Consider cycle $G^N = (X, \Sigma, \xi, x^0, X_m)$ in (4.3), and the network structure of Figure 4.2. Let $P_{j+1} = (X_{P_{j+1}}, \Sigma_{P_{j+1}}, \xi_{P_{j+1}}, x^0_{P_{j+1}}, X_{m_{P_{j+1}}})$. Let y be a state represented by a full, consistent subgraph $\bar{\mathscr{D}}$ of the dependency graph of the network. Assume events $\alpha \in \Sigma_{j+1}$ and $\beta \in \Sigma_{P_{j+1}}$ are defined from y_j in G_j. State y is reachable in the global network if and only if $\bar{\mathscr{D}}$ is output-reachable.

Proof. Let y' and y'' be the components of y corresponding to states of cycles one and two respectively. Then y'_1 to y'_j and y''_1 to y''_j are one and the same, corresponding to states of G_1 to G_j respectively.

(*If*) According to Proposition 10(c), y' and y'' are forward dependent states. By Proposition 10(c) and Lemma 4 state y' is reachable in isolated cycle number one and therefore within the global network. Let states y'_1 to y'_j be the new initial states of G_1 to G_j respectively. We will show that cycle number two can reach y'' from these initial states and without changing the states of subprocesses G_j to G_{N-1} of cycle number one. By output-reachability assumption, there exists a path in G_j from the initial state to $y_j = y'_j$ consisting of local events and events shared with G_{j-1}. So by (4.9) and the definition of weak invariant simulation,

$$(y'_j, x^0_{P_{j+1}}) \in \mathcal{Q}_{P_{j+1}}$$

where $\mathcal{Q}_{P_{j+1}}$ is the weak invariant simulation of P_{j+1} by G_j w.r.t. $\Sigma_{S_j} \setminus \Sigma_{j-1}$. Therefore by Proposition 8.(a), $(y'_j, x^0_{P_{j+1}}) \in \hat{\mathcal{V}}_{P_{j+1}}$, where $\hat{\mathcal{V}}_{P_{j+1}}$ is a weak invariant simulation of P_{j+1} by $\hat{G}_j$ w.r.t. events shared between $\hat{G}_j$ and P_{j+1} (P_{j+1} is not affected by the isolation of the cycle number two). Therefore by Proposition 8.(b), cycle number two satisfies the assumptions of Lemma 4. In fact, by the proof of Lemma 4, y'' is reachable in isolated cycle number two. This means that components y' and y'' of y are simultaneously reachable within the global network.

(*Only if*) Assume that state y is reachable within the global state network. We will show that there exists string r containing only local events and events shared with G_{j-1} such that $y_j \in \xi_j(x^0_j, r)$. By assumption, shared events β and α are defined from y_j. Since y is a forward dependent state, by Proposition 10(a),

$$(y'_j, y'_{j+1}) \in \mathcal{Q}_{j+1},$$

where $\mathcal{Q}_{j+1}$ is the weak invariant simulation of G_{j+1} by G_j w.r.t. $\Sigma_{S_j} \setminus \Sigma_{j-1}$. By (4.4), and the fact that α is defined from y'_j, there exists string $s'_{j+1} \in (\Sigma_{j+1} \setminus \Sigma_j)^*$ and event $\sigma \in \Sigma_j \cap \Sigma_{j+1}$ such that $\xi_{j+1}(y'_{j+1}, s'_{j+1}\sigma) \neq \emptyset$. Again by (4.4), there exists string $s_{j+1} \in \Sigma^*_{j+1}$ such that $y'_{j+1} \in \xi_{j+1}(x^0_{j+1}, s_{j+1})$. Therefore by (4.5) and the definition of weak invariant simulation there also exists a path s_j from the initial state of G_j to y'_j such that the projections of s_j and $s_{j+1}s'_{j+1}$ onto the set $\Sigma_{S_j} \setminus \Sigma_{j-1}$ are the same. If s_j contains only local events and events shared with G_{j-1}, the result holds. Assume therefore that all such s_j contain an event in $\Sigma_j \cap \Sigma_{j+1}$. But given that $\xi_{j+1}(x^0_{j+1}, s_{j+1}s'_{j+1}\sigma) \neq \emptyset$, in order for (4.9) to hold, any s_{j+1} must also contain such an event γ. Similarly, consider any path from $x^0_{P_{j+1}}$ to $\chi_{P_{j+1}}(\beta)$, and assume it contains an event $\delta \in \Sigma_j \cap \Sigma_{P_{j+1}}$ (see Figure 4.3.)

By assumption y'_j, y'_{j+1} and y''_{j+1} are simultaneously reachable in the global network. Therefore, there must exist a string $l \in \Sigma_j^*$ such that $x_j \in \xi_j(x_j^0, l)$ and l contains suitable events γ and δ. String s_{j+1} is defined from x_{j+1}^0 in G_{j+1}; therefore by (4.9) there must exist a path from x_j^0 to y_j that contains δ but no event in $\Sigma_j \cap \Sigma_{P_{j+1}}$ (does not contain γ). With similar reasoning for P_{j+1}, there must exist a path from x_j^0 to x_j that contains δ but no event in $\Sigma_j \cap \Sigma_{j+1}$ (hence does not contain γ). Since by (4.5) companion states of shared events γ and δ are unique in G_j, there exists a string from x_j^0 to y_j that contains no event in $\Sigma_j \cap \Sigma_{j+1}$ or $\Sigma_j \cap \Sigma_{P_{j+1}}$. This is demonstrated in Figure 4.3. Consider the state labellings of this figure for the rest of the proof. By (4.9), $G_j Q_{j+1} G_{j+1}$, where Q_{j+1} is the weak invariant simulation of G_{j+1} by G_j w.r.t. $\Sigma_{S_j} \setminus \Sigma_{j-1}$. Therefore by the definition of weak invariant simulation, pair $(3, y'_{j+1}) \in Q_{j+1}$. Since the transition between states 3 and 1 consists of only local events and events shared with G_{j-1}, therefore $(1, y'_{j+1}) \in Q_{j+1}$. Since G_{j+1} can reach companion state of α from y'_{j+1}, by (4.5) there must exists path l_2 from state 1 to y'_j, containing only local events and events shared with G_{j-1}. On the other hand, by (4.9) G_j weakly invariantly simulates P_{j+1} w.r.t. $\Sigma_{S_j} \setminus \Sigma_{P_{j-1}}$. Since δ is reachable from initial state of P_{j+1}, by the definition of weak invariant simulation and (4.5), there must exist a path l_1 from initial state of G_j to state 1. Therefore $l_1 l_2$ constitutes a path from initial state of G_j to y'_j containing only local events and events shared with G_{j-1}. Therefore y' and y'' are components of a state represented by an output-reachable subgraph of the dependency graph.

Note that in a network with branching topology G_j may have multiple direct successors in the graph network, and l may contain events shared with the rest of the direct successors of G_j in the graph network. However, the general proof is similar to that presented above.

□

Using the above proposition, we are able to perform deadlock analysis for the network structure of Figure 4.2. The deadlock analysis involves the following question: is a forward dependent state represented by a dependency graph in fact a deadlocked state? The next theorem provides a response for the case of the network structure of Figure 4.2. As we stated in the network description, there are two types of special vertices in the network graph: the input vertex and output vertices. The argument of the theorem is based on shared events defined from the state of an output subprocess.

Theorem 4. Consider a GPDES with the network structure of Figure 4.2. Let $\mathscr{D}$ be the dependency graph of this GPDES. Let $G^N = (X, \Sigma, \xi, x^0, X_m)$. Consider state $x \in X$.

(a) Assume there is no event defined from x_j in G_j that is shared with subprocesses outside of G^N. State x is a (reachable) local deadlock if and only if it is represented by a consistent subgraph of $\mathscr{D}$.

(b) Assume there is an event defined from x_j in G_j shared with G_{j+1}, and another event shared with a subprocess outside of G^N, say P_{j+1}. Let Y be the state set if cycle number two in Figure 4.2(b). States x and $y \in Y$ consist of components of a reachable local deadlock if and only if x and y are components of an output-reachable state represented by a full, consistent subgraph of $\mathscr{D}$.

Proof. Part (a): (If) Assume x is represented by a full, consistent subgraph of $\mathscr{D}$, but is not a deadlocked state. According to Proposition 10(c), x is a forward dependent state. Therefore by Lemma 4 state x is reachable in $\hat{G}^N$, and therefore is reachable within the global network. By assumption state x is not deadlocked; therefore for some event $\beta \in \Sigma_j$, $\xi(x, \beta) \neq \emptyset$. By the structure of the network, only G_1 and G_j have events shared with subprocesses outside of G^N. But by assumption, there is no event defined from x_j in G_j that is shared with subprocesses outside of G^N. By (4.8) and the definition of forward dependent states, there is an event shared with G_2 defined from x_1 in G_1. Therefore by (4.7), there is no event defined from x_1 that is shared with P_M. This means that β is an event defined in one of the subprocesses of isolated cycle $\hat{G}^N$. Since x is a forward dependent state, by (4.15) β must be a shared event. Let i, $1 \leq i \leq N$ be such that $\beta \in \Sigma_{i-1} \cap \Sigma_i$. Given that $\xi(x, \beta) \neq \emptyset$, if δ_i is the transition function of $\hat{G}_{i-1} \| \hat{G}_i$, we have $\delta_i((x_{d_{i-1}}, x_{d_i}), \beta) \neq \emptyset$. But by (4.15), $\chi_{i+1}(\beta) \neq \emptyset$. This means that a β transition is defined both in $\hat{G}_{i+1}$ and in $\hat{G}_{i-1}$. This contradicts the network assumption that only neighboring subprocesses have events shared between them.

$(Only\ if)$ Consider an arbitrary reachable local deadlocked state $x \in X$. We will show that this state belongs to the subset X_d of Definition 11. Since x is a reachable state, then for all i, $1 \leq i \leq N$,

$$(x_{i-1}, x_i) \in R_i, \tag{4.20}$$

where R_i is the state set of the synchronous product $G_{i-1} \| G_i$.

Suppose that for some i, (x_{i-1}, x_i) does not satisfy (4.15); i.e., for some $\sigma \in (\Sigma_{i-1} \cup \Sigma_i) \setminus \Sigma_{i+1}$ the transition $\delta_i((x_{i-1}, x_i), \sigma)$ is defined. By assumption, σ is not an event shared with subprocesses outside of G^N. If σ is a local event or $\sigma \in \Sigma_{i-1} \cap \Sigma_i$, then it can be executed and this contradicts the assumption that x is a deadlocked state. If σ is in $\Sigma_{i-2} \cap \Sigma_{i-1}$, this means that $\xi_{i-1}(x_{i-1}, \sigma) \neq \emptyset$. We consider two cases: first, if $\sigma \notin \Sigma_N \cap \Sigma_1$, then by Lemma 3, σ can eventually be executed, a contradiction. If $\sigma \in \Sigma_N \cap \Sigma_1$, then $\xi_1(x_1, \sigma) \neq \emptyset$ (because $\delta_i((x_{i-1}, x_i), \sigma) \neq \emptyset$). Note that Lemma 3 is inapplicable for shared events between G_N and G_1, so it cannot be directly used to derive a contradiction. From (4.6) and (4.22), we have, by Proposition 7, $(x_N, x_1) \in \mathcal{V}_1$; therefore by the definition of weak invariant simulation, there exists a string $l_N \in (\Sigma_N \setminus \Sigma_1)^*$ such that $\xi_N(x_N, l_N\sigma) \neq \emptyset$. If l_N is the empty string, then σ can be executed. If the first symbol of l_N is a local event, then this local event can be executed. If the first symbol of l_N is some event $\Sigma_N \cap \Sigma_{N-1}$, then by Lemma 3, this event can eventually be executed within the global model. Execution of any event is a contradiction of the assumption that x is a deadlocked state. Hence we conclude that (x_{i-1}, x_i) satisfies (4.15). This, together with (4.22) means x is in fact a member of X_d.

Now we show that for all i, $(x_{i-1}, x_i) \in \hat{R}_i$, where $\hat{R}_i$ is the state set of the synchronous product $\hat{G}_{i-1} \| \hat{G}_i$. Because G_1 and G_j are the only subprocesses that have shared events with subprocesses outside of G^N, for $i \neq 1, j$, subprocesses G_i and $\hat{G}_i$ are the same. Therefore we only need to show $(x_{i-1}, x_i) \in \hat{R}_i$ for $i = 1, 2, j, j+1$. By (4.20), (4.6) and Proposition 7, $(x_{i-1}, x_i) \in \mathcal{V}_i$. According to (4.4), there must exist an event defined from x_j. With the reasoning provided above, this event must be in $\Sigma_j \cap \Sigma_{j+1}$. Again by (4.4), there exists $l_{j+1} \in (\Sigma_{j+1} \setminus \Sigma_j)^*$ such that for some $\sigma \in \Sigma_j \cap \Sigma_{j+1}$, $\xi_{j+1}(x_{j+1}, l_{j+1}\sigma) \neq \emptyset$. Therefore by (4.9), and the fact that events defined from x_i are in $\Sigma_j \cap \Sigma_{j+1}$, we must have $\xi_j(x_j, \sigma) \neq \emptyset$. Let s_{j+1} be the string labeling any path from x^0_{j+1} to x_{j+1}. Consider string $s_{j+1}l_{j+1}$. Since $\xi_{j+1}(x_{j+1}, s_{j+1}l_{j+1}) \neq \emptyset$, by (4.9), there must exist a string $s_j \in \Sigma_j^*$ such that $\xi_j(x_j, s_j\sigma) \neq \emptyset$ and $P_{\Sigma_{S_j} \setminus \Sigma_{j-1}}(s_j) = P_{\Sigma_{S_j} \setminus \Sigma_{j-1}}(s_{j+1}l_{j+1})$. But $l_{j+1} \in (\Sigma_{j+1} \setminus \Sigma_j)^*$; therefore

$$P_{\Sigma_{S_j} \setminus \Sigma_{j-1}}(s_j) = P_{\Sigma_{S_j} \setminus \Sigma_{j-1}}(s_{j+1}). \tag{4.21}$$

Since $s_{j+1} \in \Sigma_{j+1}^*$, s_j contains no event shared with P_{j+1}. By (4.5), companion state of σ in G_j is unique, therefore $(x_{j-1}, x_j) \in \hat{R}_j$. By (4.21), $P_{\Sigma_j \cap \Sigma_{j+1}}(s_j) = P_{\Sigma_j \cap \Sigma_{j+1}}(s_{j+1})$. Therefore $(x_j, x_{j+1}) \in \hat{R}_{j+1}$. Similar reasoning can be used to show that $(x_N, x_1) \in \hat{R}_1$ and $(x_1, x_2) \in \hat{R}_2$. Therefore for all i, $1 \leq i \leq N$,

$$(x_{i-1}, x_i) \in \hat{R}_i, \tag{4.22}$$

Part (b): By Proposition 11, states x and y are simultaneously reachable. Here we will only show that x and y are components of a local deadlock if and only if x and y are components a state represented by a full, consistent subgraph of $\mathscr{D}$.

(*If*) We assume that x and y are components of a state represented by a full, consistent subgraph of $\mathscr{D}$, and show that they are components of a local deadlock. States x and y are forward dependent states; by Proposition 10(b) some event $\alpha \in \Sigma_{j+1}$ ($\beta \in \Sigma_{P_{j+1}}$) is defined from x_j in G_j. Furthermore, by the forward dependency property no local events or events in Σ_{j-1} are defined from x_j. Without loss of generality, we assume that only α and β are defined from x_j. Event β is shared with P_{j+1}, but P_{j+1} to P_{j+M} are in states y_{j+1} to y_{j+M}. Since y is a forward dependent state, the only events defined from y_k, $j+1 \leq k < j+M$, are shared with respective neighbors of 'larger' index. The only shared event defined from y_{j+M} is shared with G_1. This forms a circular wait. Therefore no event other than α can occur from state y. But since β cannot be executed and cycle number one is in a forward dependent state x, by part (a) of the theorem no event (including α) can be executed from x. Therefore x and y are components of a local deadlocked state.

(*Only if*) Consider states $x' \in X$ and $y' \in Y$ such that there is an event defined from x_j in G_j that is shared with P_{j+1}. Assume that x' and y' are simultaneously reachable local deadlocked states; then x'_1 to x'_j are the same as y'_1 to y'_j. We will show that x' and y' are forward dependent states. Since y' is a local deadlocked state for cycle number two, no event shared between G_j and P_{j+1} can be executed. Therefore by the proof of the 'only if' section of part (a) of the theorem, $x' \in X_d$. Since x' is a local deadlock, no event shared between G_j and G_{j+1} can be executed. By similar reasoning, $y' \in Y_d$. □

Considering cycle G^N in cycle number one, Theorem 4 provides a deadlock analysis for reachable states of G^N based on the state of the distinguished subprocess G_j. The analysis is done by forming the dependency graph. For each forward dependent state x, a necessary and sufficient deadlock condition is given based on the transitions defined from x_j (the component of x corresponding to states of G_j). Part (a) of the above theorem gives the deadlock condition for the case where there is no event defined from x_j in G_j that is shared with subprocesses outside of G^N. If there is an event defined from x_j shared with P_{j+1}, and another event defined from x_j shared with G_{j+1}, then part (b) of the theorem provides the deadlock condition for both cycle one and two. Finally, if there is an event defined in x_j shared with P_{j+1} but no other event defined in x_j shared with G_{j+1}, then by interchanging cycle number one and two, part (a) of the theorem applies for cycle number two.

Based on the analysis of the network in Figure 4.2, we are able to perform the deadlock analysis for the general network with proposed topology. The following theorem, formerly stated as Claims 1 and 2, relates reachable local deadlocks to states represented by specific subgraphs of the dependency graph.

Theorem 5. Consider GPDES $\mathcal{G}$ with proposed branching topology. Let $\mathscr{D}$ be the dependency graph of this GPDES. For any full, consistent subgraph $\bar{\mathscr{D}}$ of $\mathscr{D}$ we have: (a) Any state x represented by $\bar{\mathscr{D}}$ is a local deadlocked state. (b) Any such x is reachable in the global network if and only if $\bar{\mathscr{D}}$ is output-reachable.

Proof. (a) Similar to proof of Theorem 4(b): Since $\mathscr{S}$ is a full subgraph of the dependency graph $\mathscr{D}$, by the definition of forward dependency, any event defined from any state of one subprocess is shared with the 'next' subprocess (the next subprocess in the direction of network graph arcs). This forms a generalized circular wait among the states presented by $\mathscr{S}$ and consequently a local deadlock.

(b) *(If)* Consider a full, consistent output-reachable subgraph $\mathscr{S}$ of $\mathscr{D}$. Assume that x represented by this subgraph. We show that x is reachable in $\mathcal{G}$. The proof is by induction on the structure of this subgraph. Since $\mathscr{S}$ is consistent, it contains a cycle that includes the input vertex. By Lemma 4 the state $x \in X_d$ of this cycle is reachable within the global network. This forms the base case of the induction. Now consider a consistent subgraph $\mathscr{S}'$ of $\mathscr{S}$ and assume it represents a reachable state set in $\mathcal{G}$. If $\mathscr{S}'$ and $\mathscr{S}$ are the same; we then have the the result by assumption. Otherwise, there must exist an output vertex in $\mathscr{S}'$ and an arc from that vertex that exists in $\mathscr{S}$ but not in $\mathscr{S}'$. Therefore there exists a consistent subgraph $\mathscr{S}''$ of $\mathscr{S}$ that is formed by adding a path to $\mathscr{S}'$ from an output vertex to the input vertex. By assumption, the state represented by $\mathscr{S}'$ is reachable. Now consider the cycle in $\mathscr{S}''$ that includes the new path. By the proof of Theorem 4(b), the state represented by this cycle and the state represented by $\mathscr{S}'$ are simultaneously reachable. This completes the induction. Therefore $\mathscr{S}$ represents a reachable state in GPDES $\mathcal{G}$.

(Only if) Similar to the proof of the 'only if' part of Proposition 11.

□

4.2.4 Applicability of the results to general networks

The framework presented in this chapter can be extended to more general network topologies. In this thesis, we considered a GPDES network represented by directed graph. We assumed two specific limitations for the graph representation of the network:

(a) network graph contains only one input node;

(b) network graph is strongly connected.

The main purpose of the first assumption is avoidance of cumbersome mathematical proofs. The results of this chapter can be extended to the case of a network represented by a strongly connected graph and contains multiple input nodes. However, all the input nodes of the network must satisfy (4.7) and (4.8). For this extension, we use a slightly different version of the definition of consistency of subgraphs of the dependency graph:

Definition 16. A strongly connected subgraph $\bar{\mathscr{D}}$ of the dependency graph $\mathscr{D}$, is consistent if it contains a state of *any* input vertex and does not contain two states of any distinguished subprocess.

In the above updated version of the definition, a consistent subgraph is required to contain at least one of the input nodes. Using this definition, it can easily be shown that the result of Theorem 5 holds for the extended network.

The second assumption on the graph representation of the network is the strong connectivity. Since our main concern is calculation of reachable generalized circular waits, this assumption is arguably natural. In any case, this assumption may also be removed; however the extension of results involves re-writing some the proofs without the use of notion of isolated cycle. Note that in a network without strong connectivity assumption, there may be some subprocesses that do not belong to any cycle. The results of the paper are not dependent on the use of isolated cycles. This notion is employed mainly because of the similarity of the analysis of isolated cycles to that of ring networks (Chapter 2). Algorithm 1 can also be re-written without use of isolated cycles, by appropriate disabling of events of output nodes in specific steps of the algorithm.

4.3 Illustrative example: small factory

To illustrate our framework, we present a large-scale factory consisting of three 'distinguished' machines, A_1, A_2 and A_3, three parameterized production lines P, P' and G, and two parameterized buffers, B and B'. Figure 4.4.(a) shows the graph representation of the network and Figure 4.4.(b) is the equivalent GPDES of network graph.

The distinguished subprocess A_1 with event set $\{in, s_1, f_N, c_M, c'_{M'}\}$ provides workpieces required for the factory by either receiving parts from factory input (local event in), or from production lines G, P and P' by events f_N, c_M and $c'_{M'}$ respectively.

The i_th and $i-1_{th}$, $1 < i < N_G$, subprocesses of production line G are shown is Figure 4.6(a). A workpiece can be sent back and forth between subprocesses of this production line. Subprocess G_i receives parts from G_{i-1} by event f_{i-1} and enters state 2. In this state, it decides to either process the part (local event $proc$) or return the part to G_{i-1} by first executing event r_{i-1}. This event prepares G_{i-1} to receive a part from G_i. From state 3, G_i can return a part to G_{i-1} by executing b_{i-1} or it can send the piece back to state 2 by local event re. A processed part in state 4 will be sent to the next subprocess by event f_i unless G_i receives handshaking event r_i from G_{i+1}.

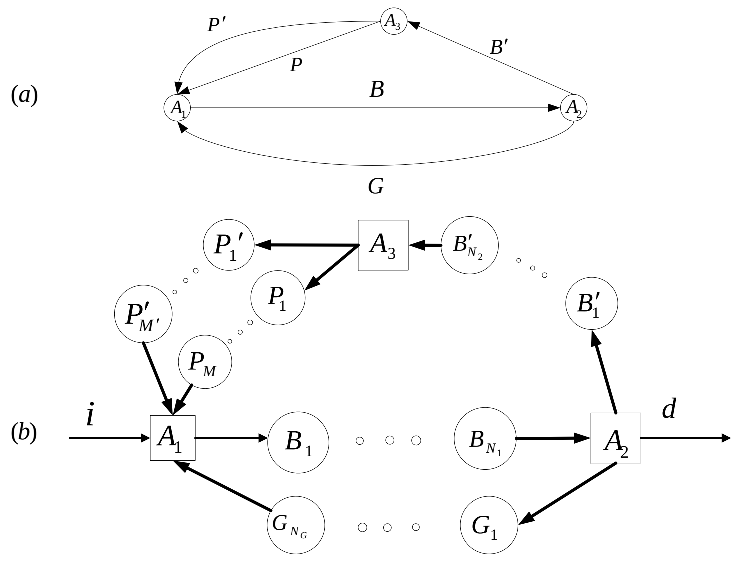

Figure 4.4: (a) Graph representation of large-scale factory example. B and B' are buffers and G, P and P' are production lines. Arcs indicate a linear parameterized section and nodes indicate a distinguished subprocess. A_1 is an input node; A_2 and A_3 are output nodes. (b) Subprocesses of large-scale factory GPDES. M, M', N_G are parameters of production lines P, P' and G; N_1 and N_2 are parameters of buffer B and B' respectively.

Subprocesses of buffers B and B', have similar structure that is depicted in Figure 4.6(b) and 4.6(c). We assume buffer B and B' have N_1 and N_2 buffer cells respectively. The i_{th} subprocess of B (B') receives a part from $i-1_{th}$ subprocess of B (B') by executing event s_{i-1} (s'_{i-1}) and sends a part to the next subprocess by executing event s_i (s'_i).

The distinguished subprocess A_2 decides what to do with the workpieces: it receives a part from the buffer B (event s_{N_1}), and examines the part. It can send the workpiece out of the factory by execution of local event d. It can also send the workpiece to production line G by event f_0, or to buffer B' by event s'_0. Subprocess G_1 can return a part to A_2 by performing event b_0.

The distinguished subprocess A_3 receives workpieces from buffer B' by event s'_{N_2} and sends the workpiece to production line P and P' by events c_0 and c'_0. Subprocess A_3 can engage in the workpiece processing with P and P' by executing events d_0 and d'_0 respectively. Event h_0 is a handshaking signal between P'_1 and A_3, indicating that P'_1 have successfully sent the workpiece to P'_2.

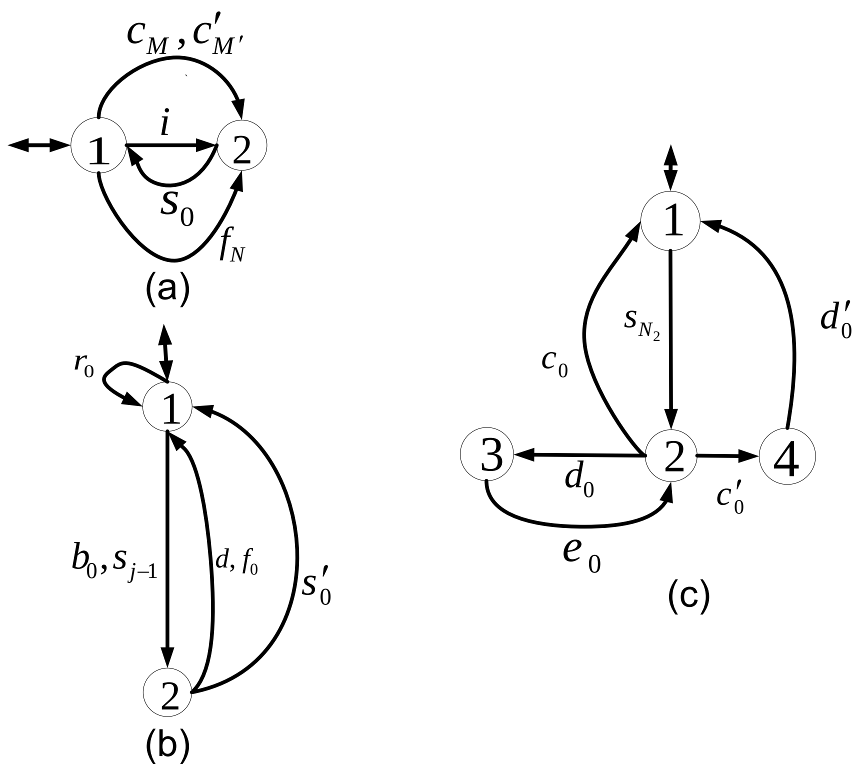

Figure 4.5: The structure of distinguished subprocesses of the large-scale factory example. (a) Subprocess A_1. (b) Subprocess A_2. (c) Subprocess A_3.

In production line P, a subprocess P_i, $1 < i < M$, receives a workpiece from neighbor P_{i-1} by event c_{i-1}. Then it decides between two actions: it can salvage the workpiece for its material by executing local event sal, or it can process the workpiece together with P_{i-1} by executing shared event d_{i-1}. In this case subprocess P_i sends the processed workpiece to P_{i+1} by event c_i and informs P_{i-1} that the workpiece was sent by performing h_{i-1}.

In production line P', a subprocess P'_i, $1 < i < M'$, receives a workpiece from neighbor P'_{i-1} by event c'_{i-1}. Then it processes the workpiece together with P'_{i-1} by executing shared event d_{i-1}. From state 1, P_i can also produce its own workpiece by local event pr, then send the workpiece to subprocess P'_{i+1} and process it together with that subprocess (events c_i and d_i).

It is easy to check that all three cycles in the graph representation of the network satisfy (4.4-4.6) and the distinguished subprocesses satisfy (4.7-4.9). Since the network described above satisfies the assumptions of our proposed framework, we can use the results of Theo-

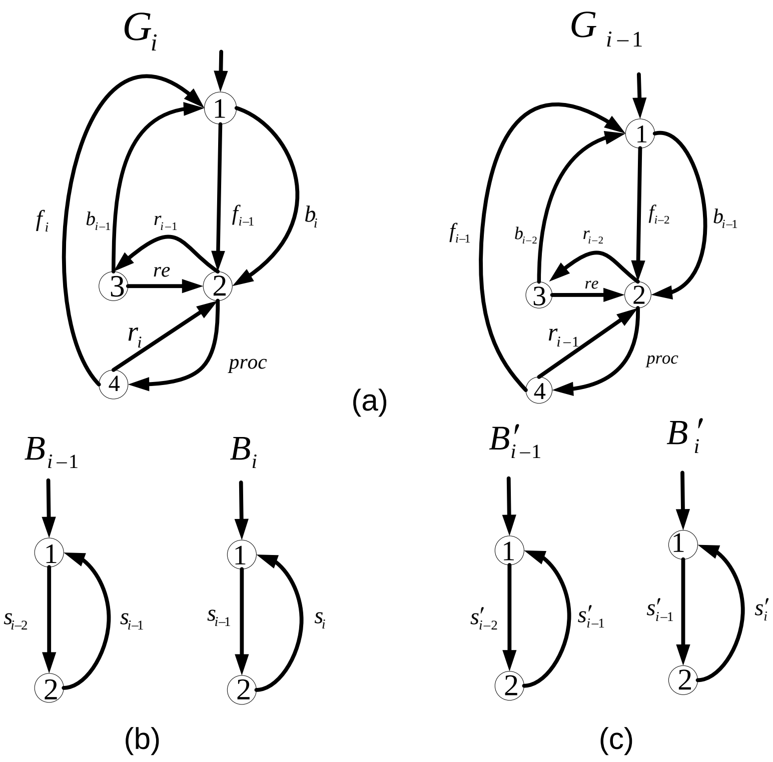

Figure 4.6: The structure of $i-1_{th}$ and i_{th} subprocesses of (a) production line G, (b) buffer B, (c) Buffer B'.

rem 5 for deadlock analysis. The first step for deadlock analysis is to build the dependency graph by application of Algorithm 1. The dependency graph of for the GPDES network of this example is given in Figure 4.8.(a). Full and consistent subgraphs of dependency graph $\mathscr{D}$ are $\mathscr{D}_1$ and $\mathscr{D}_2$, depicted in Figure 4.8 part (b) and (c), respectively. Both of these subgraphs are output reachable.

Subgraph $\mathscr{D}_1$ represents the following state set: subprocess A_1 is in state 1, subprocesses of buffer B are in state 2, subprocess A_2 is in state 2, subprocesses of buffer B' are in state 2, A_3 is in state 2 , subprocesses of P' are in state 2. According to $\mathscr{D}_1$, first subprocess of P, namely P_1, is in state 5 and the next subprocess of P, P_2, is in state 3. Similarly for all the P_i, $1 \leq i \leq M$, if i is an odd number, then P_i is in state 5 and if i is even, then P_i is in state 3. Since in $\mathscr{D}_1$, state 3 of P is connected to state 2 of A_1, the last subprocess of P, namely P_M, is in state 3. In other words, the represented state set is for the case that production line P has an even number of subprocesses (M is even). According to Theorem

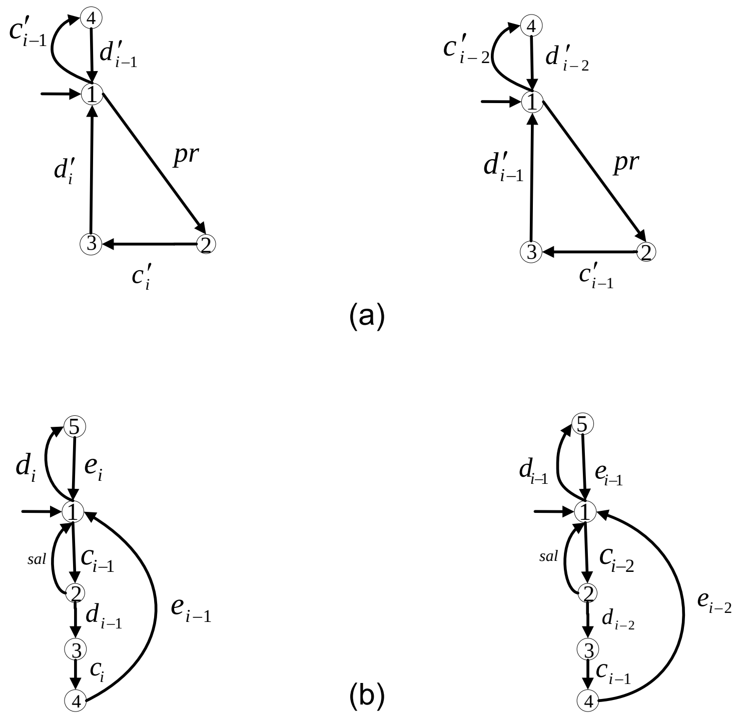

Figure 4.7: (a) The structure of $i-1_{th}$ and i_{th} subprocesses of (a) production line P and (b) production line P'.

5, this is a reachable local deadlocked state set. Similarly, subgraph $\mathscr{D}_2$ represents another set of locally deadlocked states. Note that subgraph $\mathscr{D}_2$ represents a set of local deadlocked states for the case that M is odd: for all the P_i, $1 \leq i \leq M$, if i is an odd number, then P_i is in state 3 and if i is even, then P_i is in state 5. In $\mathscr{D}_2$, state 3 of P is connected to state 2 of A_1, therefore last subprocess of P, namely P_M, is in state 3. There are no other full, consistent subgraph of the dependency graph $\mathscr{D}$; hence there are no other generalized circular waits in the network graph.

4.4 Conclusion

The deadlock analysis of a generalized parameterized discrete event network with branching topology was addressed in this chapter. Since these networks generally contain several

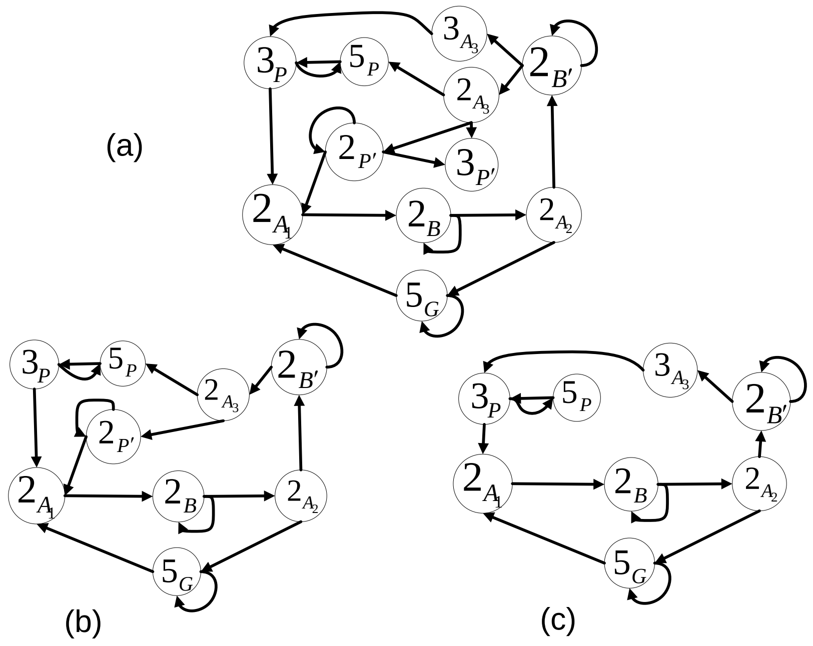

Figure 4.8: (a) The Dependency graph $\mathscr{D}$ of large-scale factory example. (b) $\mathscr{D}_1$: a full, consistent subgraphs of the dependency graph $\mathscr{D}$. (c) $\mathscr{D}_2$: another full, consistent subgraph of $\mathscr{D}$.

cycles, we initially carry out an analysis for each cycle of the network: for an arbitrary cycle in the network, we first disable events that are shared with subprocesses outside of the cycle to obtain a ring network. In the resulting ring network, we characterize the circular wait condition and reachable deadlocked state (Lemma 4). However, the resulting deadlocked states in the isolated ring are not necessarily deadlocked when we consider the interaction between the cycle with the rest of the network by enabling the disabled shared events. Therefore we developed the dependency graph to cover possible interaction scenarios between the cycle and the rest of the network, and to verify the potential occurrence of deadlock caused by generalized circular waits in a subgraph of the network graph.

Appendix

4.A Proof of Proposition 9

Proof. As mentioned earlier, only the distinguished subprocesses G_1 and G_j, $j \in J$, where J is the index set of output vertices, are affected by isolation of the cycle G^N. Let $j \in J$. By (4.6), for all $i \neq 1, j$, $\hat{G}_i \mathcal{V}_i \hat{G}_{i+1}$. But according to Proposition 8.(a), $\hat{G}_j \mathcal{V}_j \hat{G}_{j+1}$. By Proposition 8.(b) and definition weak invariant simulation, it can easily be shown that $\hat{G}_1 \mathcal{V}_2 \hat{G}_2$. Hence for all i, $1 \leq i \leq N$,

$$\hat{G}_i \mathcal{V}_i \hat{G}_{i+1} \tag{4.23}$$

By assumption, $x_d \in X_d$; so we have $(x_{d_{k-1}}, x_{d_k}) \in R_k$, where R_k is the state set of synchronous product $G_{k-1} \| G_k$. Therefore, by assumption (4.6) and Proposition 7, $(x_{d_{k-1}}, x_{d_k}) \in \mathcal{V}_k$; but x_k and x_{d_k} are the same states, hence

$$(x_{d_{k-1}}, x_k) \in \mathcal{V}_k. \tag{4.24}$$

Then again, x is a reachable state in $\hat{G}^N$, so $(x_{k-1}, x_k) \in \hat{R}_k$, where $\hat{R}_k$ is the state set of synchronous product $\hat{G}_{k-1} \| \hat{G}_k$. Therefore, by (4.23) and Proposition 7, $(x_{k-1}, x_k) \in \mathcal{V}_k$. By assumption (4.4) of the network and the definition of X_d, there exists a shared event $\beta_{k-1} \in \Sigma_{k-1} \cap \Sigma_k$ that is reachable in $\hat{G}_k$ from $x_k = x_{d_k}$ via a string in $(\Sigma_k \setminus \Sigma_{k-1})^*$. The pair $(x_{d_{k-1}}, x_{d_k})$ satisfies the forward dependency property (4.15), therefore any transition defined from $x_{d_{k-1}}$ in $\hat{G}_{k-1}$ is shared with $\hat{G}_k$. Consequently by (4.24) and the definition of weak invariant simulation, we have

$$\hat{\xi}_{k-1}(x_{d_{k-1}}, \beta_{k-1}) \neq \emptyset. \tag{4.25}$$

On the other hand, because β_{k-1} is accessible from x_k via a string in $(\Sigma_k \setminus \Sigma_{k-1})^*$, and $(x_{k-1}, x_k) \in \mathcal{V}_k$, by the definition of weak invariant simulation there exists a string

$\hat{l}_{k-1} \in (\Sigma_{k-1} \setminus \Sigma_k)^*$ and a state $\tilde{x}_{k-1} \in \chi_{k-1}(\beta_{k-1})$ such that

$$\tilde{x}_{k-1} \in \hat{\xi}_{k-1}(x_{k-1}, \hat{l}_{k-1})$$

Therefore by assumption (4.5) of the network and (4.25), there exists a $l_{k-1} \in (\Sigma_{k-1} \setminus \Sigma_k)^*$ such that

$$x_{d_{k-1}} \in \hat{\xi}_{k-1}(x_{k-1}, l_{k-1}).$$

However, such an l_{k-1} may contain events shared with $\hat{G}_{k-2}$. According to Lemma 3, for any i, $1 < i \leq N$, all these shared events can be executed in the $\hat{G}^N$ by strings with empty projection into $\bigcup_{r=k}^{N} \Sigma_r$. By repeating this argument, it can be shown that there exists a global string $l \in (\Sigma \setminus (\bigcup_{r=k-1}^{N} \Sigma_r))^*$ such that $P_{\Sigma_{k-1}}(l) = l_{k-1}$ and l_{k-1} can be executed in G_{k-1} within $\hat{G}^N$. □

4.B Proof of Proposition 10

Proof. (a) By assumption $x \in X$ represented by a cycle in a consistent subgraph of the dependency graph $\mathscr{D}$. Therefore consider an arc (x_j, x_{j+1}), $j \in J$ in the dependency graph. By the definition of forward dependence, (x_j, x_{j+1}) is reachable in the synchronous product of $\hat{G}_j \| \hat{G}_{j+1}$, where $\hat{G}_j$ and $\hat{G}_{j+1}$ are the isolated versions of G_j and G_{j+1} in isolated cycle $\hat{G}^N$. Therefore by (4.9) and the definition of weak invariant simulation,

$$(x_j, x_{j+1}) \in \mathcal{Q}_{j+1} \tag{4.26}$$

where $\mathcal{Q}_{j+1}$ is the weak invariant simulation of G_{j+1} by G_j w.r.t. $\Sigma_{S_j} \setminus \Sigma_{j-1}$.

(b) By (4.4) and the fact that x is a forward dependent state, there exists a string $l_{j+1} \in \Sigma_{j+1}^*$ such that $\xi_j(x_{j+1}, l_{j+1}) \neq \emptyset$ and $P_{\Sigma_j}(l_{j+1}) \neq \epsilon$. Therefore by (4.26), there must exists a string $l_j \in \Sigma_j^*$ such that $P_{\Sigma_{S_j} \setminus \Sigma_{j-1}}(l_{j+1}) = P_{\Sigma_{S_j} \setminus \Sigma_{j-1}}(l_j)$. By definition of forward dependence, there is no local event defined from x_j, therefore the first event of l_j is in $\Sigma_j \cap \Sigma_{j+1}$.

(c) Note that only input and output subprocesses are affected by isolation of a cycle. Therefore, we only have to show the reachability of x_1 and x_j, $j \in J$. For x_j, by part (b), there exists an event $\sigma \in \Sigma_j \cap \Sigma_{j+1}$ such that $\hat{\xi}_j(x_j, \sigma_j) \neq \emptyset$. therefore consider string $k_{j+1} \in \Sigma_{j+1}^*$ such that $\xi_{j+1}(x_{j+1}^0, k_{j+1}\sigma) \neq \emptyset$. Then by (4.26) there exists $k_j \in \Sigma_{j+1}^*$ such that $\xi_j(x_j^0, l_j\sigma) \neq \emptyset$ and $P_{\Sigma_{S_j} \setminus \Sigma_{j-1}}(k_{j+1}) = P_{\Sigma_{S_j} \setminus \Sigma_{j-1}}(k_j)$. Therefore k_j contains no event shared with the rest of direct successors of G_j in the network graph. By (4.5), the companion state if σ is unique in G_1, therefore $x_j \in \xi_j(x_j^0, l_j)$ and hence reachable in the isolated cycle. By (4.8) reachability of x_1 can be shown similarly.

□

Chapter 5

Conclusion and Future Work

Networks with arbitrarily large numbers of isomorphic subprocesses appear in areas such as computer software and hardware, transportation networks and manufacturing systems. Parameterized discrete event systems (PDES) provide a framework for modeling these networks. This modeling is specifically useful when the number of subprocesses is arbitrary, unknown or time-varying. Unfortunately, key problems such as checking the nonblocking property in these networks are undecidable. Moreover mathematical tools supporting analysis of these networks are very limited. In Chapter 2, we first introduce a novel mathematical notion, *weak invariant simulation*, which is adapted to the analysis of synchronous products of nondeterministic discrete event systems. Then we compare weak invariant simulation to other simulation relations in the literature. Moreover, we propose an efficient method to check whether a process invariantly weakly simulates another process with respect to a specific subalphabet. The greatest lower bound of all weak invariant simulations between two processes is also introduced.

To deal with undecidability results, in Chapter 3 we restrict the model of each subsystem in the network as well as communication between subsystems to seek a decidable model. In particular, we consider only networks with a ring topology – processes are arranged in a ring, and interact directly only with their immediate neighbors in the ring. Blocking in such networks is still undecidable [33], but here we introduce assumptions on the structure of processes that render analysis more tractable. Specifically, our assumptions ensure that, while both immediate neighbors may prevent a process from executing shared events, only one neighbor can permanently prevent an event from occurring. We utilize weak invariant simulation to define a tractable subclass of parameterized ring networks of isomorphic subprocesses in which deadlock-freedom is decidable. Within this framework, we give an efficient procedure to determine all the reachable deadlocked states of the ring network.

In Chapter 4, we consider a network consisting of several linear parameterized sections but exhibiting a branching topology. To model these networks we introduce Generalized Parameterized Discrete Event Systems. The difficulty in analysis of a GPDES is the fact that some of the subprocesses interact with several parameterized sections of the network. Hence the analysis proposed in this chapter involves careful study of interaction among different branches of the network. Since the general problem is undecidable, we use our previously developed mathematical notion 'weak invariant simulation' to limit the behavior of subprocesses of the network. Then we investigate interactions among different components of the network, using a dependency graph. The dependency graph is a directed graph developed to characterize reachable partial deadlocks caused by generalized circular waits in the proposed GPDES. Our results implicitly characterize reachable generalized circular waits as a language accepted by a finite automaton. Our framework allows for modeling and analysis of new parameterized problems. We investigated deadlock in a large-scale factory as an illustrative example.

Formal verification aims to answer the question of whether a system satisfies a specification. Most developed model checking algorithms perform an exhaustive search of the state-space of the system to determine satisfaction of the specification. Therefore they are only applicable to finite-state systems. Although each instance of a parameterized system is finite, any parameterized system satisfies a specification if all possible instances of the parameterized system satisfies that specification. This means that in general, checking satisfaction of a property in a parameterized system involves exploring an infinite state-space.

Exploration of an infinite state-space is possible by implicit representation for sets of states. Verification of systems using a 'symbolic' representation of their states is known as symbolic model checking [29]. The application of symbolic model checking to infinite systems is possible, however the termination of such procedures is not guaranteed. Regular model checking is a form of symbolic model checking where regular expression are used as the symbolic representation for state sets [6]. There has been a significant effort to extend the applicability of regular model checking to parameterized and infinite-state systems. These approaches are mainly based on using abstractions and ad-hoc decision procedures [7].

Our work is related to regular model-checking, insofar as it characterizes (albeit implicitly) the set of reachable deadlocked states as a language accepted by a finite automaton. In the case of parameterized ring networks (Chapter 3), the automaton runs on strings over the alphabet of subprocess state symbols; accepted strings correspond to circular waits. In the case of branching topologies (Chapter 4), the automaton is an automaton on finite trees, where the nodes of the tree are labeled by subprocess state symbols and accepted trees represent full, consistent output-reachable subgraphs of the dependency graph.

These results are particularly useful in light of limitations of previous efforts. (Note for instance that though [14] addresses the deadlock analysis of parameterized networks, its modeling framework does not admit treatment of the example of Figure 3.1.)

Nonblocking supervision of networks can guarantee satisfaction of control objectives as well as liveness of the network. For a specific network parameter, i.e specific number of subsystems, nonblocking supervision of the network is possible using a 'monolithic' synthesis approach; that is building a global network model and designing a centralized supervisor. However, this method has two major drawbacks. First, the computational complexity of monolithic supervisor design increases exponentially with number of the network subsystems (network parameter). Therefore the synthesis procedure may be impractical for large network parameters. Second, the whole procedure should be repeated if the network parameter changes.

In [3], deadlock freedom of a parametrized system synthesized from a nonblocking pair system is investigated. This is done using a technical assumption called a 'wait-for-graph assumption', which roughly guarantees that circular waiting chains cannot form. Using this result requires a symmetric nonblocking supervisor design for each subsystem in the pair system. However, there is no algorithm for designing such supervisor. As a matter of fact, automatic design of such a supervisor is still undecidable [44].

Synthesizing a scalable nonblocking supervisor method based on the template model is a challenge. This is a long-term goal. First one needs to design a procedure for the blocking analysis of parameterized networks based on calculated deadlocked states. Then based on the result of blocking analysis, a nonblocking supervisor has to be designed to prevent transitions to blocking states. Unfortunately the supervisor itself induces blocking by disabling some transitions. The typical solution to this problem is designing a possibly blocking supervisor and finding the blocking states of the supervised system, and iterating the two aforementioned procedures until the supervised system is nonblocking.

References

[1] K R Apt and D C Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22:307–309, May 1986.

[2] Tamarah Arons, Amir Pnueli, Sitvanit Ruah, Ying Xu, and Lenore Zuck. Parameterized verification with automatically computed inductive assertions? In *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 221–234. Springer Berlin / Heidelberg, 2001.

[3] Paul C. Attie and E. Allen Emerson. Synthesis of concurrent systems with many similar processes. *ACM Trans. Program. Lang. Syst.*, 20:51–115, January 1998.

[4] J. C. M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra: Equational Theories of Communicating Processes.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[5] Hans Bherer, Jules Desharnais, and Richard St-Denis. Control of parameterized discrete event systems. *Discrete Event Dynamic Systems*, 19:213–265, 2009.

[6] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *Computer Aided Verification*, pages 403–418. Springer, 2000.

[7] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 10 20 states and beyond. In *Logic in Computer Science, 1990. LICS'90, Proceedings., Fifth Annual IEEE Symposium on e*, pages 428–439. IEEE, 1990.

[8] K. Mani Chandy, Jayadev Misra, and Laura M. Haas. Distributed deadlock detection. *ACM Trans. Comput. Syst.*, 1:144–156, May 1983.

[9] Gary Chartrand, Linda Lesniak, and Ping Zhang. *Graphs & digraphs.* CRC Press, 2010.

[10] Edmund Clarke, Muralidhar Talupur, Tayssir Touili, Helmut Veith, and Technische Universitï£ · t Munchen. Verification by network decomposition. In *In 15th Concur, LNCS 3170*, pages 276–291. Springer, 2004.

[11] Ahmed K Elmagarmid. A survey of distributed deadlock detection algorithms. *ACM Sigmod Record*, 15(3):37–45, 1986.

[12] E. Emerson and Vineet Kahlon. Model checking large-scale and parameterized resource allocation systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 55–69. Springer Berlin / Heidelberg, 2002.

[13] E. Allen Emerson and Vineet Kahlon. Reducing model checking of the many to the few. In *17th International Conference on Automated Deduction*, pages 236–255, 2000.

[14] E. Allen Emerson and Vineet Kahlon. Model checking large-scale and parameterized resource allocation systems. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '02, pages 251–265, London, UK, 2002. Springer-Verlag.

[15] E Allen Emerson and Vineet Kahlon. Parameterized model checking of ring-based message passing systems. In *Computer Science Logic*, pages 325–339. Springer, 2004.

[16] E. Allen Emerson and Kedar S. Namjoshi. On reasoning about rings. *International Journal of Foundations of Computer Science*, 14(04):527–549, 2003.

[17] M.P. Fanti and MengChu Zhou. Deadlock control methods in automated manufacturing systems. *Systems, Man and Cybernetics, Part A: IEEE Transactions on Systems and Humans*, 34(1):5 – 22, 2004.

[18] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[19] R.J. van Glabbeek. The linear time – branching time spectrum (extended abstract). In J.C.M. Baeten and J.W. Klop, editors, Proceedings *CONCUR '90, Theories of Concurrency: Unification and Extension,* Amsterdam, August 1990, volume 458, pages 278–297, 1990.

[20] NejibBen Hadj-Alouane, StÃl'phane Lafortune, and Feng Lin. Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation. *Discrete Event Dynamic Systems*, 6:379–427, 1996.

[21] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 26:100–106, 1983.

[22] J. Komenda, J.H. van Schuppen, B. Gaudin, and H. Marchand. Supervisory control of modular systems with global specification languages. *Automatica*, 44(4):1127 – 1134, 2008.

[23] Sava Krstic. Parametrized system verification with guard strengthening and parameter abstraction. In *Proceedings of 4th Workshop on Automated Verification of Infinite-State Systems*, AVIS '05, 2005.

[24] Leslie Lamport. The synchronization of independent processes. *Acta Informatica*, 7:15–34, 1976.

[25] Yongjian Li. Mechanized proofs for the parameter abstraction and guard strengthening principle in parameterized verification of cache coherence protocols. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07, pages 1534–1535, New York, NY, USA, 2007. ACM.

[26] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44(3):199–224, 1988.

[27] A. Mannani and P. Gohari. Decentralized supervisory control of discrete-event systems over communication networks. *IEEE Transactions on Automatic Control*, 53(2):547 –559, march 2008.

[28] Charles E. McDowell. A practical algorithm for static analysis of parallel programs. *Journal of Parallel and Distributed Computing*, 6(3):515–536, 1989.

[29] Kenneth L McMillan. *Symbolic model checking*. Springer, 1993.

[30] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[31] S. Mohajerani, R. Malik, and M. Fabian. A framework for compositional synthesis of modular nonblocking supervisors. *Automatic Control, IEEE Transactions on*, 59(1):150–162, Jan 2014.

[32] S. Nazari and J.G. Thistle. Blocking in fully connected networks of arbitrary size. *IEEE Transactions on Automatic Control*, 57(5):1233 –1242, May 2012.

[33] Siamak Nazari. *Analysis of Parameterized Networks.* PhD Thesis, Electrical and Computer Engineering, University of Waterloo, 2008.

[34] P. Ramadge and W. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25:206–230, January 1987.

[35] Peter Ramadge and W. Murray Wonham. Control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[36] I. Romanovski and P.E. Caines. On the supervisory control of multiagent product systems. *IEEE Transactions on Automatic Control*, 51(5):794–799, may 2006.

[37] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40:1555–1575, 1995.

[38] Davide Sangiorgi. *An introduction to Bisimulation and Coinduction.* Cambridge University Press, 2011.

[39] K.W. Schmidt and J.E.R. Cury. Efficient abstractions for the supervisory control of modular discrete event systems. *Automatic Control, IEEE Transactions on*, 57(12):3224–3229, 2012.

[40] Antti Siirtola and Juha Kortelainen. Algorithmic verification with multiple and nested parameters. In *Formal Methods and Software Engineering*, volume 5885 of *Lecture Notes in Computer Science*, pages 561–580. Springer Berlin / Heidelberg, 2009.

[41] Ichiro Suzuki. Proving properties of a ring of finite-state machines. *Inf. Process. Lett.*, 28:213–214, July 1988.

[42] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[43] J. G. Thistle. Supervisory control of discrete event systems. *Mathematical and Computer Modelling*, 23(11-12):25–53, 1996.

[44] J.G. Thistle. Undecidability in decentralized supervision. *Systems and Control Letters*, 54(5):503–509, 2005.

[45] Rob J. van Glabbeek, Bas Luttik, and Nikola Trcka. Computation tree logic with deadlock detection. *Logical Methods in Computer Science*, 5(4), 2009.

[46] P. Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, feb 1993.

[47] N. Viswanadham, Y. Narahari, and T.L. Johnson. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using petri net models. *Robotics and Automation, IEEE Transactions on*, 6(6):713 –723, 1990.

[48] K.C. Wong, J.G. Thistle, R.P. Malhame, and H.-H. Hoang. Supervisory control of distributed systems: Conflict resolution. *Discrete Event Dynamic Systems*, 10:131–186, 2000.

[49] W. Wonham and P. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems (MCSS)*, 1:13–30, 1988.

[50] W. M. Wonham. Lecture notes on supervisory control of discrete event systems, 2012. Available at http://www.control.utoronto.ca/cgi-bin/dldes.cgi, [Jan. 30, 2013].

[51] W.M. Wonham and B.A. Brandin. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–342, 1994.

[52] W.M. Wonham and K.C.Wong. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 6(3):241–273, 1996.

[53] W.M. Wonham and F. Lin. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988.

[54] W.M. Wonham and P.J.Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25(3):637–659, 1987.

[55] M. H. Zibaeenejad and J. G. Thistle. Dependency graph: an algorithm for analysis of generalized parameterized networks. Submitted to American Control Conference (ACC), 2015.

[56] M. H. Zibaeenejad and J. G. Thistle. A framework for analysis of generalized parameterized networks. Submitted as journal publication.

[57] M. H. Zibaeenejad and J. G. Thistle. Deadlock analysis of generalized parameterized discrete event systems with ring topology. pages 370–375, May 2014. Proceedings of 12th IFAC - IEEE International Workshop on Discrete Event Systems.

[58] M. H. Zibaeenejad and J. G. Thistle. Weak invariant simulation and its application to analysis of parameterized networks. *Automatic Control, IEEE Transactions on*, 59(8):2024–2037, Aug 2014.

[59] M.H. Zibaeenejad and J.G. Thistle. Invariant weak simulation and analysis of parameterized networks. In *American Control Conference (ACC), 2012*, pages 6108–6113, 2012.

[60] M.H. Zibaeenejad and J.G. Thistle. Weak invariant simulation: Properties and algorithms. In *American Control Conference (ACC), 2013*, pages 911–916, June 2013.