HAL
open science

Abstraction and modular verification of inter-enterprise business processes

Hanen Ochi

To cite this version:

Hanen Ochi. Abstraction and modular verification of inter-enterprise business processes. Systems and Control [cs.SY]. Université Sorbonne Paris Cité, 2015. English. NNT : 2015USPCD064 . tel-01733337

HAL Id: tel-01733337

https://theses.hal.science/tel-01733337

Submitted on 14 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ PARIS 13, SORBONNE PARIS CITÉ

ÉCOLE DOCTORALE GALILÉ

THÈSE

Présentée par

Hanen Ochi

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

SPÉCIALITÉ : INFORMATIQUE

Abstraction and Modular Verification of Inter-Enterprise Business Processes

Membre du jury :

Directeur de thèse :	
Kais KLAI	LIPN, Université Paris 13
Rapporteurs :	
Daniela GRIGORI	LAMSADE, Université Paris Dauphine
Farouk TOUMANI	LIMOS, Université Blaise Pascale
Examinateurs :	
Walid GAALOUL	SAMOVAR, TELECOM SudParis
Carine SOUVEYET	CRI, Université Paris 1
Laure PETRUCCI	LIPN, Université Paris 13
Pascal POIZAT	LIP6, Université Paris 10

Abstract:

Today's corporations often must operate across organizational boundaries. Phenomena such as electronic commerce, extended enterprises, and the Internet stimulate cooperation between organizations. We propose a bottom-up approach to check the correct interaction between business processes distributed over a number of organizations. The whole system's model being unavailable, an up-down analysis approach is simply not feasible. We consider two correctness criteria of Inter-Enterprise Business Processes (IEBP) composed by two (or more) business processes communicating either synchronously or asynchronously and sharing resources: a generic one expressed with the well known soundness property (and some of its variants), and a specific one expressed with any linear time temporal logic formula. Each part of the whole organization exposes its abstract model, represented by a Symbolic Observation Graph (SOG), in order to allow the collaboration with possible partners. We revisited and adapted the SOG in order to reduce the verification of the entire composite model to the verification of the composition of the SOG-based abstractions. We implemented our verification algorithms, aiming at checking both specific and generic properties using SOGs, and compared our approach to some well known verification tools. The experimental results are encouraging in terms of both the construction time and the size of the abstraction's size. This strengthen our belief that the SOGs are suitable to abstract and to compose business processes especially when these are loosely coupled.

Keywords: Inter-Enterprise Business Processes, Symbolic Observation Graph, Soundness, LTL, Abstraction, Formal Verification.

Résumé :

De nos jours, les entreprises sont de plus en plus étendues et faisant collaborer plusieurs organisations pour la réalisation composée d'un objectif global. Des phénomènes tels que le commerce électronique et l'Internet stimulent en effet la coopération entre les organisations, donnant lieu a des processus métier inter-entreprises. Dans cette thèse de doctorat, nous proposons une approche ascendante pour vérifier l'interaction correcte entre des processus répartis sur un certain nombre d'organisations. Le modèle du système global étant indisponible, une approche d'analyse descendante est tout simplement impossible. Nous considérons deux critères de correction des processus métier inter-entreprises composés de deux (ou plusieurs) processus métier qui communiquent de manière synchrone et/ou asynchrone et/ou partageant des ressources. Le premier critère est générique et est exprimé par la propriété de *soundness* (robustesse), et certaines de ses variantes. Le deuxième critére est spécifique et peut être exprimé avec n'importe quelle formule de la logique temporelle linéaire. Chaque composante du processus global rend publique un modèle abstrait, représenté par un graphe appelé Graphe d'Observation Symbolique (GOS), permettant à la fois de présever la confidentialité du processus local, la vérification de sa correction et de celle du processus global par composition de GOSs. Nous avons revisité et adapté le GOS afin de réduire la vérification du modèle composite à la vérification de la composition des abstractions des ses composants (leurs GOSs).

Nous avons implémenté notre approche de vérification, basée sur le GOS, aussi bien pour les propriétés génériques que pour les propriétés spécifiques (LTL), et nous avons comparé les résultats obtenus avec ceux d'outils connus dans le domaine. Les résultats obtenus sont encourageants au vu du temps d'exécution et de l'espace mémoire consommés par notre technique. Ceci renforce notre conviction que le GOS est une structure appropriée pour l'abstraction et la vérification de processus métiers, en particulier lorsque ceux-ci sont faiblement couplés.

Mots clés: Processus Metier Inter-entreprises, Graphe d'Observation Symbolique, Soundness, LTL, Abstraction, Vérification formelle.

Acknowledgments

I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family and husband.

First and foremost, I would like to express my sincere gratitude to my advisor Kais KLAI for his continuous support, his patience, motivation, and his immense knowledge. I also thank him for the many insights about formal methods I learnt from him and even for the sleepless nights we were working together before deadlines. His guidance helped me during PhD and during the manuscript writing phase as well. I could not have imagined having a better advisor and mentor for my Ph.D.

I sincerely thank the reviewers and the jury members: Prof. Daniela Grigori and Prof. Farouk Toumani for accepting reviewing this manuscript and for their comments on my work; and Prof. Walid Gaaloul, Prof. Carine Souveyet, Prof. Laure Petrucci and Prof. Pascal Poizat for the time they have kindly employed for me.

My sincere thanks also goes to all the members of Laboratoire d'Informatique Paris Nord (LIPN), who provided me the opportunity to join their group as research engineer first and as Ph.D student later, and who gave me access to the laboratory and research facilities. Without their precious support it would not be possible to conduct this research.

Many people from LIPN contributed to making these last years a wonderful experience. I was very lucky to count on many friends to make these years enriching and worth remembering. They know who they are, and I want to specially thank all of them. Here it goes an incomplete list: Aicha, Amine, Ehab, Hanan, Ines, Issam, Leila, Naim, Nouha, Manisha, Moufida, Sarah, Mohamed, Paolo, Rakia, Zayd. Thanks for opening your hearts and letting me be part of your lives.

I would like to express my warmest appreciation to all my family, specially my dear mother Rebha for supporting me spiritually with all her best wishes throughout this thesis and during all my life in general. I particularly thank also my brother, my sisters and all my husband's family for being a constant source of encouragement, inspiration, and love all across this period.

My love and gratitude goes first to my husband Tahar for all the uncountably many ways in which he supported me during these years. He was always there cheering me up and stood by me through the good and the bad times. Second to my daughter Lina for remembering me every day how wonderful life is.

Last but not the least, I would like to dedicate this work to my father Ridha OCHI who passed away recently. I owe a great deal to him: I will always be thankful and grateful to him for showing me that the key to life is accepting challenges. I feel that he is looking at this work from the Heaven.

Contents

1 General Overview **1**
1.1 Scientific Context and issues . . . 1
1.2 Objectives and Contributions . . . 5
1.3 Organization of the Thesis . . . 7

2 Preliminaries **9**
2.1 Introduction . . . 9
2.2 Formal Models for BPs . . . 10
2.2.1 WorkFlow-nets . . . 11
2.2.2 Open WorkFlow net . . . 12
2.2.3 Resource-Constraint Workflow Nets . . . 14
2.2.4 Resource-Constraint open WorkFlow Nets . . . 14
2.3 Composition of RCoWF-nets . . . 16
2.4 Representation of the Reachable Configurations . . . 16
2.4.1 Labeled transition Systems . . . 17
2.4.2 Kriple Structure . . . 17
2.4.3 Labeled Kriple Structure . . . 18
2.4.4 Binary Decision Diagrams . . . 18
2.4.5 The Event- and State-based SOGs . . . 20
2.5 Behavioral Properties of IEBP . . . 22
2.5.1 Soundness Properties . . . 23
2.5.2 Linear Temporal Logic propreties . . . 24
2.6 Conclusion . . . 25

3 Related Work **27**
3.1 Introduction . . . 27
3.2 Formal Verification Approaches . . . 28
3.2.1 Theorem Proving . . . 28
3.2.2 Model Checking Approaches . . . 29
3.3 Related BP Verification Approaches . . . 30
3.4 Related IEBP Verification Approaches . . . 31
3.5 Conclusion . . . 35

4 Using SOGs for flat verification **37**
4.1 Introduction . . . 37
4.2 Using SOGs for Hybrid LTL . . . 38
4.2.1 Revisiting SOG for Hybrid LTL . . . 39
4.2.2 SOG-based Hybrid LTL Verification Approach . . . 44
4.3 Using SOGs for Checking Generic Properties . . . 50
4.3.1 Soundness . . . 51
4.3.2 Relaxed, Weak and Easy soundness . . . 61
4.4 Conclusion . . . 62

5 Using SOGs for Modular verification **63**
5.1 Introduction 63
5.2 Composition of SOGs 64
5.2.1 The observed behavior 64
5.2.2 Synchronous composition of SOGs 68
5.2.3 Composition of RCoWF-nets' SOGs 74
5.3 Modular verification 81
5.3.1 LTL-based Properties 81
5.3.2 Checking Soundness Properties 83
5.3.3 Soundness 83
5.3.4 Relaxed, Weak and Easy Soundness 86
5.4 Conclusion 88

6 Implementation and Experimental Results **89**
6.1 Introduction 89
6.2 Verification of Soundness Properties 91
6.3 Verification of LTL Property 95
6.3.1 Implementation 95
6.3.2 Experimental results 95
6.4 Conclusion 102

7 General Conclusion and Perspectives **103**
7.1 Summary 103
7.2 Future Work 105

List of Figures

2.1 An example of Petri Net 11
2.2 An example of an WF-Net 12
2.3 An example of an oWF-net 13
2.4 An example of RCWF-Net 14
2.5 an example of RCoWF-net 15
2.6 example of a BDD 20
2.7 An *LTS* and its SOG 21
2.8 A Kripke structure and its SOG 22

4.1 An LKS and its SOG 41
4.2 A SOG and its corresponding ESOG 46
4.3 Terminal cycles preventing the detection of the option to complete requirement 51
4.4 examples of BDDs 58
4.5 Illustration of two examples of Algorithm 4's execution 59

5.1 The WF-nets of of a trip reservation and a costumer 69
5.2 Two SOGs of the running example models 71
5.3 the SOG synchronized product 73
5.4 Two RCoWF-nets sharing resources and communicating asynchronously . 75
5.5 SOGs of the running example 76
5.6 Interface graph of the medium net 78
5.7 The synchronous product of the SOGs of the RCoWF-nets example . . . 80

6.1 Schema of our approach 90
6.2 Illustration of our Model Checker 96
6.3 Sharing resources in an emergency medical care service 97

List of Tables

6.1 *Experimental results: modular SOG* 93
6.2 *Experimental results: non-modular* 94
6.3 *Model checking of an unsatisfied formula* $(G(t_7 \implies F\ t_9))$ 98
6.4 *Model checking of a satisfied formula* $(G(t_7 \implies F\ (t_9 \vee t_{12} \vee leave))$ . . 98
6.5 *Experimental results: Reservation Trip* 101
6.6 *Experimental results: Producer-Consumer* 101

Chapter 1

General Overview

Contents

1.1 Scientific Context and issues 1
1.2 Objectives and Contributions 5
1.3 Organization of the Thesis 7

1.1 Scientific Context and issues

Competitive pressures are forcing organizations to increasingly integrate and automate their business operations such as order processing, procurement, claims processing, administrative procedures and the like. These operations, called business processes (BPs), are typically of long duration. BPs are governed by complex business rules and may involve coordination across many manual and automated tasks while requiring the access to several different databases and the invocation of several application systems (e.g ERP systems). Business process (BP) [31, 32, 56] is then defined as a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs. Defining such business processes and orchestrating their execution within organizations is ensured by the middleware system called business process management system (BPM for short or business process manager). During the three last decades, there has been a lot of work in developing middleware for integrating and automating enterprise business processes. Notable examples of BPM systems are SAP, Baan, PeopleSoft, Oracle, and JD Edward. Many people consider Business Process Management (BPM) to be the "next step" after the workflow wave of the nineties. Therefore, we use workflow terminology to define BPM. In [5], the author consider that the Workflow Management (WFM) is a sub-part of BPM. It seems that the main difference is that the cycle diagnostic phase BPM is not supported by WFM. The Workflow Management Coalition (WfMC) defines workflow as: "The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules."[134]. A Workflow Management System (WFMS) is defined as: "A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to

interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications." [134]. Note that both definitions emphasize the focus on enactment, i.e., the use of software to support the execution of operational processes. In the last few years, many researchers and practitioners started to realize that the traditional focus on enactment is too restrictive. As a result new terms like BPM have been coined. BPM is widely viewed as an established discipline for building, maintaining and evolving large enterprise systems on the basis of business process models [9].

The importance of BP design is reflected by the fact that BPs are a main constituent of many enterprise architecture frameworks, such as the Architecture of Integrated Information Systems [112], or Business Engineering [52]. In this context, business process modeling is considered as an integral part of enterprise modeling that provides a conceptual basis for the specification of all business procedures. It can be quite complex (a typical business process may consist of up to 100 tasks). It aids the coordination and integration of distributed resources, tasks, and individuals, the effective management of all of which is critical to sustaining organizational capabilities. Moreover, coordinating the entire process correctly and efficiently places severe demands on the organization's IT infrastructure. Many known issues can be derived from the choice of the modeling languages such as the compromise between the power of expressiveness of the modeling language and its analysis complexity. Some languages offer a rich syntax for expressing the most of activities and their relationships in the process model, while others provide more generic modeling constructions ensuring efficient analysis at design time.

In the literature, many existing variety of BP modeling languages was used to specify BP requirements, in order to support automated process verification, validation, simulation and process automation(e.g. such the Business Process Modeling Notation, BPMN [97], the Unified Modeling Language, UML [17, 96] or the Petri nets [105]). We believe that the use of a formal language for the BP specification is the only sure way to guarantee that alternative interpretations are ruled out. Such a formal language has an additional advantage which is the suitability to a formal verification of the process correctness. It is well known that performing such a verification on the specification rather than on the implementation of a BP can reduce significantly the cost of the detection/correction of undesirable behavior.

In this work, we adopt the Petri net formalism since it has the advantage of being powerful enough to both express and analyze successfully BPs. Indeed, Petri-nets offer the advantage of graphical appeal coupled with a rigorous formalism that has found tremendous use in modeling systems and processes that exhibit asynchronism, concurrency, and determinism [95, 34]. Intuitively, any process can be understood to be a collection of events, the conditions that enable these events to occur, and the conditions that are

satisfied following the completion of these events. A Petri net ideally mirrors this intuition, and explicitly separates the conditions, and the events involved in a process, and models state changes involved therein, through a simulated movement of tokens.

Using such a formal language allows for formal verification of the BP correction. However, the correction of a BP is a relative notion since it depends on the types of the properties we are interested in. These can be either *generic* or *specific*. Generic properties depend on neither the specification language nor the business domain, and express "good" features any kind of system should have (e.g each activity in the process can occur at least in one execution, the process can never reach a state where no task can be performed, etc). However, specific properties are described in terms of precise elements (states, tasks, events, etc.) of the specification language and thus require a certain expertise regarding the business domain. For instance, in a flight reservation process, one could be interested in checking that any client request will eventually (in the future) be followed by a response. The verification of both generic and specific properties has already been studied for other kinds (and specially critical) systems such as discret event, concurrent and distributed systems. They have been defined and checked formally using (e.g.) the model checking approach [107, 50, 25]. The *deadlock-freeness* is an example of generic properties expressing the absence of a state from which no action is possible, while temporal logics (e.g. LTL [86], CTL [26]) can be used to express specific properties.

Model checking is a fully automatic technique where the possible execution paths that the system could follow are explored exhaustively, and the compliance with the specification, for each of them, is checked. If the search terminates without finding any error, model checking establishes a formal argument proving that the system is correct with respect to the specification. If not, an execution path that falsifies the specification (called a counter example) is shown to the user, which is often highly valuable to fix the problem. Although the ability to supply such a counter example (in case the property is violated) and the fact that it is a fully automatic represent the strength of the model checking approach, its main weakness is the well known combinatory explosion of the system's state space. This problem refers to the computational difficulty of performing the analysis of the system behavior automatically, and is one of the main obstacles hindering the adoption of model checking in practice. The main source of the combinatory explosion problem is concurrency i.e. different actions of the system can be executed in any possible order. Concurrency is intrinsic to modular systems i.e. systems involving several components, where the size of the whole state space grows exponentially with respect to the number of components.

In the context of business processes, although many organizations still focus on the

design and implementation of their internal activities, an increasing number of companies are targeting the integration between enterprises, or so-called inter-enterprise business processes (IEBP). Successful companies must operate in a network with other organizations to leverage their strength and to compensate for their weakness. Typically, there are n business partners which are involved in one 'global' IEBP. Each of the partners has its own 'local' business process (designed separately) which is private, i.e., each component has no knowledge about the local process of the partners. However, mutual interdependencies are created and managed to drive additional value and to ensure high performance of the organization as a whole.

From formal point of view, the IEBP have mainly been studied through two related hot topics: abstraction and composition.

- Abstraction: Information about each enterprise has to be exposed (public view) for potential partners in order to select and compose different business processes automatically. However, organizations usually want to hide the trade secrets of their services (private view) but, at the same time, must publish enough information about their workflow in order to find compatible partners. Thus, the challenge is to find an abstraction that both hides the internal behavior of components in order to respect the privacy of every concerned organisation, and, at the same time, exposes enough information to allow for a possible collaboration. Having the formal verification of BPs in mind, the public view of a BP must be of two purposes: it should allow to check the desired properties locally and without the need of an exhaustive research in the original state space graph, and, it should be sufficient to check the desired properties of the composition i.e. Verifying the composition of the models' abstractions is equivalent to the verification of the original composite model. The question, what is the more suitable abstraction of a process will represent its public view, has been dealt with in the literature since many years (e.g. [21, 82, 92, 49, 91]).

- Composition consists of all activities that are required to combine and link existing workflow or BP fragments and other components to create new processes. With the need to frequently adapt (or restructure) BPs in a dynamic market, agile processes and (semi-)automatic process composition would be useful. When each component of an IEBP ignores the detailed description of its partners, their abstractions should be sufficient to decide about the correctness of the whole process. Indeed, the correct behavior of each process (analyzed independently) does not guarantee the correction of the behavior of the process obtained by composition (ie, most of the "good properties" are not preserved by composition). Automating and optimizing

this composition and the verification tasks is of high interest in research communities (eg. [98, 60, 80, 51, 36]).

In this work, we are particularly interested in the Symbolic Observation Graph (SOG for short) which is a formalism that tackled both the abstraction and the composition of BPs. Originally [55, 75], the SOG has been defined as a hybrid structure abstracting the state graph of a system, and has been used for model checking linear time properties. It is a graph whose construction is guided by the set of atomic propositions occurring in the formula to be checked (called observed atomic propositions). The nodes of a SOG are sets of states encoded symbolically (using BDDs [19]) and its edges are represented explicitly. It supports on-the fly model-checking and is equivalent to the reachability graph of the system with respect to linear time properties. In [76] [77], the authors have extended the SOG approach to IEBPs. By observing the collaborative activities/actions (those allowing the communication between partners) only, the SOG of a component can abstract/hide its internal behavior. It has been also proved that the deadlock freeness property can be checked on the composition of the SOGs (each abstracting a component of the IEBP) instead of the composition of the original state space graphs. Thus, the SOG has been presented as a BP abstraction that preserves the privacy of each partner and that allows the analysis of the behavior of the whole process (w.r.t. the deadlock freeness property [76, 77, 70]). Moreover, the fact that the size of the SOG is (in general) inversely proportional to the number of observed atomic propositions (here the collaborative actions) makes the SOG-based approach suitable to abstract and to check efficiently loosely coupled IEBPs.

The results presented in [76] [77] have been the starting point of this thesis and the basis of the related achievements.

1.2 Objectives and Contributions

Since the SOG-based approach for the abstraction and the verification of IEBPs dealt with the deadlock freeness property only, the main objective of this thesis was to extend this approach to other behavioral properties. In particular, we have considered a well known generic property of BPs : the *soundness* property [124], and specific properties expressed with the LTL (Linear Temporal Logic) logic.

- The *soundness* property guarantees the absence of livelocks, deadlocks and other anomalies that can be formulated without domain knowledge. Roughly speaking, it requires that every task of a business process model can actually occur and that it is always possible to reach a legal final state. Various variants of soundness notions

that weaken or strengthen the original definition exist in the literature (see [123] for a detailed description). In this work, we focused on three of them: First, the notion of *relaxed* soundness, introduced in [33], ensures that each activity should occur in at least one "good" execution path (a path leading from the initial to the final state of the BP). Second, weak soundness [88] allows for dead transitions (any transition that is never fired) as long as a final marking is reachable from any state. Finally, easy soundness [129] requires only that the final state is reachable from the initial state. It is obvious that soundness implies both relaxed and weak soundness which are incomparable and both imply easy soundness. Our work has consisted, for the soundness property and for its three variants, in revisiting the SOG-based approach to check these properties from both local (non modular) and modular perspectives. Thus, we have first extended these generic properties on a system represented by its SOG. Then, we proposed dedicated algorithms allowing to reduce the verification of these properties on the original state space graph to the verification on the corresponding SOG. Finally, in order to allow the verification of an IEBP by considering the composition of its constituent's SOGs only (the whole state space graph is unavailable anyway), we adapted the structure of the SOG by enriching its nodes by necessary and sufficient (locally computed) information [72].

- The second issue in our work was to check specific properties, especially those expressed with the LTL logic, on IEBPs [73, 72]. Depending on the type of the elements (atomic propositions) one uses to write an LTL formula, the LTL logic can follows either a state-based or an event-based semantics. A state-based LTL formula uses only atomic propositions representing state properties while an event-based formula uses only atomic propositions corresponding to events (actions) occurring in the system. Although, these two formalisms are interchangeable (an event can be encoded as a change in state variables, and likewise one can equip a state with different events to reflect different values of its internal variables), converting from one representation to the other is not trivial and often leads to a significant enlargement of the state space (due to the size of the formula). Knowing that the SOG dealt, in a non modular context, with event- and state-based semantics in [55] and [75] respectively, our goal was to extend this approach to deal with a mixed logic (namely hybrid LTL) where states and events can conjointly occur in an LTL formula. Also, we considered the verification of hybrid LTL properties on IEBPs from both local and modular point of views. In this way, the verification of a component (resp. the whole IEBP), w.r.t. an LTL formula, can be reduced to the verification of the corresponding SOG (resp. the composition of the component's SOGs).

Beside our interest in enlarging the class of properties one can check on IEBPs formally, we were interested in enlarging the class of IEBPs models that can be handled by our approaches. In particular, we focused on the way the different components of an IEBP communicate with each other. In the literature, three kinds of communication between the component of a modular system have been considered separately: synchronous communication, asynchronous communication and sharing of resources. Our contribution in this work has been to propose a generic model allowing to take into account all of these communication modes, while preserving the applicability of our SOG-based verification approaches [72].

Finally, we have applied our approaches to an other field where abstraction and composition are two primordial issues: the Web services composition [69, 78, 71, 74]. A web service can be viewed as a control structure describing its behavior according to an interface to communicate asynchronously and sharing resources with other services in order to reach a final state. A composite web service is a service that consists of the coordination of several conceptually autonomous but interface compatible services. For automatically composing Web services in a correct manner, information about their behaviors (an abstract model) has to be published in a repository. This abstract model must be sufficient to decide whether two, or more, services are compatible without including any additional information that can be used to disclose the privacy of these services. Although it is not easy to specify how this coordination should behave, we have focused in our work on semantic compatibility between web services. We have then defined different compatibility criteria (based on generic and/or specific properties of the composite service) and have proposed a SOG-based abstract model for each participant service to be published in the repository. Of course, the compatibility between two Web services is checked by considering their SOGs only.

Most of the contributions in this manuscript have been published in international conferences or journals [70, 69, 71, 78, 74, 73, 72].

1.3 Organization of the Thesis

This report is organized as follows: The next chapter (Chapter 2) introduces the formal foundations and the general concepts used in all the rest of the manuscript. It defines also the formal models used to represent inter-enterprise business processes. The third chapter (Chapter 3) introduces a state of the art dealing with formal modeling and verification of IEBPs. The abstraction of business processes and the local verification approach based on symbolic representations graphs are detailed in the fourth chapter (Chapter 4) . In this chapter, we define the different soundness variants on SOGs and revisit the SOG structure

in order to allow the verification of hybrid LTL formulae. The fifth chapter (Chapter 5) is dedicated to the extension of our verification approaches to the modular context of IEBPs. The question is what is the necessary and sufficient information (computed locally) to allow the verification of the whole process based on the composition of the SOGs of its constituents. In Chapter 6 , the implementation of our approaches and the obtained experimental results are presented. Finally, the general conclusion of the thesis and the perspectives are the issue of the last Chapter (Chapter 7) .

Chapter 2

Preliminaries

Contents

2.1 Introduction **9**

2.2 Formal Models for BPs **10**

2.2.1 WorkFlow-nets 11

2.2.2 Open WorkFlow net 12

2.2.3 Resource-Constraint Workflow Nets 14

2.2.4 Resource-Constraint open WorkFlow Nets 14

2.3 Composition of RCoWF-nets **16**

2.4 Representation of the Reachable Configurations **16**

2.4.1 Labeled transition Systems 17

2.4.2 Kriple Structure 17

2.4.3 Labeled Kriple Structure 18

2.4.4 Binary Decision Diagrams 18

2.4.5 The Event- and State-based SOGs 20

2.5 Behavioral Properties of IEBP **22**

2.5.1 Soundness Properties 23

2.5.2 Linear Temporal Logic propreties 24

2.6 Conclusion **25**

2.1 Introduction

The need for formal methods and software tools for describing and analyzing business processes is widely recognized. In this chapter, we present some formal models allowing to specify BPs and their composition (IEBP). Then, the formalisms describing the corresponding behavior are presented and, finally, the properties (generic and specific) we are interested in are defined formally.

2.2 Formal Models for BPs

Although our approach is not dependent of a particular modeling formalism (as long as the behavior can be described formally), we choose to illustrate it through some sub-classes of Petri nets [105]. Petri nets are a well known formalism used for modeling real-time systems. They have the advantage of being powerful enough to both express and analyze such systems and have been successfully used in the BPs domain during the recent decades. Although, in practice, the behavior of BPs is described using standard languages such as BPEL4WS or BPMN, several approaches allow to map these models to Petri nets (e.g. [59, 81]). Thus, our approach is relevant for a very broad class of modeling languages.

Before we introduce the sub-classes of Petri nets that are used to model BPs and their composition, let us first recall the syntax and semantics of Petri nets.

Syntax of Petri nets

Definition 1 *A Petri net (Place-Transition net)* $N = \langle P, T, F, W \rangle$ *consists of:*

- P *is a finite set of places and* T *a finite set of transitions with* $(P \cup T) \neq \emptyset$ *and* $P \cap T = \emptyset$*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a flow relation representing the arcs between places and transitions,*
- $W : F \to \mathbb{N}^+$ *is a mapping that assigns a positive weight to any arc.*

A place p is called an *input* (resp. *output*) place of a transition t iff there exists an arc from p to t (resp. from t to p). Each node $x \in P \cup T$ of the net has a pre-set and a post-set defined respectively as follows: ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$, and $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$. A *source* (resp. *sink*) place p is a place having ${}^\bullet p = \emptyset$ (resp. $p^\bullet = \emptyset$). An incidence matrix C can be associated with the net s.t. $\forall (p, t) \in P \times T : C(p, t) = W(t, p) - W(p, t)$.

Semantics of Petri nets

A marking (representing a state of the net) is a distribution of tokens over places. A marking of a Petri net N is a function $m : P \to \mathbb{N}$. The initial marking of N is denoted by m_0. The pair (N, M_0) is called a Petri net system. Figure 2.1 illustrates an example of Petri net with an initial marking where two places (p_0 and p_4) contain a token. A transition t is said to be enabled (or fireable) by a marking m (denoted by $m \xrightarrow{t}$) iff $\forall p \in {}^\bullet t$, $W(p, t) \leq m(p)$. When a transition t is fireable from a marking m, its firing (denoted by $m \xrightarrow{t} m'$) leads to a new marking m' s.t. $\forall p \in P : m'(p) = m(p) + C(p, t)$. By extension, given a finite sequence of transition σ, $m \xrightarrow{\sigma}$ and $m \xrightarrow{\sigma} m'$ denote the fireability and the firing, respectively, of σ starting from a marking m. Fireable sequences are called runs of the corresponding marked Petri net. The language of finite runs of a marked

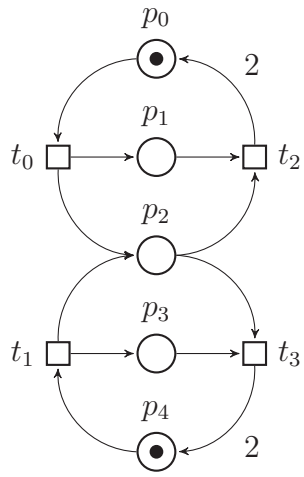


Figure 2.1: An example of Petri Net

Petri nets (N, m_0) is the (possibly infinite) set of fireable sequences of transitions i.e. $L^*(\langle N, m_0\rangle) = \{\sigma \in T^* \mid m_0 \xrightarrow{\sigma}\}$. The language of infinite runs is defined similarly and denoted by L^ω. Given a set of markings S, $Enable(S) = \{t \mid \exists m \in S;\ m \xrightarrow{t}\}$ is the set of transitions enabled by elements of S. Given a Petri net N and a marking m, the set of markings reachable from m is denoted by $R(N, m)$. The reachability graph of a marked Petri net (N, m_0), denoted by $G(N, m_0)$, is the graph where the set of nodes is equal to $R(N, m_0)$ and where an arc from m to m', labeled with t, exists iff $m \xrightarrow{t} m'$. The set of markings that are reachable from a marking m, by firing transitions of a subset T' only is denoted by $Sat(m, T')$. By extension, given a set of markings S and a set of transitions T', $Sat(S, T') = \bigcup_{m \in S} Sat(m, T')$. For a marking m, $m \not\rightarrow$ denotes the fact that m is a dead marking, i.e., $Enable(\{m\}) = \emptyset$.

From modeling point of view, several perspectives of a BP can be taken into account in its specification. One can only represent the control flow of the process, describing the different activities of the BP to be executed in some order leading from the initial state to the final state. Thus, the communication of the BP with its environment is ignored. In addition to this control flow perspective, one can be interested in the composition of several BPs communicating synchronously or asynchronously and/or the resources that are shared between the different participants. In the following, we introduce the corresponding models incrementally.

2.2.1 WorkFlow-nets

A BP can be viewed as a control structure describing its behavior in order to reach a final state (i.e. a state representing a proper termination) while abstracting from resources

and from behavior related to the interface. A particular Petri net, called *Work-Flow net (WF-net)* [124], is often used for modeling the control-flow dimension of a BP.

Definition 2 *A workflow net (WF-net for short) is defined by a tuple $N = \langle P, T, F, W \rangle$ where:*

- *$\langle P, T, F, W \rangle$ is a Petri net;*
- *N has two special places i and o such that:*
  - *i is a source place ($^\bullet i = \emptyset$),*
  - *o is a sink place ($o^\bullet = \emptyset$).*
- *each place (resp. transition) belogns to a path from i to o.*

Transitions in a WF-net correspond to activities and places represent pre-conditions for activities. A WF-net describing a workflow process satisfies two requirements. First, a WF- net is associated to an initial marking m_i (resp. a final marking m_o) where only the place i (resp. o) is marked. Without loss of generality, in the following we consider only one initial state and only one final state. m_i corresponds to a case which needs to be handled, m_o corresponds to a case which has been handled. Secondly, in a WF-net there are no dangling tasks and/or conditions. Every task (transition) and condition (place) should contribute to the processing of cases. Therefore, every transition t (place p) should be located on a path from the initial place i to the final place o.

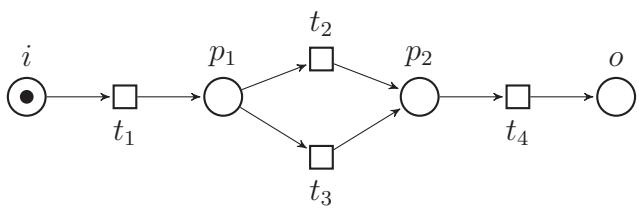


Figure 2.2: An example of an WF-Net

Figure 2.2 illustrates an example of the WF-net.

2.2.2 Open WorkFlow net

A liberal version of WF-nets, called Open WF-nets (oWF-nets) [93] has been introduced in order to allow asynchronous communication between different WF-nets. An oWF-net consists in a WF-net enriched with communication places, used for asynchronous communication. Each communication place models a channel to send (resp. receive) messages to (resp. from) another oWF-net. More precisely, each input place (i.e. with

empty pre-set) corresponds to an input port of the interface (used for receiving messages from a distinguished channel) whereas an output place (i.e. empty post-set) corresponds to an output port of the interface (used for sending messages via a distinguished channel).

Definition 3 *An open workflow net (oWF-net for short) is defined by a tuple* $N = \langle P, T, F_p \cup F_c, W, I, O \rangle$ *where:*

- I *(resp.* O*) is a set of input (resp. output) places (*$I \cup O$ *represents the set of interface places) satisfying:*
  - $(I \cup O) \cap P = \emptyset$
  - $\forall p \in I : {}^\bullet p = \emptyset$ *(input interfaces places)*
  - $\forall p \in O : p^\bullet = \emptyset$ *(output interface places)*
- $F_c \subseteq (I \times T) \cup (T \times O)$ *is a flow relation representing the arcs between interface places and transitions,*
- $W : (F_p \cup F_c) \to \mathbb{N}^+$ *is a mapping that assigns a positive weight to any arc.*
- $\langle P, T, F_p, W_{|F_p} \rangle$ *is a WF-net;*

The subnet obtained by removing from an oWF-net N the interface places and their linked arcs is called the *inner net* of N and denoted by N^*.

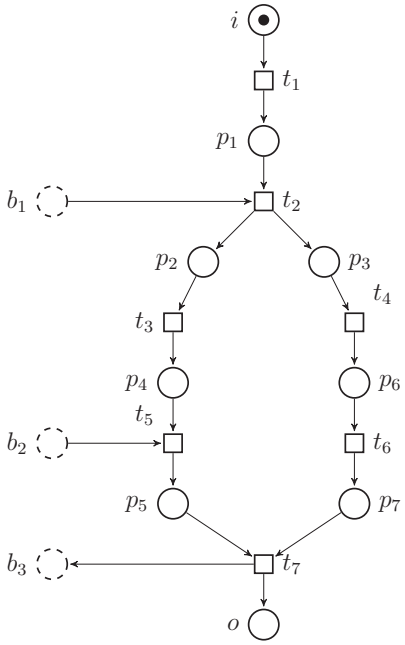


Figure 2.3: An example of an oWF-net

An example of oWF-net, where there are two input places (b_1 and b_2) and an output place b_3, is given in Figure 2.3.

2.2.3 Resource-Constraint Workflow Nets

Resource constrained workflow nets (RCW-nets) [133] was introduced to take into account resources available during the handling of tasks within the organization. Resources are claimed and released during the execution, and the task of the designer is often seen as producing a model that uses resources in the most efficient way.

Definition 4 (RCWF-net) *a resource-constrained workflow net (RCWF-net) is defined by a tuple* $N = \langle P, T, F_p \cup F_r, W, R \rangle$ *where:*

- R *is the set of resource places such that* $R \cap P = \emptyset$*;*
- $F_r \subseteq (R \times T) \cup (T \times R)$ *is a flow relation representing the arcs between resource places and transitions,*
- $W : (F_p \cup F_r) \rightarrow \mathbb{N}^+$ *is a mapping that assigns a positive weight to any arc.*
- $\langle P, T, F_p, W_{|F_p} \rangle$ *is a WF-net;*

Given an RCWF-net N, the subnet obtained by removing from N the resource places and their linked arcs is called the *inner net* of N and denoted by N^*.

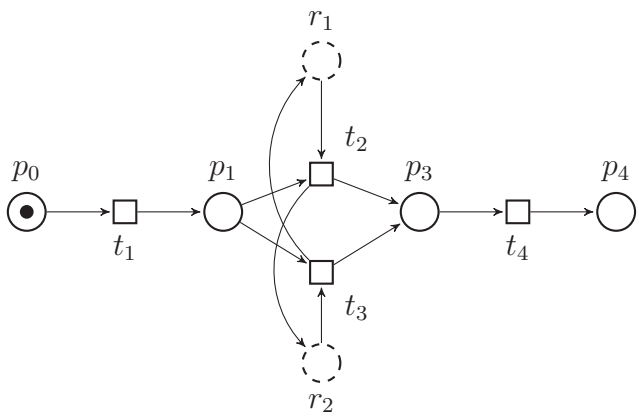


Figure 2.4: An example of RCWF-Net

Figure 2.4 gives an example of RCWF-net where two resource places (r_1, r_2) are used.

Excluding resources or interface from the model can lead to wrong verification results. In the next section, we define a model that represent both of these two perspectives.

2.2.4 Resource-Constraint open WorkFlow Nets

Combining the two previous formalisms leads to a *resource constrained open workflow net* (RCoWF-net) [72] that allows both asynchronous communication and sharing of resources between different workflows.

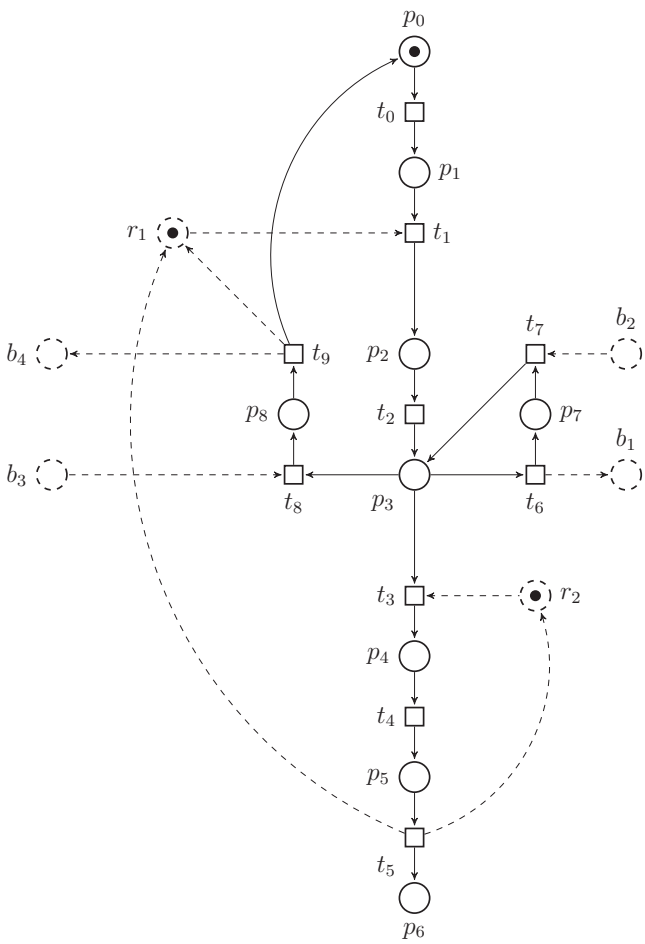


Figure 2.5: an example of RCoWF-net

Definition 5 *A RCoWF-net is a tuple* $N = \langle P, T, F_p \cup F_r \cup F_c, W, I, O, R\rangle$ *where:*

- $\langle P, T, F_p \cup F_c, W_{|F_p \cup F_c}, I, O\rangle$ *is a oWF-net;*

- $\langle P, T, F_p \cup F_r, W_{|F_p \cup F_r}, R\rangle$ *is a RCWF-net;*

Without loss of generality, we assume that resources are *durable* i.e., they can neither be created nor destroyed, they are claimed when needed and then released.

It is clear that an RCoWF-net without interface places is an RCWF-net, and an RCoWF-net without resource places is an oWF-net. Given an RCoWF-net N, the subnet obtained by removing from N both the resource and the interface places and the linked arcs is called the *inner net* of N (denoted by N^*).

2.3 Composition of RCoWF-nets

From modeling point of view, an IEBP can be described as a recursive composition of RCoWF-nets corresponding to the components' BP. Composing two RCoWF-nets is modeled by merging their respective shared constituents which are the equally labeled (input/output interface and shared resource places). Two RCoWF-nets are said to be *interface compatible* when only input interface places (resp. resource places) of the one overlap with output interface places (resp. resource places) of the other. In the following, the composition of two RCoWF-nets $\mathcal{N}_1$ and $\mathcal{N}_2$ is denoted by $\mathcal{N}_1 \oplus \mathcal{N}_2$.

Definition 6 (Composition of RCoWF-nets)

Let $\mathcal{N}_j = \langle P_j, T_j, F_j = F_{pj} \cup F_{rj} \cup F_{cj}, W_j, I_j, O_j, R_j \rangle$, for $j \in \{1, 2\}$, be two interface compatible RCoWF-nets. Let i_j and o_j, for $j \in \{1, 2\}$, be the source and the sink places of $\mathcal{N}_j$ respectively. Their composition $\mathcal{N}_1 \oplus \mathcal{N}_2 = \langle P, T, F_p \cup F_r \cup F_c, W, I, O, R \rangle$ is the RCoWF-net defined as follows:

- $P = P_1 \cup P_2 \cup \{i, o\}$, $T = T_1 \cup T_2 \cup \{t_{start}, t_{end}\}$, $R = R_1 \cup R_2$,
- $F_p = F_{p1} \cup F_{p2} \cup \{(i, t_{start}), (t_{start}, i_1), (t_{start}, i_2), (o_1, t_{end}), (o_2, t_{end}), (t_{end}, o)\}$, $F_r = F_{r1} \cup F_{r2}$, $F_c = F_{c1} \cup F_{c2}$,
- $W = (F_1 \cup F_2) \rightarrow \mathbb{N}^+$ *s.t.* $W(f) = \begin{cases} W_j(f) & if\,(f \in F_j),\ for\ j \in \{1, 2\} \\ 1 & if\,(f \in F_p \setminus (F_{p1} \cup F_{p2})) \end{cases}$
- $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$, $O = (O_1 \cup O_2) \setminus (I_1 \cup I_2)$,

Note that, in the above definition, two new places (i and o) and two new transitions (t_{start} and t_{end}) have been added to the composition in order to respect the RCoWF-net structure. In particular, the fact that there is one input and one output places in the net.

The RCoWF-net composition is commutative and associative i.e. for interface compatible RCoWF-nets $\mathcal{N}_1$, $\mathcal{N}_2$ and N_3: $\mathcal{N}_1 \oplus \mathcal{N}_2 = \mathcal{N}_2 \oplus \mathcal{N}_1$ and $(\mathcal{N}_1 \oplus \mathcal{N}_2) \oplus N_3 = \mathcal{N}_1 \oplus (\mathcal{N}_2 \oplus N_3)$.

2.4 Representation of the Reachable Configurations

More than the models used to specify BPs (or IEBP), we are interested in the behavioral properties of processes. We hence present in this section the different possible formalisms allowing to represent such behavior. Depending on desired properties, one formalism could be more suitable than an other. We distinguish three different formalisms: Labeled Transition Systems (LTS), which are suitable for even-based reasoning, Kripke Structures (KS), which are suitable for a state-based reasoning, and Labeled Kripke Structures (LKS)

which allow both reasonings. All of these structures are derived from transition systems that are usually used to represent the potential behavior of discrete systems. Nevertheless, since we analyze BP which distinguish a particular state recognized as final (e.g. m_o for WF-nets), in the following, a final state, called s_f, is added to each of these formalisms. Moreover, since the work presented in this thesis is based on the SOG abstraction model, we recall, in this section, the event- and state-based versions the SOG. The nodes of a SOG being a encoded with BDDs, we also recall how this symbolic structure is used to represent a set of states.

2.4.1 Labeled transition Systems

An *LTS* is a graph where the nodes represent the possible reachable states of a system (starting from some initial state), and edges represent state transitions. A focus is here done on the actions labeling the edges while the states are not necessarily detailed. One can also identify one or more states as final. For instance, the *LTS* associated with a Petri net is the corresponding reachability marking graph.

Definition 7 (Labeled transition System) *A* Labeled transition System *is a* 5*-tuple* $\langle \Gamma, Act, \rightarrow, s_i, s_f \rangle$ *where:*

- Γ *is a finite set of* states *;*
- Act *is a finite set of* actions*;*
- $\rightarrow \subseteq \Gamma \times Act \times \Gamma$ *is a* transition relation *;*
- $s_i \in \Gamma$ *is the* initial state*;*
- $s_f \in \Gamma$ *is the* final state.

We restrain the set of states Γ to those that are reachable from the initial state. Moreover, we assume that the final state is terminal (has no successors). Hence, a final state is a legal dead state.

2.4.2 Kriple Structure

KS is a variation of the transition system where there is a focus on the states' properties w.r.t. to a predefined set of atomic propositions. A labeling function maps each node of a *KS* to a set of atomic propositions that hold in the corresponding state. For a given Petri net, the reachability marking graph can be seen as a *KS* when each reachable state is labeled with the truth values of some atomic propositions (e.g. is the markings of some places $p_1 \dots p_n$ are equal to some values $v_1 \dots v_n$?).

Definition 8 (Kripke structure) *Let AP be a finite set of atomic propositions. A* Kripke structure *over AP is a* 5*-tuple* $\langle \Gamma, L, \rightarrow, s_0, s_f \rangle$ *where:*

- Γ *is a finite set of* states *;*
- $L : \Gamma \rightarrow 2^{AP}$ *is a labeling (or interpretation) function;*
- $\rightarrow \subseteq \Gamma \times \Gamma$ *is a* transition relation *;*
- $s_0 \in \Gamma$ *is the* initial state.
- $s_f \in \Gamma$ *is the* final state.

2.4.3 Labeled Kriple Structure

Given a reachable state of the model, one can be interested in the (state-based) atomic propositions labeling the state (given by the labeling function L) and in the events that can occur starting from this state (which are the labels of the outgoing arcs). A mix of the two previous models, called *Labeled Kripke Structure*, can then be used to represent the behavior of the system.

Definition 9 (Labeled Kripke structure) *Let AP be a finite set of atomic propositions and let Act be a set of actions. A* Labeled Kripke structure *over AP is a* 6*-tuple* $\langle \Gamma, Act, L, \rightarrow, s_0, s_f \rangle$ *where:*

- $\langle \Gamma, Act, \rightarrow, s_0, s_f \rangle$ *is an LTS*
- $\langle \Gamma, L, \rightarrow, s_0, s_f \rangle$ *is a KS*

2.4.4 Binary Decision Diagrams

A binary decision diagram (BDD) is a data structure that is used usually to represent a Boolean function. It can be considered as a compressed representation of sets or relations. Unlike other compressed representations, operations are performed directly on the compressed representation, i.e. without decompression.

Definition 10 (Binary Decision Diagram) *A* Binary Decision Diagram (BDD) *is a rooted, directed acyclic graph with :*

- *one or two terminal nodes of out-degree zero labeled* $false$ *or* $true$*, and*
- *a set of variable nodes* u *of out-degree two. The two outgoing edges are given by two functions* $low(u)$ *and* $high(u)$*.*

- *A variable $var(u)$ is associated with each variable node.*

Definition 11 (Ordered BDD) *A BDD is* Ordered *(OBDD), if on all paths through the graph, the variables respect a given linear order $x_1 < x_2 < \cdots < x_n$.*

Definition 12 (Reduced OBDD) *An OBDD is* Reduced *(ROBDD) if :*

- *(Uniqueness) no two distinct nodes u and v have the same variable name and low- and high-successor (i.e $(var(u) = var(v) \wedge low(u) = low(v) \wedge high(u) = high(v)) \Rightarrow u = v$)*

- *(Non-redundant tests) no variable node u has identical low- and high-successors, (i.e. $low(u) \neq high(u)$).*

For instance, for safe Petri nets (where every place is marked with at most one token), one can consider a marking as a Boolean vector in the form of $S = \{0, 1\}^m$ for $m \geq 1$. A set of markings can be represented with a BDD by using a Boolean characteristic function. Let S be the set of all possible markings, and let $R \subseteq S$ be a subset of markings, then, the characteristic function f_R is defined as follows: $f_R : S \longrightarrow \{0, 1\}$, and,

$$f_R(s) = \begin{cases} 1, & \text{if } s \in R \\ 0, & \text{otherwise} \end{cases}$$

Let us consider, as example, a safe Petri net containing four places i, p_1, p_2, o, and let R be the set of markings where only one place is marked. The truth table of the corresponding characteristic function is as follows:

i	p_1	p_2	o	f_R
1	0	0	0	1
0	1	0	0	1
0	0	1	0	1
0	0	0	1	1
	..	..		0

In order to construct the BDD graph associated with this example, we define $V = \{i, p_1, p_2, o\}$ as the set of totally ordered variables (e.g., $i < p_1 < p_2 < o$). Figure 2.6 illustrates such a BDD. It is actually an ROBDD, since there are no isomorphic sub-graphs and no redundant nodes. Dotted (resp. solid) outgoing arc of a node u represents the successor $low(u)$ (resp. $high(u)$. A path leading to a *true* leaf corresponds to a marking in R.

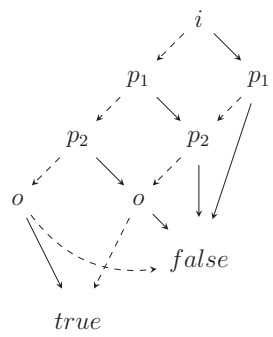


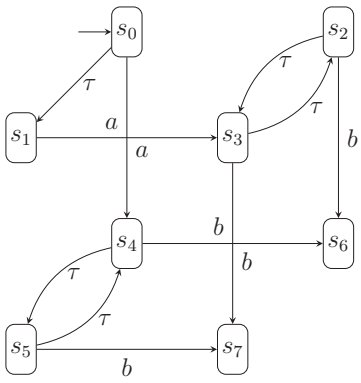
Figure 2.6: example of a BDD

2.4.5 The Event- and State-based SOGs

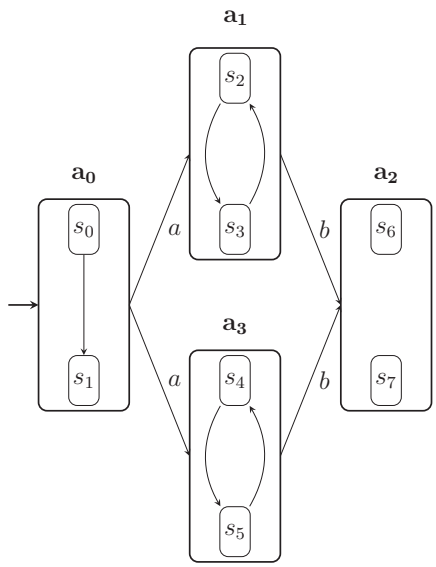
The Symbolic Observation Graph (SOG) has been introduced as an abstraction of the a *LTS* (resp. *KS*) that preserves the event-based (resp. state-based) *LTL* properties in [55] (resp. [75]). It is an explicit graph where nodes are sets of states (called *aggregates*) encoded symbolically using BDDs. It is guided by the set of atomic propositions that occur in the *LTL* formula to be checked. These are called *observed atomic propositions* while the others are *unobserved*. The main difference between the event- and the state-based versions is the aggregation criterium: In the first (observed atomic proposition corresponds to some actions of the system), an aggregate contains states that are connected by unobserved actions. In the second (observed atomic propositions are boolean state-based ones), an aggregate regroups states with the same truth values of the observed atomic propositions. An aggregate is called *final*, and denoted a_f, if it contains a final state of the corresponding model. In the following, we present the definition of an aggregate in both SOG's versions (the complete definition of the event-SOG and the state-SOG can be found in [77] and [75] respectively).

Definition 13 (Event-based aggregate) *Let $\mathcal{T} = \langle \Gamma, Act, \rightarrow, s_i, s_f \rangle$ be an LTS with $Act = Obs \cup UnObs$. An* aggregate *is a tuple $a = \langle S, d, l, f \rangle$ defined as follows:*

1. *S is a non-empty subset of Γ satisfying $Sat(S, UnObs) = S$;*
2. *$d \in \{true, false\}$; $d = true$ iff $\exists s \in S \mid s \not\rightarrow$;*
3. *$l \in \{true, false\}$; $l = true$ iff S contains an unobserved cycle (involving unobserved actions only);*
4. *$f \in \{true, false\}$; $f = true$ iff $s_f \in S$.*



(a) Example of *LTS*



(b) A corresponding SOG: $Obs = \{a, b\}$

Figure 2.7: An *LTS* and its SOG

Figure 2.7 illustrates an example of *LTS* (Figure 2.7(a)) and a corresponding SOG (Figure 2.7(b)). The set of observed actions contains two elements $\{a, b\}$ while τ represents any unobserved action. The presented SOG consists of 4 aggregates $\{a_0, a_1, a_2, a_3\}$ and 4 edges. Aggregates a_1 and a_3 contain circuits but no dead states, whereas a_2 contains two dead states but no circuit. Notice that states of the *LTS* are partitioned into aggregates which is not necessary the case in general (i.e. a single state may belong to two different aggregates). Moreover, one can merge a_1 and a_3 within a single aggregate leading to a deterministic SOG.

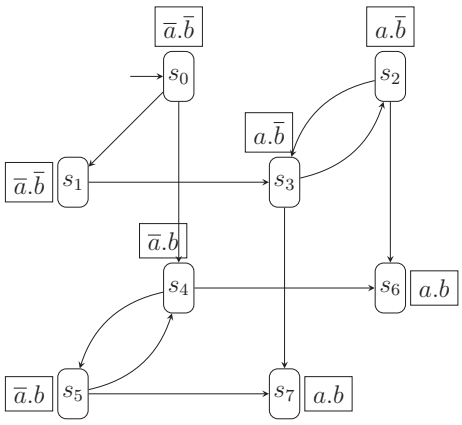
Definition 14 (State-based aggregate) *Let $\mathcal{K} = \langle \Gamma, L, \rightarrow, s_0, s_f \rangle$ be a KS over an atomic proposition set AP. An* aggregate *a of $\mathcal{K}$ is a tuple $\langle S, d, l, f \rangle$ where:*

1. *S is a non empty subset of Γ satisfying $\forall s, s' \in a, L(s) = L(s')$.*
2. *$d \in \{true, false\}$; $d = true$ iff $\exists s \in S \mid s \not\rightarrow$;*
3. *$l \in \{true, false\}$; $l = true$ iff S contains a cycle;*
4. *$f \in \{true, false\}$; $f = true$ iff $s_f \in S$.*

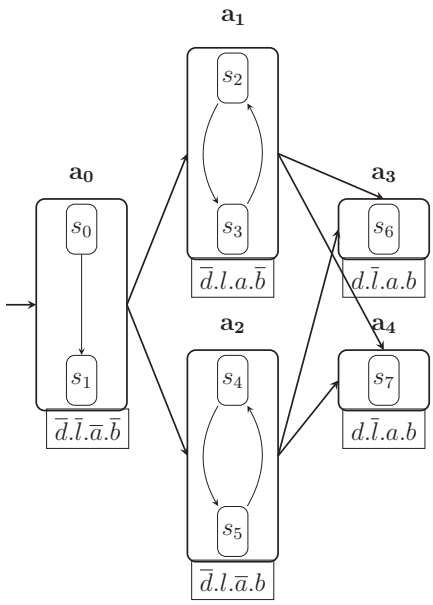
The labeling function $L : \Gamma \rightarrow 2^{AP}$ is then extended to aggregates as follows:
$L(a) = L(s)$ iff $s \in S$.

Example:

Figure 2.8 illustrates an example of KS (Figure 2.8(a)) and a corresponding SOG (Figure 2.8(b)). The set of atomic propositions contains two elements $\{a, b\}$ and each state of the KS is labeled with the values of these propositions. The presented SOG consists of 5 aggregates $\{a_0, a_1, a_2, a_3, a_4\}$ and 6 edges. Aggregates a_1 and a_2 contain circuits but no dead states, whereas a_3 and a_4 have each a dead state but no circuit. Each aggregate a is indexed with a triplet $(d, l, L(a))$. Symbols d and l are interpreted similarly to the event-state based. Again, in this case, the states of the KS are partitioned into aggregates but this is not necessary the case in general. Also, one can merge a_3 and a_4 within a single aggregate and still respect the original definition of a SOG.



(a) Example of Kripke structure



(b) A corresponding SOG: $AP = \{a, b\}$

Figure 2.8: A Kripke structure and its SOG

2.5 Behavioral Properties of IEBP

We are interested in the analysis of the behavior of BPs from both the local and the global (after composition) point of views. In particular, we consider two kinds of properties: a domain independent property called soundness [124] (as well as some of its variants), and specific properties expressed with Linear Temporal Logic (LTL).

2.5.1 Soundness Properties

The soundness property can be regarded as a minimal correctness criterion for interacting BPs and it guarantees the absence of several types of anomalies in a process model. It can be formulated, on a WF-net, without domain knowledge and requires the following conditions: (1) *option to complete*: from any reachable marking, it is possible to reach the final marking, (2) *proper completion*: no reachable marking is strictly greater than a final marking. It means that It should not be possible that the workflow definition signals termination of a case while there is still work in progress for that case, and (3) *no dead transitions*: each transition is fireable at some reachable marking. It means that for every task, there should be an execution of the workflow process definition that executes it.

If we assume an appropriate notion of fairness, then the requirements of the soundness property implies that a final state is eventually reached from an initial state. If we require termination without such an assumption, all models allowing loops in their execution sequences would be unsound, which is clearly undesirable.

In addition to the original definition of soundness, we consider the following variants of this property. *Relaxed soundness* [33] allows for potential deadlocks and livelocks, however, each transition should occur in at least one "good" execution path. *Weak soundness* [88] allows for dead transitions as long as a final marking is reachable from any state. Finally, *easy soundness* [129] requires that a final marking is reachable from the initial marking. It is obvious that soundness implies both relaxed and weak soundness which are incomparable and that each other soundness notion implies easy soundness.

Definition 15 *Let $N = \langle P, T, F_p \cup F_r \cup F_c, W, I, O, R\rangle$ be a marked RCoWF-net. Let m_0 be its initial marking and m_f its final marking. N is said to be :*

- *sound iff the following requirements are satisfied:*
  - *option to complete: $\forall m \in R(N^*, m_0)$, $m_f \in R(N^*, m)$;*
  - *proper completion: $\forall m \in R(N^*, m_0)$, $m \geq m_f \implies m = m_f$;*
  - *no dead transitions: $\forall t \in T$, $\exists m \in R(N^*, m_0)$ s.t. $m \xrightarrow{t}$.*
- *relaxed sound iff: $\forall t \in T$, $\exists m, m' \in R(N^*, m_0)$, $m \xrightarrow{t} m' \wedge m_f \in R(N^*, m')$.*
- *weak sound iff: $\forall m \in R(N^*, m_0)$ s.t. $m_f \in R(N^*, m)$;*
- *easy sound iff: $m_f \in R(N^*, m_0)$.*

2.5.2 Linear Temporal Logic propreties

LTL is an extension of the propositional logic which allows reasoning over infinite sequences of states. LTL is widely used for the verification concurrent systems with respect to a large class of properties (e.g.f safety, liveness, ... etc.). Here, we recall the syntax and the semantics of the state-based LTL logic (the event-based logic can be deduced by considering that the atomic propositions appearing in an LTL formula are actions of the system). In consequence, we chose to represent the semantics (behavior) of a BP models (e.g., RCoWF-nets) by an *KS*.

Syntax of LTL

Definition 16 *Given a set of atomic propositions AP, an LTL formula is defined inductively using the standard boolean operators, and the temporal operators X (next) and U (until) as follows:*

- *each member of AP is a formula,*
- *if ϕ and ψ are LTL formulae, so are $\neg\phi$, $\phi \vee \psi$, $X\phi$ and $\phi U \psi$.*

Other temporal operators such as F (futur) and G (globally) can be derived as follows: $F\phi = true \cup \phi$ and $G\phi = \neg F \neg \phi$.

Semantics of LTL

Checking an LTL formula over a formal model of a system (e.g. Petri net) is performed by analyzing its *KS*. An interpretation of an LTL formula is an infinite run $w = x_0 x_1 x_2 \ldots$, assigning to each state a set of atomic propositions that are satisfied within that state. We write w^i for the suffix of w starting from x_i and $p \in x_i$, for $p \in AP$, when p is satisfied by x_i. The LTL semantics is then defined inductively as follows:

- $w \models p$ iff $p \in x_0$,
- $w \models \phi \vee \psi$ iff $w \models \phi$ or $w \models \psi$,
- $w \models \neg\phi$ iff not $w \models \phi$,
- $w \models X\phi$ iff $w^1 \models \phi$, and
- $w \models \phi U \psi$ iff $\exists i \geq 0$, s.t., $w^i \models \psi$ and $\forall 0 \leq j < i$, $w^j \models \phi$.

A *KS* $\mathcal{K}$ satisfies an LTL formula ϕ, denoted by $\mathcal{K} \models \phi$, iff ϕ is satisfied by any infinite run of $\mathcal{K}$.

Model Checking of LTL formulae

The standard automata-theoretic approach [136] to model checking LTL properties is based on the use of Büchi automata [20]. Given a LTL property ϕ and a formal model of the system (e.g., *KS*), the automata-theoretic approach for LTL model checking is based on converting the negation of the property ($\neg\phi$) in a Büchi automaton, composing the automaton and the model, and finally checking the emptiness of the synchronized product [135]. The system satisfies ϕ iff the synchronized product accepts no words (i.e. iff its language is empty). The last step is the crucial stage of the verification process due to the state space explosion problem.

LTL model checking necessitates to capture special runs of the underlying *KS* called *maximal paths*. A maximal path is either a finite run leading to a dead state, or an infinite run.

Definition 17 (maximal paths) *Let $\mathcal{K}$ be LKS and let $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$ be a path of $\mathcal{T}$. Then, π is said to be* a maximal path *if one of the two following properties holds:*

- $s_n \not\rightarrow$,
- $\exists 0 \leq m \leq n$ *s.t.* $s_m \xrightarrow{a_{m+1}} \cdots \xrightarrow{a_n} s_n$ *is a circuit.*

Since LTL is interpreted on infinite paths, a usual solution in automata theoretic approach to check LTL formulae on a *KS* is to add a self loop on its dead states.

2.6 Conclusion

This Chapter introduced the theoretical basis of the work presented in this manuscript. We supplied formal description of both the structure and the behavior of a BP. We also defined the two types of properties which will be checked formally on BPs and IEBPs as it will be described in Chapters 4 and 5 respectively.

Chapter 3

Related Work

Contents

3.1 Introduction **27**

3.2 Formal Verification Approaches **28**

3.2.1 Theorem Proving 28

3.2.2 Model Checking Approaches 29

3.3 Related BP Verification Approaches **30**

3.4 Related IEBP Verification Approaches **31**

3.5 Conclusion **35**

3.1 Introduction

In this thesis, we are interested in the formal verification of inter-enterprise business processes (IEBP for short) from local and global point of views. Although, formal verification approaches have been widely developed, since three decades, independently from a target domain, we believe in domain-specific research approach. Indeed, designing domain-specific verification approaches may be more effective than general purpose verification techniques. Specific domains could have particular requirements/constraints and even specific properties that make existing monolithic verification approaches inappropriate or even inapplicable. On one hand, doing so could allow to take benefit from the own characteristics of the domain's applications, leading to a better efficiency. On the other hand, domain-specific approaches could bring new ideas to improve the verification in the general case. This would ideally create a virtuous circle where general and specific-domain verification approaches enrich each other.

In the following, we first recall the principle of the two main formal verification approaches, namely *theorem proving* and *model checking*, before discussing related work on formal verification of BPs and IEBP respectively.

3.2 Formal Verification Approaches

Formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics. It can be helpful in proving the correctness of systems such as: cryptographic protocols, combinational circuits, digital circuits with internal memory, and software expressed as source code. Formal verification encompasses a number of methods for proving correctness. Two well-established ones are theorem proving [16, 12, 99] and model checking [107, 50, 25].

3.2.1 Theorem Proving

In theorem proving, a number of proof obligations are generated from the specification and the implementation. These are formal statements whose validity entails the correctness of the system. Assisted by the theorem prover, the user constructs the proof of each obligation, either interactively or in a highly automated way, depending on the capabilities of the method used. A shortcoming of theorem proving is that it often requires substantial interaction of the user. The system that needs to be analyzed is mathematically modeled in an appropriate logic and the properties of interest are verified using computer based formal tools. The use of formal logics as a modeling medium makes theorem proving a very flexible verification technique as it is possible to formally verify any system that can be described mathematically. The core of theorem provers usually consists of some well-known axioms and primitive inference rules. Soundness is enssured as every new theorem must be created from these basic axioms and primitive inference rules or any other already proved theorems or inference rules. The verification effort of a theorem in a theorem prover varies from trivial to complex depending on the underlying logic. For instance, first-order logic [48] is restricted to propositional calculus and terms (constants, function names and free variables) and is semi-decidable. A number of sound and complete first-order logic automated reasoners are available that enable completely automated proofs. More expressive logics, such as higher-order logic [18], can be used to model a wider range of problems than first-order logic, but theorem proving cannot be fully automated for these logics and thus involves user interaction to guide the proof tools. The main advantage of theorem provers is their ability to be used in the case of infinite systems. However, they have the disadvantage of not being completely automatic. Indeed, proof assistants require a user with a strong expertise in the underlying system to "give" the path leading to the solution of the system. This problem is the most important obstacle impeding a wider industry adoption. The theorem proving is mainly used for the verification of hardware system [30]. For example, Method B [6] was used to test the critical components of the

automatic train operating system for METEOR (for the Line 14 of the Paris metro) [10].

3.2.2 Model Checking Approaches

Model checking [41, 27, 107, 25] is a powerful and widespread technique for the verification of concurrent systems. Given a (generally finite-state) formal description of the system to be analyzed and a number of properties, often expressed as formulas of temporal logic, that are expected to be satisfied by the system, the model checker either confirms that the properties hold or reports that they are violated. In the latter case, it provides a counterexample: a witness run that shows that the property is violated. Such a run gives a valuable feedback and points to design errors. At the core of model checking are algorithms that implement state space traversals. The reachable state space is traversed to find error states that violate safety properties, or to find cyclic paths on which no progress is made as counterexamples for liveness properties. In particular, given a Linear-time Temporal Logic (LTL) property and a formal model of the system (e.g., Kripke structure), the automata-theoretic approach for LTL model checking is based on converting the negation of the property in a Buchi automaton (or tableau), composing the automaton and the model, and finally checking the emptiness of the synchronized product [135]. The last step is the crucial stage of the verification process due to the state space explosion problem (the number of reachable states of a concurrent system grows exponentially with the number of its components) which is the main hindrance for wider application of the model checking technique. Due to the state space explosion problem, it may just take too much time to explore all the reachable states and typically also too much space. During the last three decades, numerous techniques have been proposed to cope with the state space explosion problem in order to get a manageable state space and to improve scalability of model checking. These techniques can roughly be classified in two large families: explicit and symbolic approaches.

Explicit model checking approaches explore an explicit representation of the product graph. A common optimization builds the graph on-the-fly as required by the emptiness check algorithm: the construction stops as soon as a counterexample is found (e.g., [28, 58, 29]). Partial order reduction (e.g., [13, 120]) is a reduction technique exploiting independence of some transitions in the system to discard unnecessary parts of the system state space. Another source of optimization is to take advantage of stuttering equivalence between paths in the Kripke structure when verifying a stuttering-invariant property [44]: this is done either by ignoring some paths in the Kripke structure [65], or by representing the property using a testing automaton [57].

Symbolic model checking approaches tackles the state space explosion problem by representing the product automaton symbolically, usually by means of decision diagrams

(a concise way to represent large sets or relations, e.g., BDDs [19]). Various symbolic algorithms exist to verify LTL using fixepoint computations (see [47, 118] for comparisons and [66] for more details). As-is, these approaches do not mix well with partial order, stuttering invariant reductions and on-the-fly emptiness checks.

However explicit and symbolic approaches are not exclusive, some combinations have already been studied [14, 55, 115, 75], to get the best of both worlds. They are referred to as hybrid approaches and consist in replacing the KS by an explicit graph where each node contains sets of states of the KS, that is an abstraction of the KS preserving properties of the original KS. The SOG-based technique, which is the core of this thesis work is an example of such approaches.

3.3 Related BP Verification Approaches

The verification process has matured to a level where it can be used in practice (techniques that assumes today's modeling languages not only simplified process models without the more advanced constructs). By performing this verification at design time, it is possible to identify potential problems, and if so, the model can be modified before it is used for execution. As some systems (e.g., workflow, BP systems) rely on process models for execution of work, careful analysis of process models at design time can greatly improve the reliability of such systems.

Since the mid nineties, many researchers have been working on workflow/business process verification techniques. In the literature we can find different directions regarding the verification and validation of a BP. At the beginning, most of the works focused on rather simple languages, e.g., AND/XOR-graphs which are even less expressive than classical Petri nets. Then, the use of Petri nets in workflow verification have been studied [3, 124, 139, 128, 111, 114, 127]. Formal methods used for verifying BP was proposed based on π-calculus [83] or Petri Nets in [125], while other techniques for showing consistency of BPs written in Business Process Execution Language for Web Services (BPEL4WS) [64] was based on Model-Checking (MC) [35].

Various properties are considered when we deal with the correctness of business processes. In [3, 124] the foundational notions of WF-nets and soundness are introduced. In [4], the author describes how structural properties of a workflow net can be used to detect the soundness property. In [140], the authors present new verification techniques that can be used to assess the correctness of real-life models. The proposed approach relies on using formal methods to determine the correctness of business processes (with cancellation and OR-joins) with respect to four studied properties namely, soundness, weak soundness, irreductible cancellation regions, and immutable OR-joins. Another related

work is presented in [128] as an extension of the results given in [130] (where it is proved that soundness is undecidable for WF-nets with reset arcs). Different notions of soundness are investigated (not only classical soundness but eight variants) and authors consider also WF-nets with inhibitor arcs. Another alternative approach for deciding relaxed soundness property using invariants is presented in [137]. For other authors [37, 141, 138], it was necessary to suggest some reduction rules for Petri nets and for various subclasses of Petri nets in order to improve the analysis and the verification of processes.

The problem of verification while considering advanced relative and absolute temporal constraints is studied in [22]. Firstly, a set of rules are proposed to prevent the designer to specify some faulty temporal combinations of absolute temporal constraints, early on, before the execution step. Second, a mapping step whose aim is to map timed business processes into timed automata is proposed in oder to capture relative temporal constraints. Finally, using the defined formal model, the proposed model checking-based verification approach aims to validate business processes against their temporal constraints. In the same context, exploiting results achieved in the field of temporal logics and runtime verification, a runtime verification of flexible, constraint-based process models formalized in terms of LTL on finite traces was introduced in [85]. The focus here was on violations arising from interference of multiple constraints. A conflicting set provides a minimal set of constraints with no continuation where all constraints can be satisfied. In [84], the authors investigated automata-based techniques for the runtime verification of LTL-based process models. In particular, they proposed colored automata to provide intuitive diagnostics for singular constraints and ways to continue verification even after a violation has taken place. Intuitively, a colored automaton is a finite state automaton built for the whole set of constraints composing a process model, where each state contains specific information (colors) indicating the state of individual constraints. Both of these two approaches have been implemented in the context of the Declare system [104] and ProM2 [132]

The nature of today's global competitive market has given rise to increased organizational cooperation in form of strategic alliances where organizations no longer compete in isolation, but as value chains. Globalization and increased market pressures lead organizations to enter into strategic partnerships with the overall goal of achieving a competitive advantage. In the next section, we give a brief overview of some works in the context of the verification of IEBP.

3.4 Related IEBP Verification Approaches

The importance of external collaboration is increasing since companies are redefining their vertical architectures [62], i.e. their scope and boundaries. Composition consists of all

activities that are required to combine and the link existing workflow or BP fragments and other components to create new processes called IEBPs. Depending on the current environment where the composition of BP takes place, one can refer to IEBP or service composition. In the following, we present some existing works which studied both of IEBP and service composition since the composition of IEBP is closely related to the selection and the composition of services topic.

With a growing number of external business relationships, business processes need to be more closely aligned across organizational boundaries. Hence, business process modeling and design have to be enhanced and extended to cover these requirements in order to facilitate the analysis process of data flow dependencies between BPs. The question, how can partner BPs be coordinated? has attracted already the attention of many researchers: Existing approaches can be classified into three categories: manual, partly automated, or fully automated. Approaches in the manual category assume that a user manually designs a process composition, including the binding to concrete services. In this category we find languages like BPEL [64] and JOpera [100] and concrete composition prototypes [143]. In semi-automatic approaches [43], the user must provide a composition skeleton which defines the process logic. This skeleton is then instantiated automatically by searching for atomic processes that match each of the processes specified in the skeleton. The focus of these approaches lies on automatically finding substitute BPs for a specified process. Fully automatic approaches (e.g. [11, 109, 39]) mostly come from the field of AI or formal reasoning. These approaches require that processes are specified formally with pre- and post-conditions. This puts a considerable burden on the shoulders of processes designers, since the most specification formalism (e.g. WSDL [23]) do not require that level of detail and hence need to be annotated with the additional pre- and post-conditions.

Many researchers have been interested in the area of IEBP composition [98, 60]. The main goal is to ensure that the process obtained by composition has the desired behavior. The need for an efficient design has been highlighted and underpinned by a large number of case studies on companies that have successfully reshaped their business relationships [24, 117, 42]. When each component of an IEBP ignores the detailed description of its partners, their abstractions should be sufficient to decide about the correctness of the whole process. Indeed, the correct behavior of each process (analyzed independently) does not guarantee the correction of the behavior of the process obtained by composition. Automating and optimizing this composition and the verification tasks is of high interest in research communities (eg. [98, 60, 80, 51, 36]). The question, what is the more suitable abstraction of a process will represent its public view, has been dealt with in the literature since many years (e.g. [21, 82, 92, 49, 91]). For instance, the public-to-private approach [122] consists of three steps. Firstly, the organizations involved

agree on a common and sound public workflow, which serves as a contract between these organizations. Secondly, each task of the public workflow is mapped into one of the domains (i.e., organization). Each domain is responsible of a part of the public workflow, referred to as its public part. Thirdly, each domain can now make use of its autonomy to create a private workflow. To satisfy the correctness of the overall inter-organizational workflow, however, each domain may only choose a private workflow which is a subclass of its public part [131]. The public-to-private approach allows to the local processes to be decoupled as much as possible and to have some degree of understanding about the nature of the interaction between the processes of the different business partners. The main disadvantage of this approach is the confidentiality that prevents a complete view of local workflow. For instance, to check the deadlock property, one needs the model of the global workflow. This model however is often not available for inter-organizational workflow since organizations are not willing to disclose their workflows [67](for privacy reasons).

The advances of Internet technology have an increasing effect on the way we do business in the current knowledge- and network-based economy. As a result, one of the key challenges for current businesses is how to effectively and efficiently integrate inter- and intra-enterprise applications. By using the Internet as the primary platform for communication, interoperability, and integration, information systems are playing an increasingly important role in providing businesses competitive advantages [144]. In recent years, technologies used for describing processes, have begun to have a profound effect on the way e-business applications are developed and the way in which sophisticated processes are designed, implemented, and managed. Several approaches investigated the issue of IEBP as composite services in general, and the Web services composition in particular. For instance, a technique for modeling multiple web services interactions between BPEL processes is discussed in [142] using an extension of Petri net models called composition net (C-net). The authors analyze the model through structural properties instead of the reachability state space in order to check compatibility: the compatibility is ensured when the composite net contains a non empty minimal siphon. They impose constraints on the model to prevent it from reaching incompatible cases by using a corresponding policy based on appending additional information to channels. Then, these channels are transformed back to a BPEL description so that a new compatible web service is obtained. An other approach [38] based on mediation aided composition has been widely adopted when dealing with incompatibilities of services. In this work, given two services modeled by oWF-net, the authors propose to compose them using Mediation Transitions (MTs). They serve as information channel specifying the transferring relation of messages between different services. Then composition compatibility is verified by automatically constructing

and analyzing the modular reachability graph (MRG) of the composition which is an abstraction of the original state graph. Even if the performance of this approach is notable compared to classical ones, MRG is represented explicitly. Another related approach has been introduced in [126]. In this work, the authors present a technique based on the Operating Guideline [93] for automatically checking accordance between a private view and a public view associated to each service involved in the overall process (composition of partners). A multiparty contract is specified in order to define the rules of engagement of each partner without describing its internal behavior. It can be seen as the composition of the public views from all partners. Based on the resulting contract, all participants implement their private view on the global process in such a way that it agrees with the contract. Then, checking accordance guarantees that the process is deadlock-free and that it will always terminate properly.

In [54], the authors propose an approach for services retrieval based on behavioral specification. The idea consists in reducing the problem of service behavioral matching to a graph matching problem and then adapting existing algorithms for this purpose. The complexity of a graph matchmaking algorithm used is $O(m^2 * n^2)$ in the best case and $O(m^n * n)$ in the worst case where m is the number of nodes of the request graph and n is the number of nodes of the advertised graph [54]. It is obvious that this approach is not suitable for workflow matching and composition when the number of advertised abstractions increases. Another approach for workflow matchmaking was proposed in [87, 90, 89]. It assumes that two workflows match if they are equivalent. To reach this end, the author introduces the notions of communication graph c-graph and usability graph u-graph. If the u-graph of a workflow is isomorphic to the c-graph of another workflow, then the two workflows will be considered equivalent. However, the complexity of c-graph construction is exponential [87] in terms of the number of nodes. Moreover, it is well known that the subgraph isomorphism detection problem is NP-complete (see for example [110]).

Among works that used LTL logics to express correction properties, we distinguish two classes of approaches: Those whose tackle the issue of runtime verification and monitoring based on observed behavior rather than the modeled behavior, and those which start the analysis and the verification from the specification of the service. We can refer to the survey paper [8] for an overview of existing approaches. Many authors on software engineering and service oriented computing were interested to the declarative logic based language to define a service. For instance, in some recent works [85, 84], the authors propose to use dynamic language based on LTL (called "ConDec" [103, 102] and "DecSerFlow" [101, 103] for describing the constraints of service. They specify what should be done instead of specifying how it should be done by leaving more flexibility

to users. This approach can be applied in the context of process mining since the goal is to check conformance. Based on a variety of events which is logged, authors check whether a service follow the specification or not. Since LTL is likely to be difficult to use directly by the end user, authors suggest a "Declare" tool [104] which offer a graphical notation for common patterns of temporal constraints which are compiled to LTL later. The idea is to investigate automata-based techniques for the verification process. The authors of [104] of use colored automata as a finite state automaton built for the whole set of constraints composing a different services, and this to provide intuitive diagnostics for singular constraints and ways to continue verification even after a violation has taken place. Classical approaches was used to translate a specification expressed in LTL into Buchi automaton accepting all infinite execution traces satisfying the formula [50], and the verification algorithm is similar to model checking. The work in [49] is related to our work in the sense that the analysis starts from service models. Authors present a tool for analyzing interactions of composite services specified in BPEL language and communicate through asynchronous messages. The approach use SPIN Model checker [61] as a finite state verification tool. For this, BPEL specifications are translated to the verification language of SPIN after a mapping to an intermediate representation called "Guarded Automata". Since SPIN can only achieve partial verification (by fixing the size of input queues), the authors, based on the concept of synchronizability, show that a large class of composite services can be completely using SPIN.

In [116], the authors present various composition alternatives and their ability to preserve an example of generic property: the relaxed soundness [33]. The aim of this work was to analyze a list of significant composition techniques in terms of WF-nets and to prove that the composition of relaxed sound models is again relaxed sound. Since relaxed sound models might have deadlocks, using these composition techniques does not preserve the deadlock-freeness property. In order to verify this property one has to explore the composed model, even though the component models are deadlock-free.

3.5 Conclusion

After a brief description of the main two formal verification approaches, we discussed in this Chapter, some work of the literature dealing with BPs. The need of formal methods to design and analyze BPs is reflected in numerous research papers in the domain. This need is naturally extended to IEBPs where several independent BPs collaborate in order to accomplish a global goal. Arises then the problem of preserving both privacy of each component and its "nice" properties. The work presented in this phD thesis goes on step forward in this topic using the hybrid SOG-based approach and answering the

double challenge of preserving privacy of each component of and IEBP, and allowing the verification of the whole process in a modular way.

Chapter 4

Using SOGs for flat verification

Contents

4.1 Introduction **37**

4.2 Using SOGs for Hybrid LTL **38**

4.2.1 Revisiting SOG for Hybrid LTL 39

4.2.2 SOG-based Hybrid LTL Verification Approach 44

4.3 Using SOGs for Checking Generic Properties **50**

4.3.1 Soundness 51

4.3.2 Relaxed, Weak and Easy soundness 61

4.4 Conclusion **62**

4.1 Introduction

The SOG is used during this work in order to represent each component of an IEBP allowing its abstraction (preserving the privacy) and the verification of the whole process. However, before analyzing the suitability of the SOG for the verification of the complete IEBP (the object of the next Chapter), it is necessary to study this structure from a local point of view: How the SOG can be used/adapted in order to check behavioral properties of each component locally? This Chapter introduces our first two contributions in this issue: (1) Adapting the SOG to allow the verification of hybrid LTL (where formulae can involve state- and event-based atomic propositions conjointly), and (2), proposing dedicated algorithms, using the SOG, for the verification of generic properties such as the soundness property and three of its variants (weak, relaxed and easy soundness). In the first contribution, the aggregation criterium of the SOG is presented as a mix of the aggregation criteria of the event and the state SOGs versions, while, in the second, new attributes are added to each aggregate in order to accomplish the verification of the soundness properties. Finally, for both contributions, and along this manuscript, the collaborative activities/actions (those allowing the interaction of a component with its

partners through both communication buffers and ressources) are observed while internal activities/actions are unobserved.

In this manuscript, the SOG will be defined on a behavior-based formalism (e.g. *LTS*, *KS* or *LKS*) and not on the model used to specify it, we will consider in the following that such a formalism is associated with an underlying RCoWF-net model. Given a RCoWF-net $N = \langle P, T, F, W, I, R, O \rangle$, the interface transitions are then defined by $Int = \{t \in T \mid ({}^\bullet t \cup t^\bullet) \cap (I \cup O \cup R) \neq \emptyset\}$. Depending on the property we are interested in, the SOG is built over a particular set of observed elements, namely *Obs*: In case of generic properties, it is based on the set of interface transitions $Obs = Int$. In case of (hybrid) $LTL \setminus X$ property, the observed elements contains *Int* in addition to the elements occurring in the formula to be checked. The unobserved transitions are defined by $UnObs = T \setminus Obs$.

4.2 Using SOGs for Hybrid LTL

In this section, we propose to adapt the SOG in order to abstract RCoWF-nets' behavior while preserving LTL formulae that involve a mix of state-based (the marking of some places) and event-based (transitions) atomic propositions. In consequence, we chose to represent the behavior of a BP model (e.g., RCoWF-nets) by an *LKS* over a set of atomic propositions AP and a set of actions Act. The main differences between the syntax and the semantics of hybrid LTL and the state-based LTL (see the Preliminaries Chapter) are the following:

- *Syntax*: any element of $AP \cup Act$ is a formula.

- *Semantics*: Each state of an infinite run $w = x_0 x_1 x_2 \ldots$ is assigned with a set of atomic propositions and a set of actions that are satisfied within that state. An action is said to be satisfied within a state if it occurs from this state. In our case (interleaving model of concurrency), where a single action can occur at a time, at most one transition can be assigned to a state of a run. For example, for a given marked Petri net, the run $(m(p_1) = 1 \wedge t_1).(m(p_2) = 1 \wedge t_2) \ldots$ is a run where, in the initial state, the marking of the place p_1 is equal to 1, and where the transition t_1 occurred leading to a marking where p_2 contains a token and where the transition t_2 occurred. We write w^i for the suffix of w starting from x_i and $p \in x_i$, for $p \in AP \cup Act$, when p is satisfied by x_i.

4.2.1 Revisiting SOG for Hybrid LTL

The adaption of the *SOG* to hybrid LTL leads to a new aggregation criterium (see the following definition): (1) states belonging to a same aggregate must have the same truth values of the state-based atomic propositions, and (2), the occurrence of an event-based atomic proposition from a state of an aggregate must lead to an other aggregate.

Definition 18 *Let $\mathcal{K} = \langle \Gamma, Act, L, \rightarrow, s_0, s_f \rangle$ be an LKS over a set of atomic propositions AP and let $Obs \subseteq Act$ be a set of observed actions of $\mathcal{S}$. An aggregate a of $\mathcal{K}$ w.r.t. Obs is a triplet $\langle S, d, l, f \rangle$ satisfying:*

- *$S \subseteq \Gamma$ where:*
  - *$\forall s, s' \in S,\ L(s) = L(s')$;*
  - *$\forall s \in S,\ (\exists (s', u) \in \Gamma \times (Act \setminus Obs) \mid L(s') = L(s) \wedge s \xrightarrow{u} s') \Leftrightarrow s' \in S$;*
  - *$\forall s \in S,\ (\exists (s', o) \in \Gamma \times Obs \mid s \xrightarrow{o} s') \wedge (\nexists (s'', u) \in S \times (Act \setminus Obs) \mid L(s'') = L(s') \wedge s'' \xrightarrow{u} s') \Leftrightarrow s' \notin S$.*
- *$d \in \{true, false\}$; $d = true$ iff S contains a dead state.*
- *$l \in \{true, false\}$; $l = true$ iff S contains an unobserved cycle (i.e., with unobserved transitions).*
- *$f \in \{true, false\}$; $f = true$ iff S contains a final state (i.e. $s_f \in S$).*

In addition to the original d, l and f attributes of an aggregate, the above definition first states that two states belonging to a same aggregate have necessarily the same label. It then specifies the states that must belong to an aggregate (the aggregation criterium) and those that must be excluded: (1) For any state s in the aggregate, any state s', having the same truth values of the atomic propositions and being reachable from s by the occurrence of an unobserved action, belongs necessarily to the same aggregate. (2) For any state s in the aggregate, any state s' which is reachable from s by the occurrence of an observed action is necessarily not a member of the same aggregate (even if it has the same label as s), unless the aggregation criterium includes it in the aggregate through an other state s'' of the aggregate.

Before defining the SOG, let us introduce the following operations:

- $SAT_{AP}(S)$: returns the set of markings that are reachable from any marking in S, by a sequence of unobserved transitions and which have the same value of the atomic propositions as S, and is defined as follows:

  $SAT_{AP}(S) = \{s'' \in \Gamma \mid \exists s \in S, \exists \sigma \in UnObs^*, s \xrightarrow{\sigma} s'' \wedge \forall s' \in \Gamma, \forall \beta\ prefix\ of\ \sigma, s \xrightarrow{\beta} s' \Rightarrow L(s) = L(s')\}$.

- $Out(a,t)$: returns, for an aggregate a and a transition t, the set of states outside of a that are reachable from some state in a by firing t, and is defined as follows:

  $$Out(a,t) \begin{cases} \text{if } t \in Obs & \{s' \in \Gamma \mid \exists s \in a.S, s \xrightarrow{t} s'\} \\ \text{if } t \in UnObs & \{s' \in \Gamma \mid \exists s \in a.S, s \xrightarrow{t} s' \wedge L(s) \neq L(s')\} \end{cases}$$

- $Out_\tau(a)$: returns, for an aggregate a, the set of states whose label is different from the label of any state of a, and which is reachable from some state in a by firing unobserved actions, and is defined as follows:

  $Out_\tau(a) = \bigcup_{t \in UnObs} Out(a,t)$.

- $Part_{AP}(S)$: returns, for a set of states S, the set of subsets of S that define the smallest partition of S according to the labeling function L, and is defined as follows:

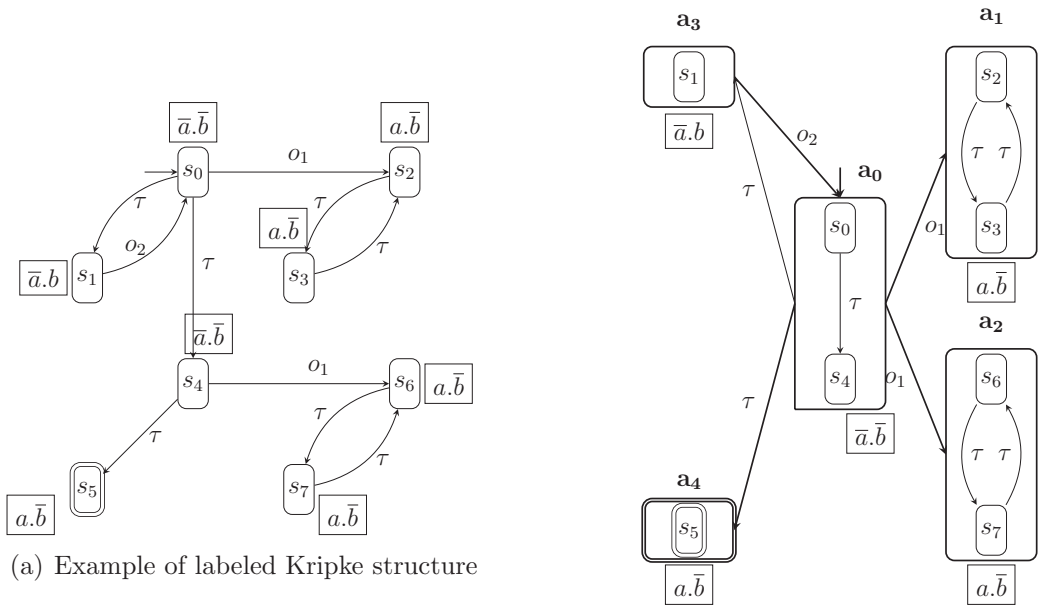
  $Part_{AP} : 2^\Gamma \longrightarrow 2^{2^\Gamma}$
  $Part_{AP}(S) = \{S_1, S_2, ..., S_n\} \Leftrightarrow S = \bigcup_{i=1}^{n} S_i \wedge \forall i \in \{1..n\}, \forall s, s' \in S_i, L(s) = L(s') \wedge \forall s \in S_i, \forall s' \in S_j, j \neq i, L(s) \neq L(s')$.

Definition 19 *Let $\mathcal{K} = \langle \Gamma, Act, L, \rightarrow, s_0, s_f \rangle$ be an LKS over a set of atomic propositions AP and let $Obs \subseteq Act$ be a set of observed actions of $\mathcal{K}$. The SOG associated with $\mathcal{K}$, over AP and Obs, is an LKS $\mathcal{G} = \langle A, Obs \cup \{\tau\}, L', \rightarrow', a_0, \Omega \rangle$ where:*

1. *A is a non empty finite set of aggregates satisfying :*
   - $\forall a \in A,\ \forall t \in Obs, \forall o_i \in Part(Out(a,t)), \exists a' \in A$ *s.t.* $a' = SAT_{AP}(o_i)$
   - $\forall a \in A,\ \forall o_i \in Part(Out_\tau(a)), \exists a' \in A$ *s.t.* $a' = SAT_{AP}(o_i)$
2. *$L' : A \rightarrow 2^{AP}$ is a labeling (or interpretation) function s.t. $L'(a) = L(s)$ for $s \in a.S$;*
3. *$\rightarrow \subseteq A \times Act \times A$ is the transition relation where:*
   - $((a,t,a') \in \rightarrow') \Leftrightarrow ((t \in Obs) \wedge (\exists o_i \in Part(Out(a,t))$ *s.t.* $SAT_{AP}(o_i) = a')$
   - $((a,\tau,a') \in \rightarrow') \Leftrightarrow (\exists o_i \in Part(Out_\tau(a))$ *s.t.* $SAT_{AP}(o_i) = a')$
4. *a_0 is the initial aggregate s.t. $s_0 \in a.S$.*
5. *$\Omega = \{a \in \mathcal{A} \mid s_f \in a.S\}$.*

The finite set of aggregates A of a SOG is defined in a complet manner so that the necessary aggregates are represented. The labeling function associated with a SOG gives to any aggregate the same label as its states. Point (3) defines the transitions relation: (1) there exists an arc, labeled with an observed transition t (resp. τ), from a to a' iff a' is obtained by saturation (using SAT_{AP}) on a set of equally labeled reached states ($Out(a,t)$

(resp. $Out_\tau(a)$)) by the firing of t (resp. any unobserved transition) from $a.S$. The last two points of Definition 19 characterize the initial aggregate (which contains the initial state of the LKS) and the set of final aggregates (i.e. any aggregate containing the final state) respectively.



(a) Example of labeled Kripke structure

(b) A corresponding SOG: $AP = \{a, b\}$ and $Obs = \{o_1, o_2\}$

Figure 4.1: An LKS and its SOG

Figure 4.1 illustrates an example of LKS (Figure 4.1(a)) over $AP = \{a, b\}$ and a corresponding SOG (Figure 4.1(b)) over $Obs = \{o_1, o_2\}$ (τ represents any unobserved action). The presented SOG consists of 5 aggregates $\{a_0, a_1, a_2, a_3, a_4\}$ and 5 edges. The initial aggregate a_0 is obtained by adding any state reachable from the initial state s_0 of the LKS, by unobserved sequences of actions only, and labeled similarly to s_0. Doing so, the initial aggregate contains s_4 but not s_1, neither s_5 which are labeled differently from s_0 (although reachable from s_0 by unobserved actions). State s_2, which is reachable from s_0 by an observed action (o_1), is immediately excluded from a_0 and belongs to a_1. The same holds for s_6 which is reachable from s_4 by o_1 and belongs to the aggregate a_2. s_3 (resp. s_7) is added to a_1 (resp. a_2) since it is reachable from s_2 (resp. s_6) by an unobserved action and since it is labeled similarly. Note that one can merge a_1 and a_2 because they have the same label. This is not the case for aggregates a_3 and a_4 which have different labels, although both are reachable from the same aggregate a_0 by the same action τ. None of the aggregates of the obtained SOG contain a dead state while a_1 and a_2 contains cycles (livelock) and a_4 is the unique final aggregate (it contains the final state s_5).

According to Definition 19, and similarly to the event- and the state-based SOGs, the SOG associated with an *LKS* is hence not unique. It can also be non deterministic since, for instance, an aggregate can have several successors with τ.

Algorithm 1 Building SOG with depth-first traversal

Require: an *LKS* $\langle \Gamma, Act, L, \rightarrow, s_0, s_f \rangle$
Ensure: Compute a SOG $\langle A, Act, L', \rightarrow', a_0, \Omega \rangle$

```
Aggregate a, a';
stack st;
Set of states S';
Set of actions E_s, E'_s;
a_0 = NewAgg(s_0, E_s);
->' = ∅;
E_s = E_s ∪ fireableObs(a_0)
st.push(⟨a_0, E_s⟩)
while (st ≠ ∅) do
  ⟨a, E_s⟩ = st.top()
  if E_s ≠ ∅ then
    t = E_s.next()
    S' = Img(a.S, t)
    a' = NewAgg(S', E'_s)
    if (a' is encountered for the first time) then
      E'_s = E'_S ∪ fireableObs(a')
      st.push(a', E'_s)
    else
      free a'
      Let a' be the already existing aggregate
    end if
    ->' = ->' ∪ {a -t-> a'}
  else
    A = A ∪ a'
    st.pop()
  end if
end while
```

Algorithm 1 builds a SOG $\langle A, Act, L', \rightarrow', a_0, \Omega \rangle$ associated with an *LKS* $\langle \Gamma, Act, L, \rightarrow, s_0, s_f \rangle$. Three functions are used in this algorithm:

- The $NewAgg()$ function requires a set of states S and a set of transitions E_s. S represents the input states of a new aggregate on which the aggregation criterium (see Definition 18) is applied. This allows to complete the aggregate with states that are reachable from S, with unobserved actions, while being labeled similarly to S. During this step, the function stores in E_s the set of unobserved actions that lead to states having a different label from S. These will be treated by Algorithm 1 to

Algorithm 2 Compute an aggregate

```
1: NewAgg(Set of states S, Set of events E_s)
2: Set of states From, To, Select;
3: From = S;
4: E_s = ∅;
5: repeat
6:    for t ∈ UnObs do
7:       To = Img(From, t)
8:       Select = To ∩ L(S)
9:       if To ≠ Select then
10:         E_s = E_s ∪ {t}
11:      end if
12:      from = Select \ S
13:      S = S ∪ Select
14:   end for
15: until From == ∅
16: return (ComputeAttr(S))
```

build the successor aggregates. $NewAgg()$ function allows also to compute (through $ComputeAttr()$ function) the other attributes of an aggregate i.e., d, l and f.

- The $fireableObs()$ function computes, for an aggregate a, the set of observed actions that are enabled by (some of) its states.
- The $Img()$ function allows to compute the immediate successors of a set of states by the firing of a given transition. The obtained states represent the input states of the new successor aggregate and will be then processed by $NewAgg()$ to achieve its construction.

Algorithm 1 processes through a depth first traversal manner in order to build the SOG. It contains two main steps: The first (lines $5-8$) initializes the SOG components and a stack (where elements are couples of aggregates, associated with the set of enabled transitions). The initial aggregate is obtained (line 5) by a call to the $NewAgg()$ function in order to build an aggregate from the initial state s_0 of the LKS. The transition relation is initially empty (line 6). In order to process the successors of the initial aggregate, the set of enabled transitions is completed with the observed ones (the unobserved enabled transitions are already stored in E_s by function $NewAgg()$) through function $fireableObs()$ (line 7). Finally (line 8), the initial aggregate and the corresponding enabled transitions are pushed in the stack st (line 9).

The second step of Algorithm 1 is the main loop (line$9-27$) where each iteration consists in picking (line 10) and processing an item (a, E_s) of the stack st. For each enabled transition in E_s (line 12), the immediate (state) successors are computed by

the $Img()$ function (line 13). The successor aggregate is then completed by $NewAgg()$ function (line 14). If the generated aggregate has not been encountered (line $15-17$), the SOG is updated with a new arc (line 22). Then, the set of enabled observed transition of the new aggregate is computed and a new couple is pushed into the stack to be processed in the next iteration. Otherwise (the successor aggregate has been already encountered), only the transition relation of the SOG is updated with a new arc (line 22).

The $Newagg()$ function is illustrated by Algorithm 2. It is a direct application of the aggregation criterium of Definition 18: Starting from a set of states S, all the states that are reachable from S, with unobserved actions, while being labeled similarly to S, are added to the aggregate. For any enabled unobserved transition, the set of successors are filtered basing on the label of S, denoted by $L(S)$ (line 8). Notice that we abusively consider $L(S)$ as a set of states (all those satisfying the label of S) since this can be done immediately by using the BDD structure. If an unobserved transition leads to some states with a different label (line 9), these must be excluded from the current aggregate (line 11) and the transition must be fired outside the aggregate. Thus, the $Newagg()$ function puts such transitions in the E_s set (line 10). Finally (line 16), the call to the $ComputeAttr()$ function computes the d, l and f attributes of an aggregate following Definition 18.

4.2.2 SOG-based Hybrid LTL Verification Approach

Using SOGs, one can deal with LTL properties that do not involve the next operator (X). Indeed, the aggregation criterium hides the immediate successors of states. However, we can use a special next operator whose interpretation would be the *observed next* (not necessarily the immediate successor). The equivalence between checking a given $LTL \setminus X$ formula on the new adapted SOG and checking it on the original LKS is ensured by the preservation of maximal paths (finite paths leading to a dead/final state and infinite paths). This corresponds to the CFFD semantics [65], which is exactly the weakest equivalence preserving next time-less linear temporal logic. The maximal paths of the original model are preserved by the SOG and characterized as follows:

Definition 20 (maximal paths of a SOG) *Let $\mathcal{G}$ be a SOG and $\pi = a_0 \xrightarrow{t_1} a_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} a_n$ be a path of $\mathcal{G}$. Then π is said to be* a maximal path *iff one (at least) of the four following properties holds:*

- $a_n.d = true$,
- $a_n.l = true$,
- $a_n.f = true$,

- $\exists 0 \leq m \leq n$ *s.t.* $a_m \xrightarrow{t_{m+1}} \cdots \xrightarrow{t_n} a_n$ *is a circuit.*

While the deadlock and the final attributes allow to detect finite maximal paths respectively, the livelock attribute allows to detect infinite runs involving infinitely often unobserved transitions. Notice that, on the contrary to the original definition of the SOG, such runs can be visible outside aggregates (e.g. an infinite unobserved run that continuously change the label of the traversed aggregates). However, as originally, the infinite runs involving infinitely often observed transitions are directly visible on the SOG structure. Since the SOG is finite, infinite runs can be expressed as runs ending into a circuit.

For sake of efficiency, the detection of dead states and cycles inside an aggregate are performed using symbolic operations (BDD-based set's operations) only. Thus, the symbolic observation graph preserves the validity of formulae written in classical Manna-Pnueli linear time logic [86] (LTL) from which the "next operator" has been removed (because of the abstraction of the immediate successors) (see for instance [106, 53]).

Since LTL is interpreted on infinite paths, the usual solution in automata theoretic approaches to check LTL formulae on a Kripke structure is to convert each of its finite paths to an infinite one by adding a loop on its dead states. Following the same approach, we define the *extended symbolic observation graph* (ESOG) as a transformation of the SOG allowing to capture all the maximal paths under the form of infinite runs. For this, we transform each finite maximal path (i.e. those ending into an aggregate with deadlock/livelock/terminal state) into infinite ones.

Definition 21 (Extended SOG) *Let* $\langle A', Obs \cup \{\tau\}, L', \rightarrow', a'_0, \Omega' \rangle$ *be a SOG over a set of observed actions* Obs *and a set of state-based atomic propositions* AP*. The associated ESOG is a an LKS* $\langle A, Obs \cup \{\tau\}, L, \rightarrow, a_0, \Omega \rangle$ *where:*

1. $A = A' \cup \{v \in 2^{AP} \mid \exists a \in A', L'(a) = v \wedge (a.d = true \vee a.l = true \vee a.f = true)\}$
2. $Act = Act' \cup \{true, dead, live, term\}$
3. $L : A \rightarrow 2^{AP}$ *is a labeling (or interpretation) function s.t.* $\forall a \in A', L(a) = L'(a)$ *and* $\forall v \in A \setminus A',\ L(v) = v$ *;*
4. $\rightarrow \subseteq A \times Act \times A$ *is the transition relation satisfying:*
   (a) $\rightarrow' \subseteq \rightarrow$
   (b) $\forall a \in A'$
      - $a.d = true \Rightarrow (a, dead, L(a)) \in \rightarrow)$
      - $a.l = true \Rightarrow (a, live, L(a)) \in \rightarrow)$

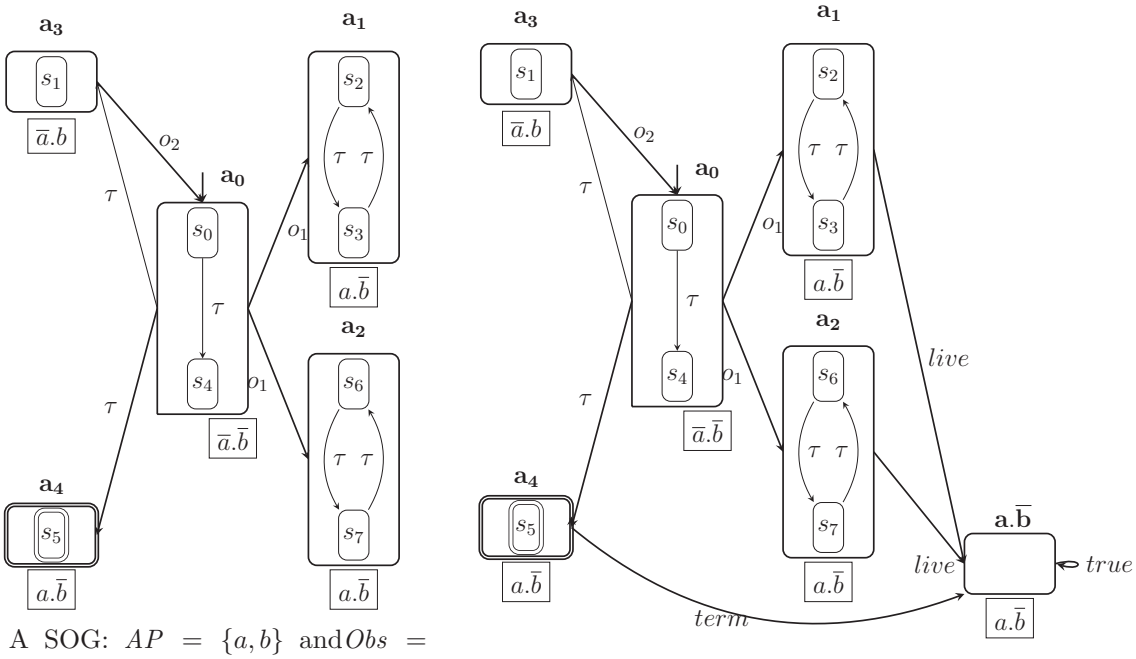
- $a.f = true \Rightarrow (a, term, L(a)) \in \rightarrow)$

(c) $\forall v \in A \setminus A',\ (v, true, v) \in \rightarrow$

5. $a_0 = a'_0$

6. $\Omega = \{a \in A \setminus A' \mid \exists a' \in A \wedge (a', term, a) \in \rightarrow\}$

The ESOG is obtained from a SOG by adding three actions *dead*, *live* and *term* representing deadlock, livelock and termination respectively. Each aggregate having one of these three features is connected, with the appropriate label, to a new aggregate having the same label (which is also its name). Each added aggregate has a self loop labeled with *true*. Moreover, the initial aggregate of the ESOG is the same as the SOG. Finally, the set of final aggregates contains any aggregate having a predecessor by the *term* action.



(a) A SOG: $AP = \{a, b\}$ and $Obs = \{o_1, o_2\}$

(b) A corresponding ESOG

Figure 4.2: A SOG and its corresponding ESOG

The extended SOG of Figure 4.2(b) is the obtained from the SOG of Figure 4.2(a). Since a_1 contains a livelock, a new aggregate having the same label as a_1 and called $a.\bar{b}$ (which is the label of a_1 as well). This new aggregate is a successor of a_1, by the *live* transition and has a self loop labeled with true. Aggregates a_2 and a_4 are concerned by the same reasoning since a_2 contains a livelock and a_4 contains a final state. However, the fact that both aggregates have the same label as a_1 implies that the added aggregate $a.\bar{b}$ is used as a successor of a_2 and a_4 by transitions *live* and *term* respectively.

Thus, the ESOG allows to explicitly represent the infinite executions and can be submitted to any existing LTL model checker. Moreover, it allows to write LTL formulae involving three new actions (*live*, *dead* and *term*) expressing, deadlock, livelock and termination respectively.

In the following, the SOG and the ESOG denote the same graph when it comes to check LTL formulae.

In conclusion, the following result establishes that an LKS satisfies an LTL formula iff the corresponding ESOG does. Moreover, once built, a SOG over a set of atomic propositions AP and a set of observed transitions Obs can be reused to check any $LTL \setminus X$ formula involving a subset of $AP \cup Obs$.

Theorem 4.2.1 *Let $\mathcal{K}$ be an LKS and let $\mathcal{G}$ be the corresponding SOG over the set of the observed transitions Obs and over a set of atomic propositions AP. Let φ be an $LTL \setminus X$ formula on a subset of $Obs \cup AP$. Then, $\mathcal{K} \models \varphi \Leftrightarrow \mathcal{G} \models \varphi$*

To prove Theorem 4.2.1, we will prove that the SOG $\mathcal{G}$ preserves the maximal paths of the corresponding LKS $\mathcal{K}$. We recall that maximal paths are any path π satisfying one of the following requirements:

1. $\pi = s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} s_n$ such that m_n is a dead/final state
2. $\pi = s_0 \xrightarrow{t_1} \cdots \xrightarrow{t_l} s_l \xrightarrow{t_{l+1}} \cdots \xrightarrow{t_n} s_n$ such that $s_l \xrightarrow{t_{l+1}} \cdots \xrightarrow{t_n} s_n$ is a circuit.

Before giving the proof of the preservation of maximal paths, let us present tow lemmas about the correspondence between paths of $\mathcal{K}$ and those of $\mathcal{G}$.

Lemma 4.2.2 *Let $\pi = s_1 \xrightarrow{t_2} s_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} s_n$ be a path of $\mathcal{K}$ and a_1 be an aggregate of $\mathcal{G}$ such that $s_1 \in a_1$. Then, there exists a path $a_1 \xrightarrow{t'_2} a_2 \xrightarrow{t'_3} \cdots \xrightarrow{t'_l} a_l$ of $\mathcal{G}$ and a strictly increasing sequence of integers $i_1 = 1 < i_2 < \cdots < i_{l+1} = n+1$ satisfying $\{s_{i_k}, s_{i_k+1}, \cdots, s_{i_{k+1}-1}\} \subseteq a_k.S$ for all $1 \leq k \leq l$.*

Proof 1 *We proceed by induction on the length of π. If $n = 1$, knowing that $s_1 \in a_1.S$ concludes the proof. Let $n > 1$ and assume that $a_1 \xrightarrow{t'_2} a_2 \xrightarrow{t'_3} \cdots \xrightarrow{t'_l} a_l$ and $i_1, \cdots, i_{l+1}$ correspond to the terms of the lemma for the path $s_1 \xrightarrow{t_2} s_2 \xrightarrow{t_3} \cdots \xrightarrow{t_{n-1}} s_{n-1}$. Then, $s_{n-1} \in a_l.S$. Let us distinguish two cases.*

(i) If $s_n \in a_l.S$ then the path $a_1 \xrightarrow{t'_2} a_2 \xrightarrow{t'_3} \cdots \xrightarrow{t'_l} a_l$ and the same sequence used for the path $s_1 \xrightarrow{t_2} s_2 \xrightarrow{t_3} \cdots \xrightarrow{t_{n-1}} s_{n-1}$ (i.e. $i_1, \cdots, i_{l+1}$), except $i_{l+1} = n$ which is replaced by $n+1$, satisfy the proposition.

(ii) If $s_n \notin a_l.S$ then, since $s_{n-1} \xrightarrow{t_n} s_n$, t_n is either an observed transition, or the truth values of the state atomic propositions in s_{n-1} and in s_n are different. In this case, by construction of the SOG, there exists an aggregate a_{l+1} such that $a_l \xrightarrow{t_n} a_{l+1}$ and

$s_n \in a_{l+1}$. As a consequence, the path $a_1 \xrightarrow{t'_2} a_2 \xrightarrow{t'_3} \cdots \xrightarrow{t_l} a_l \xrightarrow{t'_l = t_n} a_{l+1}$ and the sequence $i_1, \cdots, i_l, i_{l+1}, i_{l+1}+1$ (where a new element $i_{l+2} = i_{l+1}+1$) satisfy the proposition.

The next lemma shows that the converse also holds.

Lemma 4.2.3 *Let $\pi = a_1 \xrightarrow{t_2} a_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} a_n$ be a path of $\mathcal{G}$. Then, there exists a path $s_1 \xrightarrow{t_2} (s_2 \xrightarrow{*}_{a_2} s'_2) \xrightarrow{t_3} \cdots \xrightarrow{t_n} (s_n \xrightarrow{*}_{a_n} s'_n)$ of $\mathcal{N}$ satisfying $s_1 \in a_1.S$ and $s_n \in a_n.S$.*

Proof 2 *We consider π in reverse order and proceed by induction on its length. If $n = 1$, it is sufficient to choose a state $s_1 \in a_1.S$.*

If $n = 2$, we have to distinguish two cases.

- *If $a_1 \neq a_2$ then, by construction of the SOG, there exists a state $s_1 \in a_1.S$ and a state $s_2 \in a_2.S$ such that $s_1 \xrightarrow{t_2} s_2$. This path verifies the proposition.*
- *If $a_1 = a_2$, then there exists a circuit σ of a_1. Let $s_2 \in \overline{\sigma}$ (i.e. the set of states involved in σ). The path $s_2 \xrightarrow{*}_{a_1} m_2$ (where all the traversed states belong to a_1) satisfies the proposition.*

Let $n > 2$ and assume that $s'_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} s_n$ corresponds to the terms of the lemma for the path $a_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} a_n$. We know that $s'_2 \in a_2.S$. Here, four cases have to be considered. We denote by $Out(a) = \{s \in a.S \mid \exists t \in Act,\ \exists s' \notin a.S, s \xrightarrow{t} s'\}$

1. *If $a_1 \neq a_2 \wedge s'_2 \in Out(a_2)$ then, by construction of the SOG, we know that there exists a state $s_2 \in a_2.S$ such that $s_2 \xrightarrow{*}_{a_2} s'_2$ and a state $s_1 \in Out(a_1)$ such that $s_1 \xrightarrow{t_2} s_2$. The path $s_1 \rightarrow (s_2 \xrightarrow{*}_{a_2} s'_2) \xrightarrow{t_3} \cdots \xrightarrow{t_n} s_n$ verifies the proposition.*
2. *If $a_1 = a_2 \wedge s'_2 \in Out(a_2)$ then, by construction of the SOG, we know that a_1 contains a circuit σ. Let $s_1, s_2 \in \overline{\sigma}$ such that $s_1 \xrightarrow{t_2} s_2$. Since $s'_2 \in Out(a_1)$ then s'_2 is reachable from s_2 in a_1. In consequence, the path $s_1 \xrightarrow{t_2} (s_2 \xrightarrow{*}_{a_2} s'_2) \xrightarrow{t_3} \cdots \xrightarrow{t_n} s_n$ satisfies the proposition.*
3. *$a_1 \neq a_2 \wedge s'_2 \in Out(a_2)$ then $(a_2 \xrightarrow{t_3} a_2)$. Thus, there exists a circuit σ of a_2 reachable from some state $s_2 \in a_2.S$. Moreover, there exists $s_1 \in Out(a_1)$ such that $s_1 \xrightarrow{t_2} s_2$. Let $c \in \overline{\sigma}$. Let us distinguish the two following subcases:*
   (a) *If there exists $i > 2$ such that $s'_i \in a_i$ and $s'_i Out(a_i)$ then, let j be the smallest such an i. Then, s'_j is reachable in a_j from c. Hence, the path $s_1 \xrightarrow{t_2} s_2 \xrightarrow{*}_{a_2} c(\xrightarrow{+}_{a_2} c)^{j-1} \xrightarrow{*}_{a_j} s'_j \cdots \xrightarrow{t_n} s_n$ verifies the proposition.*
   (b) *If for all $i > 2$, $s'_i \notin Out(a_i)$ then the path $s_1 \xrightarrow{t_3} s_2 \xrightarrow{*}_{a_2} c(\xrightarrow{+}_{a_2} c)^{n-1}$ satisfies the proposition.*

4. $a_1 = a_2 \wedge s'_2 \notin Out(a_2)$ *then, by construction of the SOG, we know that* a_1 *contains a circuit* σ*. We also know that* $s'_2 \in \overline{\sigma}$ *by construction. Let* $s_1 \in \overline{\sigma}$ *such that* $s_1 \xrightarrow{t_2} s'_2$*. Then the path* $s_1 \xrightarrow{t_2} (s'_2 \xrightarrow{*}_{a_2} s'_2) \rightarrow \cdots \rightarrow b_n$ *satisfies the proposition.*

We are now in position to study the correspondence between maximal paths.

Lemma 4.2.4 *Let* $\pi = s_0 \xrightarrow{t_1} \cdots \xrightarrow{t_n} s_n$ *be a maximal path of* $\mathcal{K}$*. Then, there exists a maximal path* $\pi' = a_0 \xrightarrow{t'_1} \cdots \xrightarrow{t'_l} a_l$ *of* $\mathcal{G}$ *such that there exists a sequence of integers* $i_0 = 0 < i_1 < \cdots < i_{l+1} = n+1$ *satisfying* $\{s_{i_k}, s_{i_k+1}, \cdots, s_{i_{k+1}-1}\} \subseteq a_k \ \forall\ 0 \leq k \leq l$*.*

Proof 3 *If* s_n *is a dead/finale state then knowing that* $s_0 \in a_0.S$ *and using Lemma 4.2.2, we can construct a path* $\pi' = a_0 \xrightarrow{t'_1} a_1 \cdots \xrightarrow{t'_l} a_l$ *and the associated integer sequence corresponding to* π*. Because the last visited state of* π *belongs to* a_l*, the dead/final attribute of* a_l *is necessarily equal to true and* π' *is then a maximal path of the SOG. Now, if* s_n *is not a dead/final state then, one can decompose* π *as follows:* $\pi = \pi_1\pi_2$ *s.t.* $\pi_1 = s_0 \xrightarrow{t_1} s_1 \rightarrow \cdots \xrightarrow{t_{n-1}} s_{n-1}$ *and* $\pi_2 = s_n \xrightarrow{t_{n+1}} s_{n+1} \xrightarrow{t_{n+2}} \cdots \xrightarrow{t_{n+k}} s_{n+k}$ *(where* π_2 *is a circuit). Once again, applying Lemma 4.2.2 from* a_0*, one can construct a path* $\pi'_1 = a_0 \xrightarrow{t'_1} a_1 \xrightarrow{t'_2} \cdots a_o$ *corresponding to* π_1*. The corresponding path of* π'_2 *can be also constructed applying the same lemma. However, this path must be constructed from* a_o *if* $s_n \in a_o.S$ *or from a successor of* a_o *containing* s_n *otherwise. Let* $\pi'_2 = a_{b_1} \xrightarrow{t'_{b_1+1}} a_{b_1+1} \rightarrow \cdots a_{e_1}$ *be this path. Then, let us distinguish the following four cases: We denote by* $In(a, a') = \{s' \in a' \setminus a \mid \exists s \in a.S, s \rightarrow s'\}$*.*

1. *if* π'_2 *is reduced to a single aggregate* a *then* $\overline{\pi_2} \subseteq a$ *and, because* π_2 *is a circuit of* $\mathcal{K}$*, the livelock attribute of* a *is true. Then, the path* $\pi'_1\pi'_2$ *is maximal in* $\mathcal{G}$*.*

2. *else if* $a_{e_1} \xrightarrow{t'_{b_1}} a_{b_1} \wedge s_n \in In(a_{e_1}, a_{b_1})$ *then* π'_2 *is a circuit of* $\mathcal{G}$ *and* $\pi'_1\pi'_2$ *is a maximal path of* $\mathcal{G}$ *satisfying the proposition.*

3. *else if* $s_n \in a_{e_1}$ *(i.e* $a_{b_1} = a_{e_1}$*) then the path* $a_{b_1+1} \rightarrow \cdots a_{e_2}$ *is a circuit of* $\mathcal{G}$ *and* $\pi'_1 \xrightarrow{t'_{b_1}} a_{b_1} \xrightarrow{t'_{b_1+1}} a_{b_1+1} \rightarrow \cdots a_{e_2}$ *is a maximal path of* $\mathcal{G}$ *satisfying the proposition.*

4. *else, by construction of the SOG, there exists a successor of* a_{e_1} *containing* s_n*. Applying again Lemma 4.2.2 from this aggregate, we can construct a new path in* $\mathcal{G}$ *corresponding to* π_2*. Let* $a_{b_2} \xrightarrow{t'_{b_2+1}} a_{b_2+1} \rightarrow \cdots a_{e_2}$ *be this path. If we can deduce a circuit of* $\mathcal{G}$ *from this path applying one of the three above points, this concludes the proof. Otherwise, it is also possible to construct a circuit of* $\mathcal{G}$ *by linking* a_{e_2} *to* a_{b_1} *similarly to the point 2 and 3 above and deduce a circuit. If this is not the case, we can construct a new path corresponding to* π_2 *starting from a successor of* a_{e_2}*. Because the number of aggregates in* $\mathcal{G}$ *is finite, a circuit will be obtained.*

Notice that for all the above cases, a sequence of integers can be easily constructed from the ones produced by Lemma 4.2.2.

Lemma 4.2.5 *Let* $\pi' = a_0 \xrightarrow{t_1} \cdots \xrightarrow{t_n} a_n$ *be a maximal path of* $\mathcal{G}$*. Then, there exists a maximal path* $\pi = (s_0 \xrightarrow{*}_{a_0} s'_0) \xrightarrow{t_1} \cdots \xrightarrow{t_n} (s_n \xrightarrow{*}_{a_n} s'_n)$ *of* $\mathcal{K}$*.*

Proof 4 *Let* π' *be a maximal path reaching an aggregate* a_n *such that* $a_n.d = true \vee a_n.f = true \vee a_n.l$ *(either the dead/final or the livelock attribute is true). First, let us notice that the proof is trivial if the path* π' *is reduced to a single aggregate because dead/final state (resp. a state of a circuit of* a_0*) is necessarily reachable from* s_0*.*

Otherwise, using Lemma 4.2.3, there exists a path $\pi = e_0 \xrightarrow{t_1} (s_1 \xrightarrow{*}_{a_1} s'_1) \xrightarrow{t_2} \cdots \xrightarrow{t_n} s_n$ *of* $\mathcal{K}$ *satisfying* $s'_0 \in a_0.S$ *and* $s_n \in a_n.S$*. If* $s'_0 \in Out(a_0)$*, we have* $s_0 \xrightarrow{*}_{a_0} s'_0$ *since* a_0 *is obtained by saturation from* $\{s_0\}$*. Otherwise,* s'_0 *belongs to a circuit of* a_0 *and there exists in* $\mathcal{G}$ *an arc from* a_0 *to itself. This circuit can then be chosen to be reachable from* s_0 *during the construction of* π*. Finally, there exists a state* $s'_n \in a_n.S$ *such that* $s_n \xrightarrow{*}_{a_n} s'_n$*, where* s'_n *is a dead/final state (if* $a_n.d = true \vee a_n.f = true$*) or a state of a circuit of* a_n *(if* $a_n.l = true$*), because* a_n *is obtained by saturation from* $In(a_{n-1}, a_n)$*. Thus, the path* $(s_0 \xrightarrow{*}_{a_0} s'_0) \xrightarrow{t_1} (s_1 \xrightarrow{*}_{a_1} s'_1) \xrightarrow{t_2} \cdots \xrightarrow{t_n} (s_n \xrightarrow{*}_{a_n} s'_n)$ *satisfies the lemma.*

Now, if neither $a_n.d$ *(resp.* $a.f = true$*) nor* $a_n.l$ *is true, then by construction of the SOG,* $\pi' = a_0 \xrightarrow{t_1} \cdots \xrightarrow{t_l} a_l \xrightarrow{t_{l+1}} \cdots \xrightarrow{t_n} a_n$ *with* $a_l \xrightarrow{t_{l+1}} \cdots \xrightarrow{t_n} a_n$ *a circuit of* $\mathcal{G}$*. We distinguish two cases:*

1. *If* $\forall l \leq i \leq n, a_i = a_l$*. Using Lemma 4.2.3, we can construct a path of* $\mathcal{K}$*, namely* $\pi = e_0 \xrightarrow{t_1} (s_1 \xrightarrow{*}_{a_1} s'_1) \xrightarrow{t_2} \cdots \xrightarrow{t_l} b_l$ *corresponding to* $a_0 \xrightarrow{t_1} \cdots \xrightarrow{t_l} a_l$ *such that* e_0 *is chosen to be reachable from* s_0 *(similarly to the above case). Because* $a_l \xrightarrow{t_l} a_l$*,* a_l *contains a circuit and* s_l *can be chosen such that this circuit is reachable from* s_l*. This leads to the construction of a maximal path of* $\mathcal{K}$*.*

2. *Otherwise,* l *can be chosen such that* $a_l \neq a_n$ *and* $a_n \rightarrow a_m$*. From this decomposition of* π'*, Lemma 4.2.3 can construct a maximal path of* $\mathcal{K}$ *satisfying the current lemma.*

4.3 Using SOGs for Checking Generic Properties

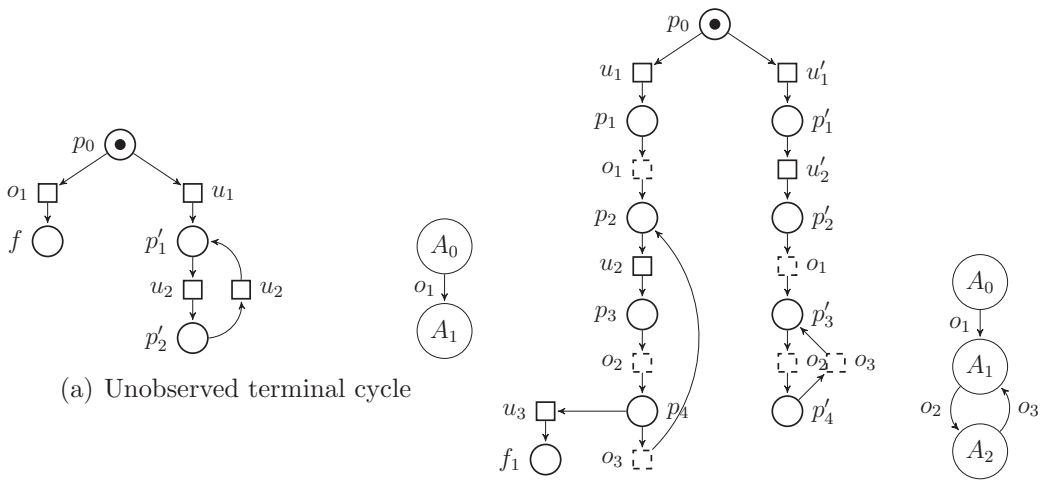
In this section we will show how checking generic properties on the original models can be reduced to the verification on their abstractions (SOGs). We are interested in the soundness property and three of its variants (weak, relaxed and easy soundness). Since the soundness properties have been defined on WF-nets, we use Petri net based models to illustrate the adaptation of the SOG structure to the verification of these properties.

We recall that in this section, the SOG is based on the observation of the set of interface transitions ($Obs = Int$). In this case the set of atomic propositions AP is the empty set and the SOG according to Definition 19 coincides with the event-based SOG [55].

We first establish that the current attributes of an aggregate of a SOG (i.e. dead, live and final) are not sufficient to check the soundness properties. Then, we give the necessary and sufficient information to be added to the aggregates in order to make it possible. Finally, we design dedicated verification algorithms based on the SOG graph.

4.3.1 Soundness

Given an aggregate a of a SOG, the l attribute, determining the presence/absence of a cycle inside a is completely useless regarding the soundness property since it does not allow to distinguish a terminal cycle from a non terminal one. Indeed, the presence of a terminal cycle that does not cross a final state violate the *option to complete* requirement of the soundness property, while a non terminal cycle can lead to a final state, and hence satisfy this requirement. In addition, the d attribute, determining the presence/absence of a dead state inside an aggregate, is not sufficient to allow one to check the same requirement. In fact, if the presence of a dead state implies the violation of the requirement, the absence of a dead state does not imply necessarily its satisfaction.



(a) Unobserved terminal cycle

(b) Observed terminal cycle

Figure 4.3: Terminal cycles preventing the detection of the option to complete requirement

Figure 4.3 illustrates two examples of terminal cycles where the system is deadlock free but the option to complete requirement is violated. The first one is illustrated by an unobserved terminal cycle (see Figure 4.3(a)) inside aggregate A_0 which is not a final aggregate (we consider that only the marking with a token in the place f is final). The

second example illustrates an observed terminal cycle (see Figure 4.3(b)) crossing the two aggregate A_1 and A_2. Although A_2 is a final aggregate (since it contains the marking where f_1 is marked), there exists a cycle (crossing the marking p'_4 and p'_3, belonging to A_3 and A_2 respectively) from which it is not possible to reach the final marking f_1.

Notice that using our model checker presented in the previous Section, and under a fairness assumption, one can check the *option to complete* requirement by checking the following LTL formula $\phi = G\ F\ term$ on the SOG. It expresses that, for any execution of the process, the *term* action occurs eventually in the future. We present in the following an other algorithmic solution in case the fairness property is not satisfied.

Regarding the *no dead transitions* requirement of the soundness property, it is clear that the firing of unobserved transitions is not detectable once the SOG is built. One should memorize, during the SOG construction, which unobserved transitions have not been fired. Finally, the *proper completion* condition requires to pick each state (a marking within an aggregate) individually and to compare it to the final state. It is clear that such a naive way of checking this condition would lead to bad performances and one must design a dedicated fully symbolic algorithm.

In order to solve these problems, we consider two new predicates of an aggregate: $M_f(a)$ which is the set of markings, in a, from which the final making is reachable, and $E_t(a)$ which contains the transitions that are enabled by (some) markings of a. At the end of this section, we supply a fully symbolic algorithm allowing to check the *proper completion* requirement. Such an algorithm exploits the BDD structure and avoids to pick each marking of an aggregate separately.

Definition 22 *Let $G = \langle A, Obs \cup \{\tau\}, L, \rightarrow, a_0, \Omega\rangle$ be a SOG corresponding to a marked RCoWF-net (N, m_0). The M_f and E_t predicates are defined as follows:*

- $M_f : A \longrightarrow 2^{R(N,m_0)}$
  $M_f(a) = \{m \in a.S \mid m_f \in R(N, m)\}$;
- $E_t : A \longrightarrow 2^{Act}$
  $E_t(a.S) = \{t \in Act \mid \exists m \in a.S \colon m \xrightarrow{t}\}$.

As for the *proper completion* condition, the computing of these two new attributes is accomplished in a pure symbolic way. While the computing of $E_t(a)$ for a given aggregate a is fairly trivial, the computing of $M_f(a)$ is not. In the following, we discuss the algorithm allowing this computing.

M_f Computing Algorithm

Given an aggregate a, the computing of the corresponding M_f necessitates to explore the future of the markings of a (by observed and unobserved transitions) and check whether

the final marking belongs to this future. One can compute this information by exploring the SOG once built. However, we chose to do it on-the-fly, during the construction of the SOG in order to be able to stop the construction as soon as the M_f of some aggregate is proved to be empty. Indeed, if such an aggregate exists, the *option to complete* condition of the soundness property is violated. Algorithm 3 updates Algorithm 1 (allowing to build a SOG) while emphasizing on the instructions related to the computing of M_f (underlined instructions). We are concerned with the computing of M_f at two points of the construction algorithm: when a new aggregate is encountered for the first time, and when the current aggregate is processed entirely (popped from the stack and stored in the SOG). First, when a new <u>final</u> aggregate a' is encountered for the first time (line $15-16$), we add to the corresponding M_f (which is empty) the set of states in a' from which the final state is reachable (by unobserved transitions). This is done by using the $SaturatePre()$ function (line 17). Next, once the processing of the current aggregate a (the construction of its future and the computing of the corresponding M_f is definitely done) is finished (lines $29-33$), two cases are considered: If the $M_f(a)$ is empty, then we are sure that none of the states of a allows to reach the final state. In this case, the *option to complete* requirement is not satisfied (and so is the soundness property), and, if desired, the construction of the SOG can be stopped.

Otherwise, the M_f of the immediate predecessor aggregate, if any, (which is in the top of the stack) is updated using the M_f of the current one. This is ensured by the $UpdateM_f()$ function. This function requires a source and a destination aggregates, src and tg respectively, and updates the $M_f(src)$ using the $M_f(tg)$. If t is the transition labeling the edge from src to tg, this function starts by getting the states of src enabling t (these will be added to $M_f(src)$), and complete $M_f(src)$ by a call to the $SaturatePre()$ function.

Notice finally that we assume that $M_f(a)$ is initialized by the empty set (this could be done by the $NewAgg()$ function) and stored as an attribute of each aggregate of the SOG.

Algorithm 3 Compute M_f while the construction of the SOG

Require: an *LKS* $\langle \Gamma, Act, L, \rightarrow, s_0, s_f \rangle$

Ensure: Compute a SOG $\langle A, Act, L', \rightarrow', a_0, \Omega \rangle$

```
Aggregate a, a';
stack st;
Set of states S';
Set of actions E_s, E'_s;
a_0 = NewAgg(s_0, E_s);
→' = ∅;
E_s = E_s ∪ fireableObs(a_0)
st.push(⟨a_0, E_s⟩)
while (st ≠ ∅) do
  ⟨a, E_s⟩ = st.top()
  if E_s ≠ ∅ then
    t = E_s.next()
    S' = Img(a.S, t)
    a' = NewAgg(S', E'_s)
    if (a' is encountered for the first time) then
      if (m_f ∈ a'.S) then
        a'.M_f = SaturatePre({m_f}, a')
      end if
      E'_s = E'_s ∪ fireableObs(a')
      st.push(a', E'_s)
    else
      free a'
      Let a' be the already existing aggregate
    end if
    →' = →' ∪ {a -t-> a'}
  else
    A = A ∪ a
    st.pop()
    if (a.M_f = ∅) then
      return false //Option to complete violated / may stop the SOG construction
    else
      UpdateM_f(st.top.first(), a) //only if st.top() exists i.e. a ≠ a_0
    end if
  end if
end while
```

BDD-based Algorithm for Proper Completion

To deal with the proper completion requirement, we implement a recursive algorithm (Algorithm 4) based on a parallel exploration of the BDDs B_a and B_f representing the states of an aggregate a and the final marking m_f respectively. For sake of simplicity, we consider safe Petri nets (i.e. the marking of any place is at most equal to 1). In this case each BDD variable corresponds to a place of the Petri net, and a path π in the BDD leading to the true (resp. false) node represents a marking m which (resp. does not) belongs to the underlying set of states. In the following, we abusively denote by π the corresponding marking and by π_i the marking of the place p_i number i in π.

The first call starts from the root of B_a and B_f BDDs and the exploration aims at finding a marking in B_a which is greater than the final marking in B_f as soon as possible. Thus, for each call of the algorithm, A (resp. F) designates the current node in B_a (resp. B_f) through a current path π_a (resp. π_f) in the same BDD. In a call number i, we have $\pi_{a_j} \geq \pi_{f_j}$, for any $0 \leq j \leq i-1$ (this has been ensured by the previous calls $i-1 \ldots ...1$). The current task (of the call i) is then to check the sub-marking $\pi_i \ldots \pi_n$ (assuming that there are n places in the Petri net). Moreover, the fact that m_f is unique, there is a unique path in B_f leading to the leaf *true* node, which is not necessarily the case of B_a. This has the consequence that when the current node in B_f is a leaf node, then the current node in B_a is necessarily of the same kind (a leaf). Notice that in Algorithm 4, if we are not sure that the current path in B_a leads to *true*, we are sure that the current one in B_f does. The current BDD nodes A and F are compared through the following cases:

- $A \neq F$: three cases are considered:
  - $A = true$ (line 2): This means that the current path in B_a is a marking belonging to the aggregate a. This means that F corresponds necessarily to some variable/place p_i and that the marking represented by the current path, completed by $m(p_j) = 1$, for any $i \leq j \leq n$, is a marking in a. Lines $3-9$ aim at finding, starting from place p_i, some unmarked place in m_f. If such a place exists, then we are sure that a contains a marking which is strictly greater than m_f and the algorithm returns *sup*. Else, the algorithm returns *equal* which means that the places p_j, for $i \leq j \leq n$ are also marked in m_f.
  - $A = false$ (lines $11-12$): This means that the current path in B_a does not correspond to a marking in the aggregate a. The algorithm returns $false$.
  - $A = v_i$ and $F = v_j$, with $i \neq j$: In this case, we are sure that $j < i$ (because m_f is unique), and the algorithm looks whether there exists a variable p_k (for $k = j \ldots i-1$) which is unmarked in m_f (lines $14-21$). If this is the case,

then the current path in B_a is strictly greater than the corresponding path in B_f (and the variable *strict* is set to *true* at line 16). Otherwise these places $p_j \dots p_{i-1}$ are marked in m_f, and the variable *strict* is not set to *true*. At the end of the loop, A and F are equal and both refer to the same BDD variable.

- $A = F$: If $A = true$ and $F = true$, the Algorithm returns *equal* without executing the body of the loop line 3. The case $A = false$ and $F = false$ is not possible since the current path in B_f is necessarily a path leading to *true*. Finally, if $A = F$ and both correspond to a place p_i, a recursive call is performed to pursue the parallel exploration of B_a and B_f. Let $(d_a, d_f) \in \{low, high\} \times \{low, high\}$ be the current exploration starting from the node p_i in both A and F. For instance, $(high, low)$ corresponds to a traversal in the right subgraph of A in parallel with a traversal of the left subgraph of F. Let us consider the two following cases:

  - When a $(high, low)$ traversal is finished with the result that the sub vector $(p_{i+1}, \dots, p_n)$ in B_a is greater or equal to the corresponding vector in B_f, then the algorithm ends after finding a marking in a which is strictly greater than m_f (lines $23 - 25$). Otherwise, the current explored path in B_a does not correspond to a marking in a (lines $27 - 28$), and one should explore another path in B_a, which is (low, low) (line 34).

  - When a $(high, high)$ (resp. (low, low)) traversal is required, we are sure that the marking of the places $p_0 \dots p_i$ is greater or equal to the corresponding vector in B_f. If such a traversal leads to the result that $(p_i \dots p_n)$ in B_a is strictly greater than the corresponding vector in B_f, then the algorithm ends by finding a marking in a strictly greater than the final marking (lines $35 - 36$). If the result is that $(p_i \dots p_n)$ in B_a is equal, then the final result depends on the value of the *strict* variable. If this is equal to true, then the algorithm returns *sup* (lines $35 - 36$), else it returns *equal* (line 38).

Algorithm 4 StrictGreater : Comparison of markings

Require: $Bdd\, A, Bdd\, m_f$

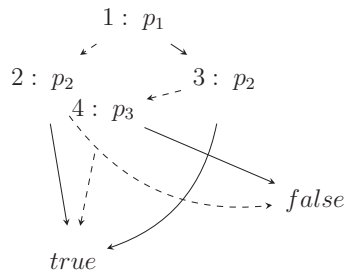
Ensure: $false, sup, equal$

```
bool strict = false
if A = true then
    while m_f ≠ True do
        if low(m_f) ≠ False then
            return sup
        end if
        m_f = high(m_f)
    end while
    return equal
end if
if A = False then
    return false
end if
while var(m_f) ≠ var(A) do
    if low(m_f) ≠ False then
        strict = true
        m_f = low(m_f)
    else
        m_f = high(m_f)
    end if
end while
if low(m_f)! = False then
    resRec = StrictGreater(high(A), low(m_f))
    if resRec = sup ∨ resRec = equal then
        return sup
    else
        Arec = low(A)
        mfrec = low(m_f)
    end if
else
    Arec = high(A)
    mfrec = high(m_f)
end if
resRec = StrictGreater(Arec, mfrec)
if resRec = sup ∨ (resRec = equal ∧ strict) then
    return sup
else
    return resRec
end if
```

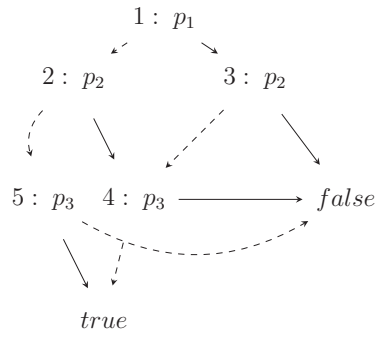
Finally, if the traversal call (from the variable p) returns false, then the path is not concluent and the algorithm returns $false$ to try an other path in B_a and B_f (line 38).

To conclude, the *proper completion* requirement is violated iff the Algorithm 4 returns *sup*.

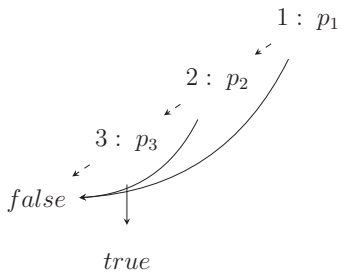
Let us illustrate the execution of Algorithm 4 on the BDDs of Figure 4.4. We assume a



(a) The BDD representing of the set of markings $\{p_1, p_2, p_2p_3\}$



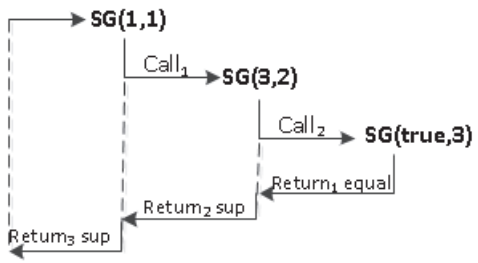
(b) The BDD representing of the set of markings $\{p_1, p_2, p_3\}$



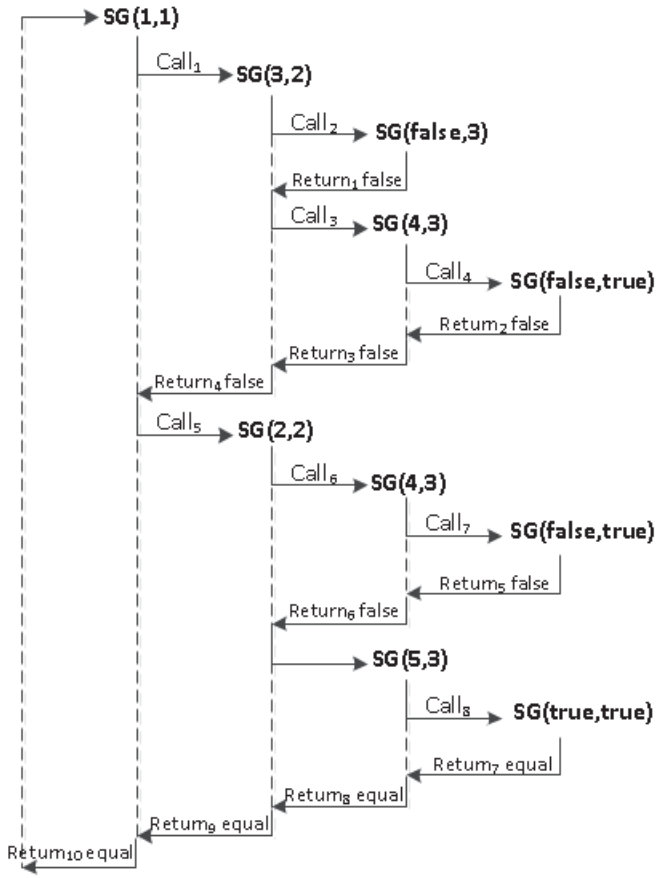
(c) The BDD representing the marking p_3

Figure 4.4: examples of BDDs

Petri net with three places p_1, p_2 and p_3, and a final marking m_f where only p_3 is marked. The corresponding BDD is presented in Figure 4.4(c). In Figure 4.4(a) and Figure 4.4(b), we consider the BDDs of two aggregates containing the sets of markings $\{p_1, p_2, p_2p_3\}$, $\{p_1, p_2, p_3\}$ respectively. It is clear that the first aggregate contains a marking (p_2p_3) which is strictly greater than m_f while the second does not. The processing of these two examples is illustrated by Figure 4.5(a) and Figure 4.5(b) respectively. The processing starts by the lefthand first call $StrictGreater(A, F)$ (SG(1,1) in the Figure), where A and F represent the roots of the aggregate and the final marking BDDs respectively. A left-to-right arrow from $SG(A, F)$ to $SG(A', F')$, where $(A', F) \in \{low(A), high(A)\} \times \{low(F), high(F)\}$



(a)



(b)

Figure 4.5: Illustration of two examples of Algorithm 4's execution

corresponds to a call executed by $SG(A, F)$. A return (right-to-left) arrow from $SG(A', F')$ to $SG(A, F)$ represents the result of the corresponding call.

- The processing of the first example is illustrated by Figure 4.5(a). In the first call ($SG(1, 1)$), and since $low_F(1) \neq false$ (line 22), a first recursive call ($call_1$) with $high_A(1)$ and $low_F(1)$ is done (line 23) leading to the execution of $SG(3, 2)$. The last call processes similarly, leading to the call ($call_2$) leading to the execution of $SG(true, 3)$. This call being done with a *true* leaf node in A, the body of the loop in

lines $3-8$ is executed once reaching the *true* leaf node of F. Thus, the $SG(true, 3)$ returns *equal* (line 9, $return_1$) to $SG(3, 2)$ which returns *sup* (lines $24-25$, $return_2$) (proving the existence, in the aggregate, of a marking strictly greater than m_f).

- The processing of the second example is illustrated by Figure 4.5(b). The execution starts similarly to the previous example until $call_2$ executed by $SG(3, 2)$ to $SG(false, 3)$. Since $A = false$ (lines $11-12$), $SG(false, 3)$ returns $false$ ($return_1$). Coming back to $SG(3, 2)$ (lines $27-28$) where a new call to $SG(4, 3)$ is performed (line 34, $call_3$). In $SG(4, 3)$, since $low_F(3) = false$ (line 22), a call to $SG(false, true)$ is done (line 34, $call_4$) which returns $false$ (lines $11-12$, $return_2$) to $SG(4, 3)$ which forwards this result (line 38, $return_3$) to $SG(3, 2)$. $SG(3, 2)$, in his turn, returns the same result to $SG(1, 1)$ (line 38, $return_4$). Coming back to the first call $SG(1, 1)$ (lines $27-28$), a final call to $SG(2, 2)$ (line 23, $call_5$) is performed. This call will perform two calls to $SG(4, 3)$ (line 23, $call_6$) and to $SG(5, 3)$ (line 34, $call_8$) which return $false$ and *equal* respectively. This result is returned successively to the first call $SG(1, 1)$, which returns the same result. Thus, there is no marking in the aggregate which is strictly greater than m_f.

Soundness On SOGs

Now we can completely characterize the soundness property using the SOGs.

Definition 23 *Let $\mathcal{G} = \langle \mathcal{A}, Obs \cup \{\tau\}, \rightarrow, a_0, \Omega' \rangle$ be a SOG associated with an RCoWF-net N. Then, G is sound iff the following requirements are satisfied:*

- *option to complete:* $\bigcup_{a \in \mathcal{A}} M_f(a) = \bigcup_{a \in \mathcal{A}} a.S$.
- *no dead transitions :* $\bigcup_{a \in \mathcal{A}} E_t(a.S) = T$.
- *proper completion:* $\forall a \in \mathcal{A}$ *Algorithm 4 applied to* $(a.S, m_f)$ *does not return sup.*

First, the *option to complete* requirement insures that a final marking is reachable starting from any reachable marking. After the construction of the SOG, and the compute of the M_f predicate for each aggregate (see Algorithm 3), this requirement is satisfied if the union of the sets $a.M_f$, $\forall a \in A$, is equal to the set of reachable markings of the system. The *option to complete* requirement is violated as soon as there is an aggregate a such that $M_f(a) = \emptyset$.

Second, the *no dead transition* requirement is checked after the construction of the SOG: Using the set of enabled transitions E_t of an aggregate, which is computed on the fly, one can check if all the transitions have been fired or not (by comparing the union of E_t with the set of transitions T).

Finally, the *proper completion* requirement is checked during the construction of the SOG by applying the Algorithm 4. This property is violated only if the Algorithm return *sup*.

Theorem 4.3.1 *Let N be an RCoWF-net, and let $\mathcal{G}$ be its corresponding SOG. Then, N is sound iff G is sound.*

4.3.2 Relaxed, Weak and Easy soundness

First the Relaxed soundness is a variant of soundness property which requires that each transition should occur in at least one "good" execution path that leads to a final marking. With the classical definition of SOG, one problem can arise when a terminal cycle (or dead state) belongs to an aggregate a : assume that such an aggregate leads to a final aggregate. Moreover, it could exist an unobserved transition t which is enabled only by a marking m inside this aggregate leading to the terminal cycle (or the dead state). However, if we suppose that m do not belong to another aggregate $a' \neq a$, we can not assert whether this unobserved transition occurs somewhere else in a "good" execution or not. For instance, in Figure 4.3(a), the unobserved action u_2, which is enabled by a marking in A_0, do not belong to any path leading to the final marking (where only the place f is marked). Therefore, to resolve this problem, we define a new predicate, denoted $T_f(a)$, which represents the set of transitions (observed or not), involved in any correct execution (ending at a final marking), starting from some special states inside a. This set of marking in $a.S$, from which some final making is reachable, is represented by the attribute M_f previously added to the aggregate.

Definition 24 *Let $G = \langle A, Obs \cup \{\tau\}, L, \rightarrow, a_0, \Omega\rangle$ be a SOG corresponding to a marked RCoWF-net (N, m_0). The T_f predicate is defined as follows:*

- $T_f : A \longrightarrow 2^T$
  $T_f(a) = \{t \in UnObs \mid Succ(M_f(a), t) \cap M_f(a) \neq \emptyset\} \cup Enable(M_f(a)) \cap Obs$ *where* $Succ(S, t) = \{s' \mid \exists s \in S\colon s \xrightarrow{t} s'\}$ *the set of states reachable from any state of S by the firing of t.*

Second, in some cases, it has to be guaranteed that the termination of a workflow occurs eventually. As long as a final marking is reachable from any state, any possible deadlock is allowed. That is we called weak soundness variant. Finally, the easy soundness is less exigent. It represent the case in which we accept that there exists at least only a final marking reachable from the initial state, so that we have at least one "good" execution of the workflow. For checking these two variants, the definition is based on the attribute M_f (already defined in the previous section).

We note that, for all three variants of the soundness properties, we will not detail the corresponding algorithm since we consider that they are too trivial properties.

Theorem 4.3.2 *Let $\mathcal{G} = \langle \mathcal{A}, Obs \cup \{\tau\}, \rightarrow, a_0, \Omega' \rangle$ be a SOG associated with an RCoWF-net N*

- *N is relaxed sound iff $\bigcup_{a \in \mathcal{A}} T_f(a) = T$.*
- *N is weak sound iff $\bigcup_{a \in \mathcal{A}} M_f(a) = \bigcup_{a \in \mathcal{A}} a.S$*
- *N is easy sound iff $\bigcup_{a \in \mathcal{A}} M_f(a) \neq \emptyset$*

4.4 Conclusion

Checking properties (insure a correct behavior) on the original models of inter-entreprise business processes can be reduced to the check on their abstractions (SOGs). In this chapter we had revisited the SOG structure to insure that it contains sufficient and necessary information to check the properties. We have interested in specific properties that can be expressed in temporal logic (LTL), and generic properties such as soundness properties (with most variants). In the next Chapter, we will adapt the SOG abstraction in order to show how we can analyze the composition of workflows using their corresponding SOGs.

Chapter 5

Using SOGs for Modular verification

Contents

5.1 Introduction **63**

5.2 Composition of SOGs **64**

5.2.1 The observed behavior 64

5.2.2 Synchronous composition of SOGs 68

5.2.3 Composition of RCoWF-nets' SOGs 74

5.3 Modular verification **81**

5.3.1 LTL-based Properties 81

5.3.2 Checking Soundness Properties 83

5.3.3 Soundness 83

5.3.4 Relaxed, Weak and Easy Soundness 86

5.4 Conclusion **88**

5.1 Introduction

In this Chapter, we will define how we compose two (ore more) business processes (each ignoring internal details about the other), and how we check generic and/or specific properties which are satisfied by each component locally. It is well known that correction properties like deadlock-freeness, soundness and LTL formulae are not preserved by composition. This is mainly due to the interaction between components which can lead to interlocks. An interlock is a "global" dead state where the whole process is locked, and it can arise even if the different constituents of the process are separately deadlock free. For instance, an interlock occurs when two processes send a request to each others, and each of them is waiting for a response from the other.

5.2 Composition of SOGs

Given two RCoWF-nets $\mathcal{N}_1$ and $\mathcal{N}_2$ that have been analyzed locally and proved to be correct (w.r.t. soundness notions or w.r.t. some LTL formulae), our goal is to reduce the verification of their original composition (which is not available anyway) to the verification of the composition of the corresponding SOGs $\mathcal{G}_1$ and $\mathcal{G}_2$, namely $\mathcal{G}_1 \oplus \mathcal{G}_2$. Such an approach presents several advantages: First, the verification of the composition takes into account the local verification process. We only focus on the common activities between the processes to be composed. The main task at this stage is to check whether, due to the composition, the nice properties that have been checked locally are violated after composition. Second, such an approach allows to reduce the state space explosion due to the composition. Finally, by abstracting a business process with a SOG, we hide the local behavior of the process which would represent internal organization and private information. This allows to respect the privacy feature of the enterprise and to avoid to expose irrelevant or sensitive information.

The main difficulty of our approach is to adapt the SOGs so that they contain the necessary and sufficient information making the composition of SOGs representative of the composition of the corresponding models (w.r.t. the desired properties). In particular, one must detect interlocks which represent an important behavior in the decision procedure of both soundness and LTL formulae. To reach this goal, a new attribute, called the observed behavior of an aggregate a, and denoted by $\lambda(a)$, will be added to the aggregates of a SOG. We are interested in the *observed behavior* of each state s belonging to an aggregate : (1) could s lead to the firing of some observed transitions in the future? (2) could s lead to a final state or a dead state in the future?

5.2.1 The observed behavior

The *observed behavior* of each state s inside an aggregate contains a set of observed transitions: It contains all the observed transitions that can be fired in the future of s via (a possibly empty) sequence of unobserved transitions. The observed behavior of an aggregate is then defined as the set of the observed behaviors of its states (i.e. a set of sets of observed transitions). In order to distinguish dead states from final states, a new *virtual* observed transition, called **term**, is considered s.t. it belongs to the observed behavior of any state from which a final state is reachable via (a possibly empty) sequence of unobserved transitions.

Definition 25 *Let* $N = \langle P, T, F_p \cup F_r \cup F_c, W, I, O, R\rangle$ *be an RCoWF-net. Let* m_0 *and* m_f *be the corresponding initial and the final markings and let* $\mathcal{G} = \langle A, Obs \cup \{\tau\}, L, \rightarrow, a_0, \Omega\rangle$ *be a corresponding SOG over the set of observed transitions*

Obs and a set of atomic propositions AP. The observed behavior is progressively defined by :

1. $\lambda_{\mathcal{N}} : R(N^*, m_0) \rightarrow 2^{Obs \cup \{\tau\}}$

   - $\forall t \in Obs,\ t \in \lambda_{\mathcal{N}}(m) \Leftrightarrow \exists m' \in R(N^*, m)$ *s.t.* $m \xrightarrow{\sigma} m' \xrightarrow{t}$ *where* $\sigma \in UnObs^*$,
   - $term \in \lambda_{\mathcal{N}}(m) \Leftrightarrow \exists t_1 \ldots t_n \in UnObs$ *s.t.* $m \xrightarrow{t_1} m_1 \ldots m_n \xrightarrow{t_n} m_f$ *and* $L(m) = L(m_i) = L(m_f)$, *for* $i = 1 \ldots n$.
   - $\tau \in \lambda_{\mathcal{N}}(m) \Leftrightarrow \exists t_1 \ldots t_n \in UnObs$ *s.t.* $m \xrightarrow{t_1} m_1 \ldots m_{n-1} \xrightarrow{t_n} m_n$ *and* $L(m) = L(m_i)$ *for* $i = 1 \ldots n-1$ *and* $L(m) \neq L(m_n)$.

2. $\lambda_{\mathcal{N}} : 2^{(R(N^*, m_0))} \rightarrow 2^{2^{Obs \cup \tau}}$
   $\lambda_{\mathcal{N}}(S) = \{\lambda_{\mathcal{N}}(m) \mid m \in S\}$

3. $\lambda_{min} : 2^{R(N^*, m_0)} \rightarrow 2^{2^{Obs \cup \{\tau\}}}$
   $\lambda_{min}(S) = \{X \in \lambda_{\mathcal{N}}(S) \mid \nexists Y \in \lambda_{\mathcal{N}}(S) : (Y \subset X) \wedge (Y \cap \{term, \tau\} = X \cap \{term, \tau\})\}$

4. $\lambda : A \rightarrow 2^{2^{Obs \cup \{\tau\}}}$
   $\lambda(a) = \lambda_{min}(a.S)$

Informally, for each marking m in $R(N^*, m_0)$, the observed behavior of m, $\lambda_{\mathcal{N}}(m)$, represents the set of observed actions, possibly completed with τ and/or *term*. An observed action t belongs to $\lambda_{\mathcal{N}}(m)$ when it is possible to fire t from m, possibly via a sequence of unobserved actions while traversing equally labeled states. τ belongs to $\lambda_{\mathcal{N}}(m)$ when the firing of some unobserved transition t' from m, possibly via a sequence of unobserved actions (traversing equally labeled states), leads to a marking m' labeled differently from m. *term* is a member of $\lambda_{\mathcal{N}}(m)$ iff the final marking is reachable from m using an unobserved sequence of actions σ (traversing equally labeled states). The observed behavior $\lambda_{\mathcal{N}}$ associated with a set of markings S contains the observed behavior of the markings of S. Actually, in order to detect interlocks (see next subsection), it is not necessary to keep in the observed behavior λ any set X if there exists a subset $Y \subset X$ in λ while Y contains the same information regarding τ and *term*. Thus, the observed behavior mapping λ_{min} applied to a set of markings S is defined as the set of minimal subsets (w.r.t. the set inclusion relation) of $\lambda_{\mathcal{N}}(S)$ preserving τ and *term*. Finally, the observed behavior $\lambda(a)$ associated with an aggregate a is the observed behavior λ_{min} applied to the corresponding set of states $a.S$.

Algorithm 5 Computing the Observed Behavior

Require: $Agregate\, a,\ Obs,\ UnObs,\ Final\ state\ m_f$
Ensure: $\lambda(a)$

```
Map < Set of actions, Set of states > R
if m_f ∈ a.S then
   insert ({term}, PreIm(m_f, a.S, UnObs)) in R
end if
for u ∈ UnObs do
   S ∪ PreIm(Out(a, u), a.S, {u})
end for
if(S ≠ ∅) then insert ({τ}, S) in R endif
for o ∈ Obs do
   if Enable(a.S, o) ≠ ∅ then
      insert ({o}, Enable(a.S, o)) in R
   end if
end for
R_N = ∅
while R_N ≠ R do
   R_N = R
   for (O, S) ∈ R_N, (O', S') ∈ R_N do
      if (S ∩ S') ≠ ∅ then
         S_old ← S; S ← S \ S'; S'_old ← S'; S' ← S' \ S
         if(S == ∅) then remove (O, S) from R_N endif
         if(S' == ∅) then remove (O', S') from R_N endif
         if (S == ∅) ∧ (S' == ∅) then
            add (O ∪ O', S_old) in R_N
         else
            if (O ∩ {τ, term}) ≠ (O' ∩ {τ, term}) then
               add (O ∪ O', S_old ∩ S'_old) in R_N
            end if
         end if
      end if
   end for
end while
λ(a) ← Set of keys of R; Set of states E ← ∅
for t ∈ (Obs ∪ UnObs) do
   E ← E ∪ EnableMarking(S, {t})
end for
if (E ≠ S)∨((PreIm*(EnableMarking(a.S, Obs)∪{m_f}, a.S, UnObs) ≠ a.S)) then
   λ(a) ← λ(a) ∪ {∅}
end if
```

From now on, a state (marking) m is said to be dead if and only if its observed behavior is the empty set. This generalizes the original definition of a dead state since a terminal cycle (a cycle from which no observed action is enabled in the future) is considered as a

deadlock as well. The d attribute of the originally defined aggregate (Definition 18) is then no more useful since a dead state can be detected locally when its observed behavior is $\emptyset$. The same holds for the f attribute of an aggregate a (stating that the final marking belongs to a), since, in this case, the $term$ element belongs to some set in $\lambda(a)$. Thus, in the remaining part of this Chapter, the d and the f attributes of an aggregate a are replaced by $\lambda(a)$.

A direct implementation of the observed behavior of a given set of states (following Definition 25) implies to consider each state belonging to the set separately. This would considerably decrease the efficiency of the approach. In fact, each aggregate is encoded with a BDD and all the operations manipulating the aggregates should be based on set operations. Therefore, we have implemented an algorithm (Algorithm 5) for the computing of the observed behavior that is exclusively based on set operations applied to the states of a given aggregate.

The input of Algorithm 5 are an aggregate a, the set of observed transitions Obs, the set of unobserved transitions $UnObs$ and the final state m_f. It computes the observed behavior associated with the aggregate a (i.e., $\lambda(a)$). We use a map (called R) whose elements are couples of sets of transitions and sets of states (line 1). Each element (O, S) eventually satisfies the following: each state of S enables only the transitions of O. This map is progressively updated so that, at the end of the algorithm, the set of its keys (the first element of the couples) forms the observed behavior of the aggregate a (line 38). The first step of the algorithm (lines $2-4$) consists in: (1) checking whether the final state belongs to $a.S$, (2) if it is the case creating a new couple $(\{term\}, S)$ where S is the set of the immediate predecessors, in $a.S$, of the final state by firing unobserved actions. The latter task is performed by using the $PreIm()$ function. The second step of the algorithm (lines $5-8$) allows to fill the map R with couples of the form $(\{\tau\}, S)$, where S is the non empty set of immediate predecessors of the output states of a (i.e. which have a successor outside of a by firing unobserved actions). After that, (lines $9-13$) the map R is filled with couples of the form $(\{o\}, S)$ where o is an observed action and S the subset of states of a enabling o. Once the map R is filled, it is analyzed in the forth part of the algorithm (lines $15-31$). For any two couples (O, S) and (O', S') in R, we consider only the case where $(S \cap S') \neq \emptyset$. Indeed, in this case, elements in $(S \cap S')$ enable any transition of $(O \cup O')$, and (O, S) must be updated by $(O, S \setminus (S \cap S'))$, which is done line 19. If $S \subseteq S'$, then $S \setminus (S \cap S')$ is empty and the couple (O, S) must be removed (lines 20). We do the same for (O', S') (lines 19 and 21). The question then is to detect when one must add the new couple $(O \cup O', S \cap S')$ in R. It is done in two cases: when $S = S'$ (lines $22-23$), and, according to Definition 25, when $(O \cap \{\tau, term\}) \neq (O' \cap \{\tau, term\})$ (lines $25-26$). Finally, the analysis of the elements of R finish when no more update is

possible (the use of R_N).

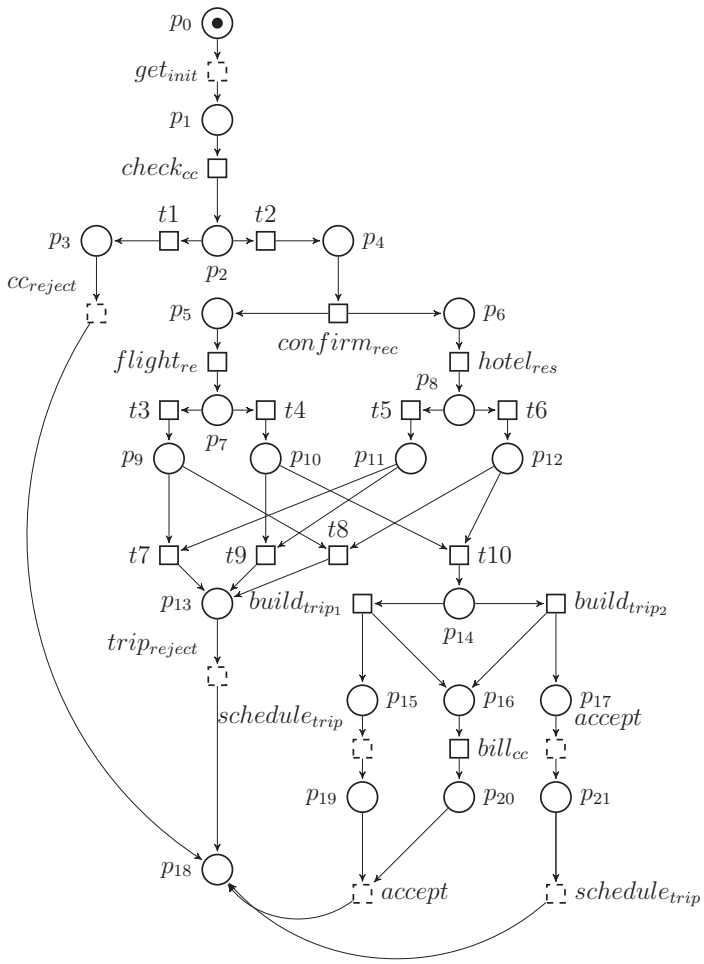
The final part of the algorithm (lines $33-38$) allows to decide whether the empty set must be added to $\lambda(a)$ or not. It actually should be added in two cases: When there is a deadlock state or when there is a terminal loop inside a. The dead state is detected (lines $38-41$) when the set of states that enable at least one transition is different from $a.S$. The terminal loop is detected when, starting from the set of states enabling observed actions and from the final state, there exists states in a that are not reachable by unobserved actions. We use iteratively use $PreIm()$ in order to detect such set of states (line 36).

Starting from several processes which communicate synchronously, asynchronously or by sharing resources within an IEBP, we show, in the following, how to compose the corresponding SOGs so that the obtained graph satisfies the property to be checked if and only if the whole IEBP satisfies it. Our solution to tackle this problem is based on a synchronized product between several SOGs. Thus, in the next part of this section, we first deal with a synchronous communication before the general case which possibly combine the other kinds of communication.

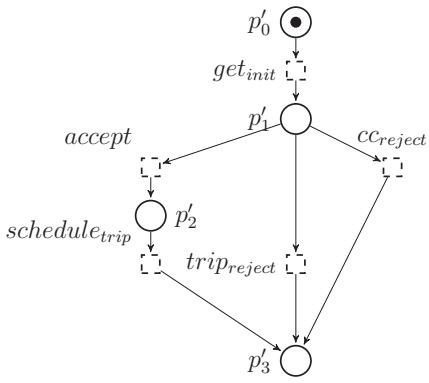
5.2.2 Synchronous composition of SOGs

A synchronous composition between two processes of an IEBP, sharing a subset of actions, involves, for each shared action, a "rendez-vous" between the two processes. The whole process can perform such an action only if both processes are separately able to execute it. In terms of Petri nets, it is about two models sharing a subset of transitions and the whole model is obtained by merging these transitions. To illustrate the problem of the non preservation of properties by synchronous composition, let us consider two RCoWF-nets of two business processes (taken from [87]) modeling the trip reservation and a possible costumer (see Figure 5.1). The set of input/output places and the set of resource places being both empty, the obtained model is then a WF-net. It shows the planning of a trip by a travel agency collaborating with a customer. Figure 5.1(a) illustrates the WF-net associated with the trip reservation's process while Figure 5.1(b) illustrates the one of the consumer.

Figure 5.1(a) illustrates the WF-net associated with the trip reservation's process that includes the reservation of a flight and booking of a room. When a request is received (get_{init}), the credit card information is checked ($check_{cc}$). The information can be not valid (t_1) then the request is rejected (cc_{reject}). Otherwise, a room and a flight are searched ($flight_re$ and $hotel_{res}$). If no room is available (t_3) or no flight is available (t_5) then the request is rejected ($trip_{reject}$). If there are a room and flight available then the trip is booked. But, there are two possible behaviors starting from this point: Either, the schedule is ready to be sent to the customer immediately (for example, a similar schedule



(a) WF-net of trip reservation



(b) WF-net of customer

Figure 5.1: The WF-nets of of a trip reservation and a costumer

has been sent to another customer) or not. In the first case, it is sent ($schedule_{trip}$) before charging the bank account of the consumer ($bill_{cc}$) and an acceptation is sent to the consumer (*accept*). Otherwise, the acceptation is sent to the customer before the schedule

(which can be prepared in parallel with the bill preparation).

Figure 5.1(b) illustrates the WF-net associated with a customer's process. First, the customer selects a trip program that he is interested in and provides the needed information for the reservation including the credit card information (get_{init}). Then, three cases can occur: (1) The request can be rejected when the credit card information are not valid (cc_{reject}), (2) the request can be rejected when there are no available rooms or flights ($trip_{reject}$), and (3) the can be accepted (accept) and the information about the trip is given ($schedule_{trip}$).

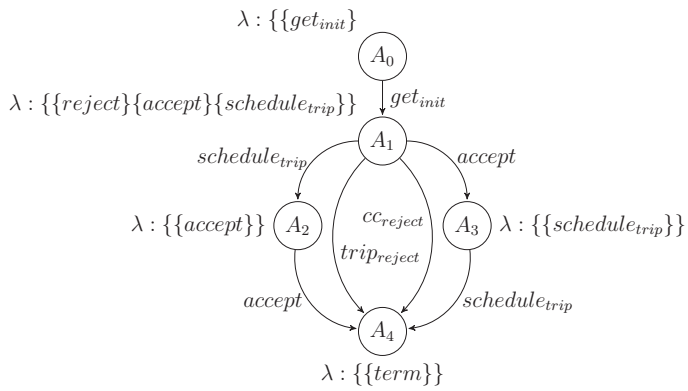
The first WF-net contains a big proportion of unboserved actions (17/24), while, in the second, all actions are observed. The two processes can collaborate by merging related transitions (the dashed transitions) to form an IEBP.

These two WF-nets are sound. LTL can be used to express one or several specific properties to guarantee some required behaviors of each WF-net. For instance, on the customer side, the two following properties can be of interest: $\phi_1 = G(get_{init} \implies F(accept \vee reject))$ and $\phi_2 = G(accept \implies F\ schedule_{trip})$. ϕ_1 means that each time the customer sends a trip request (via transition get_{init}), he/she will eventually receive either a positive (by transition accept) or a negative (by transition reject) answer. ϕ_2 means that each received positive answer is followed eventually by a description of the required trip. Both formulae are satisfied by the customer and the trip reservation processes when checked locally. In this example, the formulae are expressed using interface transitions only, but one can imagine other examples where local atomic propositions/transitions are involved in the desired formula.

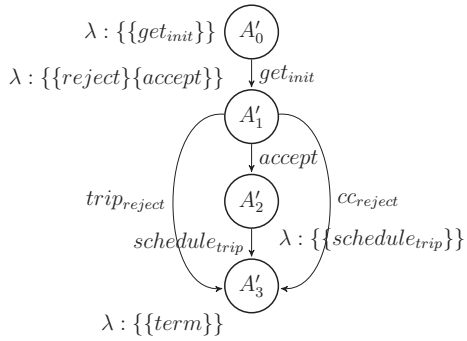
Figure 5.2 shows the two SOGs associated with the WF-nets of Figure 5.1. Figure 5.2(a) illustrates the SOG of the reservation trip model while Figure 5.2(b) shows the SOG of the customer model. Here, the $reject$ transition (either as a member of the observed behavior or as a label of an edge) stands for both transitions $trip_{reject}$ and cc_{reject}. We note that none of the aggregates of both SOGs contains a deadlock ($\emptyset$ is not a member of $\lambda(a)$ for any aggregate a), and each SOG has only one terminal aggregate, A_4 and A'_3 respectively (the $term$ action belongs to some elements of their observed behavior). It is clear, through this example, that bigger is the number of observed transitions, smaller is the size of the obtained SOG. When all the transitions are observed, the SOG is isomorphic to the reachability graph.

Before defining the composition of two SOGs (the synchronized product), we first show how the attributes of an aggregate a, resulting from the composition of two aggregates $a1$ and $a2$, are deduced from the (locally computed) attributes of a_1 and a_2. The aggregate product between n aggregates for $n > 2$ can be constructed by iterative multiplication.

Definition 26 (aggregate product)



(a) A SOG of trip reservation



(b) A SOG of customer

Figure 5.2: Two SOGs of the running example models

Let $\mathcal{G}_i$, for $i = 1, 2$, be two SOGs associated with two WF-nets. Let $a_i = \langle S_i, l_i, \lambda_i \rangle$ be two aggregates of two associated SOGs. The product aggregate *$a = \langle S, l, \lambda \rangle$, denoted by $a_1 \oplus a_2$, is defined as follows:*

- $S = S_1 \times S_2$;
- $l = l_1 \vee l_2$
- $\lambda = \{(x \cap y) \cup (x \cap ((Obs_1 \cup \{\tau\}) \setminus Obs_2)) \cup (y \cap ((Obs_2 \cup \{\tau\}) \setminus Obs_1)) \mid \forall x \in \lambda_1,\ \forall y \in \lambda_2\}$.

Although the set of states of a product aggregate $a_1 \oplus a_2$ is defined as the product of the sets of states of a_1 and a_2, S_1 and S_2 have not to be stored explicitly. Once the SOG is built, it will not play any role in the composition process. Then, $a_1 \oplus a_2$ contains a loop if and only if a_1 or a_2 contains a loop. Moreover, as for a local aggregate (before composition), the d and the f attributes are omitted since they can be deduced from λ. There is a deadlock in $a_1 \oplus a_2$ if and only if the empty set belongs to $\lambda(a_1 \oplus a_2)$ and $a_1 \oplus a_2$ is a final aggregate if and only if there exists an element $X \in \lambda(a_1 \oplus a_2)$ containing the

term action. Since the elements of $\lambda(a_1 \oplus a_2)$ are obtained by conjonction of the elements of $\lambda(a_1)$ and $\lambda(a_2)$, we have the following properties: (1) The local deadlocks are preserved, (2) the interlocks are detected as soon as there exists a subset $X \subseteq a_1.S$ and a subset $Y \subseteq a_2.S$ such that $\lambda_{\mathcal{N}}(X) \cap \lambda_{\mathcal{N}}(Y) = \emptyset$. Notice that, in this case, any other subset X' of $a_1.S$ such that $\lambda_{\mathcal{N}}(X') \subset \lambda_{\mathcal{N}}(X) \wedge (\lambda_{\mathcal{N}}(X') \cap \{\tau, term\} = \lambda_{\mathcal{N}}(X) \cap \{\tau, term\})$ allows to detect the interlock as well ($\lambda_{\mathcal{N}}(X') \cap \lambda_{\mathcal{N}}(Y) = \emptyset$). Thus, there is no need to keep both $\lambda_{\mathcal{N}}(X)$ and $\lambda_{\mathcal{N}}(X')$ in $\lambda(a_1)$, which justifies the use of λ_{min} to define the observed behavior of an aggregate (see Definition 25). Finally, (3) $a_1 \oplus a_2$ is a terminal aggregate if and only if a_1 and a_2 are both terminal (*term* is a shared action).

Intuitively, the observed behavior of the composition of two aggregates allows the following: (1) An observed action is possible from a state $s = (s_1, s_2)$ in $a_1 \oplus a_2$ if it is observed in $\mathcal{G}_1$ and $\mathcal{G}_2$ and is possible from both states s_1 and s_2, or it is observed only in $\mathcal{G}_1$ (resp. $\mathcal{G}_2$) and is possible from s_1 (resp. s_2). (2) A τ action is possible from a state $s = (s_1, s_2)$ in $a_1 \oplus a_2$ if there is an unobserved action which is possible from s_1 or from s_2). This is the natural behavior starting from a state of a synchronized product graph. The composition of two SOGs is similar to the classical synchronized product between two graphs, except the fact that nodes are aggregates (carrying additional information) instead of single states. Moreover, although the atomic proposition sets are disjoint in a synchronous composition, the following definition allows for non disjoint sets. The label associated with $a_1 \oplus a_2$ is then obtained by the conjunction of the labels of a_1 and a_2 completed by the own atomic propositions of each component. Finally, we notice that the τ action is not considered as a synchronization action i.e. when it is possible from a_1 (resp. a_2) leading to a'_1 (resp. a'_2), the product aggregate $a_1 \oplus a_2$ should have two successors by τ: $a'_1 \oplus a_2$ and $a_1 \oplus a'_2$.

Definition 27 (SOG synchronized product)
Let $\mathcal{G}_i = \langle \mathcal{A}_i, Obs_i \cup \{\tau\}, L, \rightarrow_i, a_{0_i}, \Omega_i \rangle, i = 1, 2$, be two SOGs over two sets of atomic propositions AP_1 and AP_2. The synchronized product *of $\mathcal{G}_1$ and $\mathcal{G}_2$, denoted by $\mathcal{G}_1 \oplus \mathcal{G}_2$ is a SOG $\langle \mathcal{A}, Obs \cup \{\tau\}, L, \rightarrow, a_0, \Omega \rangle$ over $AP_1 \cup AP_2$ where:*

1. *$\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$;*

2. *$Obs = Obs_1 \cup Obs_2$;*

3. *$L : A \rightarrow 2^{(AP_1 \cup AP_2)}$ is the labeling function s.t. $L(a_1 \oplus a_2) = (L_1(a_1) \setminus AP_2) \cup (L_2(a_2) \setminus AP_1) \cup (L_1(a_1) \cap L_2(a_2))$;*

4. *$\rightarrow$ is the transition relation, defined by:*
   $\forall (a_1, a_2) \in \mathcal{A} : (a_1, a_2) \xrightarrow{o} (a'_1, a'_2) \Leftrightarrow$

$$\begin{cases} a_1 \xrightarrow{o}_1 a_1' \wedge a_2 \xrightarrow{o}_2 a_2' & \textit{if } o \in Obs_1 \cap Obs_2 \\ a_1 \xrightarrow{o}_1 a_1' \wedge a_2 = a_2' & \textit{if } o \in ((Obs_1 \cup \{\tau\}) \setminus Obs_2) \\ a_1 = a_1' \wedge a_2 \xrightarrow{o}_2 a_2' & \textit{if } o \in ((Obs_2 \cup \{\tau\}) \setminus Obs_1) \end{cases}$$

5. $a_0 = a_{0_1} \times a_{0_2}$;

6. $\Omega = \Omega_1 \times \Omega_2$.

The set of aggregates $\mathcal{A}$ is reduced to the states that are reachable from the initial aggregate. i.e., $\mathcal{A} = \{(a_1, a_2) \in \mathcal{A}_1 \times \mathcal{A}_2 \mid \exists \sigma \in Obs^* : (a_{0_1}, a_{0_2}) \xrightarrow{\sigma} (a_1, a_2)\}$. Again, building a synchronized product of n SOGs can be done by iterative multiplication.

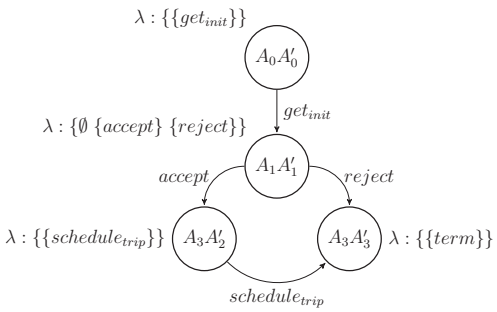


Figure 5.3: the SOG synchronized product

Figure 5.3 illustrates the SOG obtained by composing the SOGs of Figure 5.1. Again, the *reject* transition stands for both transitions $trip_{reject}$ and cc_{reject}. We note that the composed SOG contains a dead aggregate A_1A_1' (since, for instance, $\{accept\} \cap \{schedule_{trip}\} = \emptyset$) although A_1 and A_1' are deadlock-free. In fact, in the trip reservation process, transitions $schedule_{trip}$ and *accept* can be executed in any order. If the process decides to first execute transition $schedule_{trip}$, then the composite process gets in deadlock. Concerning formulae ϕ_1 and ϕ_2 presented previously, we can see that ϕ_1 is not satisfied by the composition while ϕ_2 is. Indeed, the deadlock state could prevent the occurrence of *accept* and/or *reject* transitions, while it has no effect on ϕ_2 since it occurs after the execution of the *accept* transition.

The following theorem establishes that the synchronized product of two SOGs is a SOG.

Theorem 5.2.1 *Let $\mathcal{N}_i$, for $i \in \{1, 2\}$, be two WF-nets. Let $\mathcal{G}_i$ be a SOG associated with $\mathcal{N}_i$ with respect to the set of observed actions Obs_i. Then $\mathcal{G}_1 \oplus \mathcal{G}_2$ is a SOG of the $\mathcal{N}_1 \oplus \mathcal{N}_2$ with respect to $Obs_1 \cup Obs_2$.*

Proof 5 *The proof of Theorem 5.2.1 is trivial since each aggregate (resp. edge) of $\mathcal{G}_1 \oplus \mathcal{G}_2$ can be matched with an aggregate (resp. edge) of the SOG of $\mathcal{N}_1 \oplus \mathcal{N}_2$ w.r.t. $Obs_1 \cup Obs_2$, namely $G_\oplus$, and vice-versa.*

- *Let $a = \langle S, l, \lambda \rangle$ be an aggregate of $\mathcal{N}_1 \oplus \mathcal{N}_2$. Let us prove that $a \in G_\oplus$ iff there exists an aggregate $a' = \langle S', l', \lambda' \rangle$ in $\mathcal{G}_1 \oplus \mathcal{G}_2$ such that a marking $m \in S$ iff $\langle m_{\mathcal{N}_1}, m_{\mathcal{N}_2} \rangle \in S'$, where $m_{\mathcal{N}_i}$, for $i \in \{1, 2\}$, is the projection of m on the places of $\mathcal{N}_i$.*

  Let a_0 be the initial aggregate of $G_\oplus$, we prove that there exists a sequence of transitions $\sigma \in (Obs_1 \cup Obs_2)^*$ such that $a_0 \xrightarrow{\sigma} a$ iff the sequence σ is also enabled by the initial aggregate a'_0 of $\mathcal{G}_1 \oplus \mathcal{G}_2$ leading to the required aggregate a'. We proceed by induction on the length of σ:

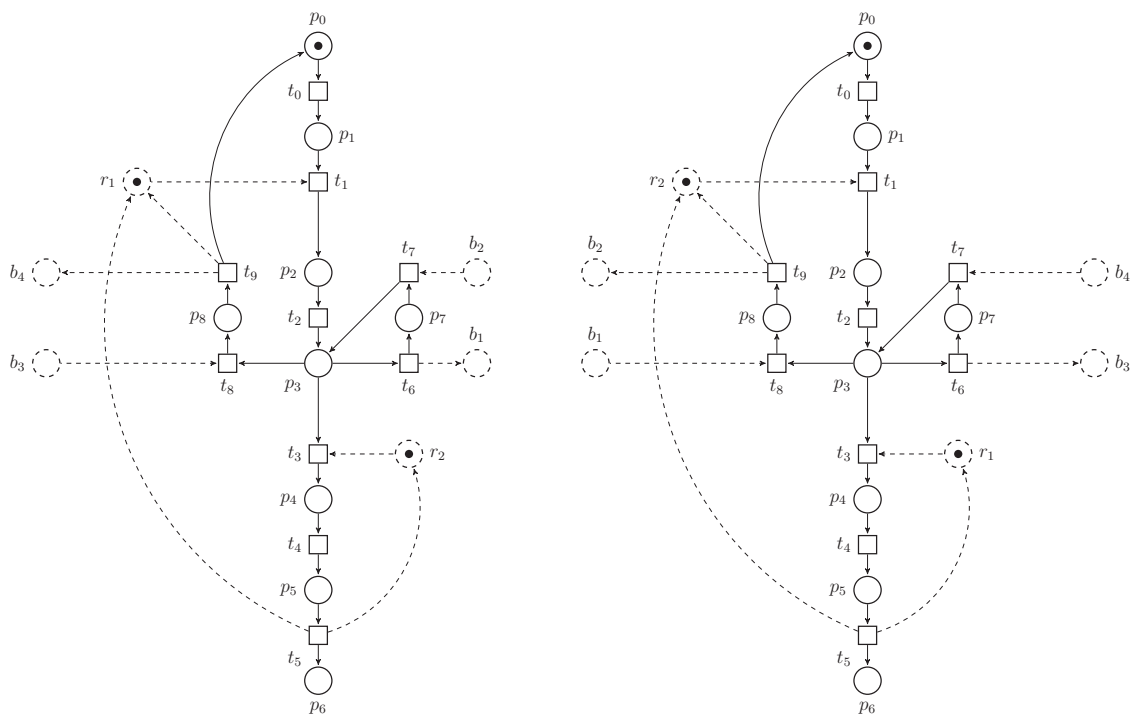
  1. *$\mid \sigma \mid = 0$. Then, $a = a_0$ and $a' = a'_0$ and the previous relation between a and a' is clearly satisfied.*
  2. *Assume the property is satisfied for any sequence of length n, and let $\sigma = t_1 \ldots t_{n+1}$ be an $n+1$ length sequence leading, from a_0, to an aggregate a_{n+1}. Let a_n and a'_n be the aggregates reachable in $G_\oplus$ and $\mathcal{G}_1 \oplus \mathcal{G}_2$, respectively, by the firing of the prefix $\sigma = t_1 \ldots t_n$. Then, since t_{n+1} is fireable from a_n, it is clearly enabled by a'_n leading to an aggregate a'_{n+1} satisfying the required condition on its constituant markings (the opposite holds as well).*

- *The second part of the proof requires that there exists an edge (a_1, t, a_2) in $G_\oplus$ iff (a'_1, t, a'_2) is an edge in $\mathcal{G}_1 \oplus \mathcal{G}_2$. Considering the previous proof of the relation between the marking constituting a_1 and a'_1 (resp. a_2 and a'_2), the existence of such an edge is trivial.*

5.2.3 Composition of RCoWF-nets' SOGs

In this section, we consider a communication between BPs that involves exchanging messages and sharing of ressources. Thus, we consider that each component of an IEBP is modeled with an RCoWF-net. The whole IEBP model is obtained by merging the respective shared constituents of the local models, which are the equally labeled (input/output interface places and shared resource places). Given a component N_i of the whole process, the set of atomic propositions AP_i involves places of the RCoWF-net and can be written as $AP_i = AP_{Li} \cup AP_{BRi}$, where AP_{Li} is the subset of atomic propositions that involve local places only, while AP_{BRi} is the subset of atomic propositions that involve interface places (i.e. input/output and resources places).

During this section, we will use the examples of Figure 5.4 in order to illustrate our approach. There are two RCoWF-nets , N_1 and N_2, sharing two resources, r_1 and r_2, and communicating asynchronously via four buffers (b_1, b_2, b_3 and b_4). Figure 5.4(a) depicts the model of N_1 whose behavior is as follows: t_0 initiates the workflow, where the resource



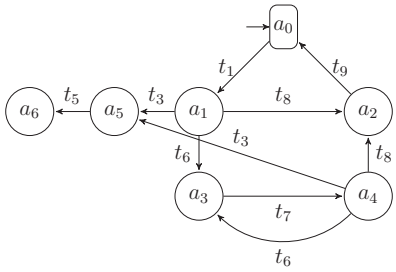
(a) a RCoWF-net N_1 (b) a RCoWF-net N_2

Figure 5.4: Two RCoWF-nets sharing resources and communicating asynchronously

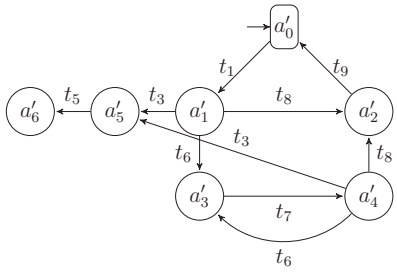
r_1 is required before asking for the second resource r_2. When r_1 is owned by N_1, three behaviors are possible: r_2 is available and the workflow can hold it, use the two resources and finally release them and finish. The second case is that r_2 is not available (it is hold by N_2), and N_1 sends a request in b_1 asking N_2 to release r_2. If N_2 accepts the request, a token in b_2 is present and r_2 is released so that N_1 can finish. The third behavior is when there is a pending message at buffer b_3 asking N_1 to release the resource r_1 in order to unlock N_2. If N_1 accepts, then it will release the resource and returns to state p_0 in order to restart. N_2 has a similar behavior except that it asks for the resource r_2 before r_1, it sends his request in b_3 (and waits for the answer from b_4) and receives requests in b_1 (and send his answer in b_2). Note that the RCoWF-nets of Figure 5.4 are both locally sound (hence relaxed, weak and easy sound). One can also be interested, for instance, in a temporal property ensuring the safe use of the resources: A property of N_1 (reps. N_2) such as "when r_2 (resp. r_1) is owned, it will eventually be released" can be expressed by the following LTL formula: $\varphi_1 = G\ (t_3 \implies F\ t_5)$. It is clear that this property is locally satisfied by both processes N_1 and N_2. One can also be interested in checking that, when the N_1 (resp. N_2) process is waiting for the resource r_1 (resp. r_2), then it will eventually hold it. This property can be expressed by the LTL formula $\varphi_2 = G\ (p_1 \implies F\ t_1)$ which involves both state and event-based atomic propositions. This property is also satisfied

locally.

Figure 5.5 shows the SOGs associated with the two RCoWF-nets of Figure 5.4. For sake of clarity, the observed behaviors of the aggregates are given separately in Figure 5.5(c).



(a) SOG of N_1



(b) SOG of N_2

Aggregates	λ
a_0, a'_0	$\{\{t_1\}\}$
a_1, a'_1	$\{\{t_6\, t_8\}\}$
a_2, a'_2	$\{\{t_9\}\}$
a_3, a'_3	$\{\{t_7\}\}$
a_4, a'_4	$\{\{t_8\, t_6\}\}$
a_5, a'_5	$\{\{t_5\}\}$
a_6, a'_6	$\{\{term\}\}$

(c) λ of each aggregate

Figure 5.5: SOGs of the running example

In order to take into account the asynchronous composition and the sharing of resources between $\mathcal{N}_1$ and $\mathcal{N}_2$, we define a medium net N_{12} as an RCoWF-net representing the interaction between $\mathcal{N}_1$ and $\mathcal{N}_2$.

Definition 28 (The medium net) *Let $N_i = \langle P_i, T_i, F_{p_i} \cup F_{r_i} \cup F_{c_i}, W_i, I_i, O_i, R_i \rangle$, for $i = 1, 2$, be two interface compatible RCoWF-nets and let m_{0_i} and m_{f_i} be the corresponding initial and final markings respectively. The medium net related to N_1 and N_2, denoted by $N_{12} = \langle P_{12}, T_{12}, F_{12}, W_{12} \rangle$, is the WF-net associated with the initial and final markings $m_{0_{12}}$ and $m_{f_{12}}$ as follows :*

- $P_{12} = (I_1 \cap O_2) \cup (O_1 \cap I_2) \cup (R_1 \cap R_2)$
- $T_{12} = \{t \in T_i;\ {}^\bullet t^\bullet \cap ((I_i \cap O_j) \cup (O_i \cap I_j) \cup (R_i \cap R_j)) \neq \emptyset\}$ *for* $i, j \in \{1, 2\}$ *and* $i \neq j$
- $F_{12} = (F_{p_1} \cup F_{r_1} \cup F_{c_1})|_{(P_{12} \times T_{12}) \cup (T_{12} \times P_{12})} \cup (F_{p_2} \cup F_{r_2} \cup F_{c_2})|_{(P_{12} \times T_{12}) \cup (T_{12} \times P_{12})}$
- $W_{12} : F_{12} \to \mathbb{N}^+$ *s.t.* $W_{12}(f) = W_i(f) \Leftrightarrow f \in (F_{p_i} \cup F_{r_i} \cup F_{c_i})$
- $m_{0_{12}}$ *is the initial marking where only resources places of* $R_1 \cap R_2$ *are marked as in the initial markings* m_{01} *and* m_{02}

- $m_{f_{12}} = \{m_{0_{12}}\}$

The transitions of N_{12} are the input/output transitions and those that are linked to the resource places of $\mathcal{N}_1$ and $\mathcal{N}_2$, while its places are the input/output and resource places. The connection between places and transitions of the medium net is inherited from $\mathcal{N}_1$ and $\mathcal{N}_2$. Its initial and final states can be obtained by projection of those of the involved components, i.e. $\mathcal{N}_1$ and $\mathcal{N}_2$, on its places. It is clear that, when the set of input/output places is not empty, the set of reachable markings of the medium net is infinite. However, if we assume that the composite net $\mathcal{N}_1 \oplus \mathcal{N}_2$ is bounded, then the corresponding reachability graph is finite. If the bound of an interface place is n then this place can be in $n+1$ different states at most. Under such an assumption and knowing the bound of each place of the medium net, one can build a reachability graph that covers all the possible behaviors related to the interface places in $\mathcal{N}_1 \oplus \mathcal{N}_2$. The obtained graph is called *Interface graph* and defined in the following as a SOG.

Definition 29 *Consider two RCoWF-nets $\mathcal{N}_1$ and $\mathcal{N}_2$ and their medium net N_{12}. For each place p_i (for $i = 1 \ldots m$) of N_{12}, let n_i be the bound of p_i in $\mathcal{N}_1 \oplus \mathcal{N}_2$. Then, the Interface graph is an LKS $\mathcal{G}_{12} = \langle \Gamma_{12}, T_{12}, L_{12}, \rightarrow_{12}, m_{0_{12}}, m_{f_{12}} \rangle$ over the set of atomic propositions $AP_{12} = AP_{BR1} \cup AP_{BR2}$ s.t.:*

1. $\Gamma_{12} \subseteq \mathbb{N}^{|P_{12}|}$
2. $L_{12} : \Gamma_{12} \rightarrow 2^{AP_{12}}$ *is the labeling function s.t. $L(s)$ contains the set of atomic propositions that are satisfied by s ;*
3. $\rightarrow_{12} \subseteq \Gamma_{12} \times T_{12} \times \Gamma_{12}$ *is a transition relation such that:* $(m, t, m') \in \rightarrow \Leftrightarrow ((m \xrightarrow{t} m') \wedge (m' \leq \langle n_1, \ldots, n_m \rangle))$
4. m_0 *is the initial marking where only resources places are marked.*
5. $m_f = m_0$ *is the final marking*

The above definition constructs a reachability graph where each marking represents a possible configuration of the interface places (input/output and ressource places) of $\mathcal{N}_1$ and $\mathcal{N}_2$. The labeling function defines, for each state s, the set of atomic propositions (of AP_{12}) that are satisfied. The transition relation allows the evolution of the interface places' states in the following manner: A successor of a given marking m is a marking reachable from m and where the number of tokens in each place does not exceed its defined bound. Moreover, the initial marking (which is the final marking as well) is the state where only the resource places are marked.

By observing all the transitions of the medium net, this graph can be seen as a SOG where: the aggregates are singletons (each reachable marking is an aggregate) and the

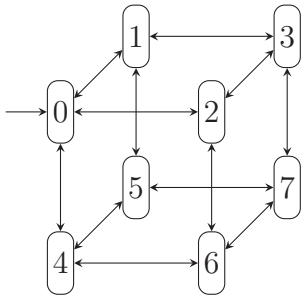


Figure 5.6: Interface graph of the medium net

observed behavior of each aggregate is also a singleton (the set of transitions appearing on the outgoing arcs of the corresponding marking). The final marking/aggregate will have $\{\{term\}\}$ as observed behavior. Finally, the l attribute of any aggregate of this SOG is equal to false.

Regarding the example of Figure 5.4, the SOG associated with the medium net contains at most 32 aggregates since there are 6 one bounded interface places. Figure 5.6 illustrates an example of medium net's SOG containing three interface places (namely p_1, p_2 and p_3). The binary representation of each state number gives the state of these interface places. For instance, state number 5 stands for 101, i.e., only the interface place p_2 is not marked. Unlike the SOGs associated with $\mathcal{N}_1$ and $\mathcal{N}_2$, the SOG of the medium net is not supposed to be built a priori. Thus, the bounds of the places of N_{12} have not to be known in advance, as long as the composed net $\mathcal{N}_1 \oplus \mathcal{N}_2$ is bound.

In the following, the SOG of the medium net will be computed on-the-fly during the composition of $\mathcal{G}_1$ and $\mathcal{G}_2$. The composition of $\mathcal{G}_1$ and $\mathcal{G}_2$, denoted by $\mathcal{G}_1 \oplus \mathcal{G}_2$ is then defined as the synchronized product between three SOGs corresponding to $\mathcal{N}_1$, $\mathcal{N}_{12}$ and $\mathcal{N}_2$ respectively. As long as a medium net $\mathcal{N}_{12}$ between two RCoWF-nets $\mathcal{N}_1$ and $\mathcal{N}_2$ is concerned, $\mathcal{G}_1 \oplus \mathcal{G}_2$ states for $\mathcal{G}_1 \oplus \mathcal{G}_{12} \oplus \mathcal{G}_2$. Hence, one can iteratively use Definition 27 and Definition 26 in order to obtain $\mathcal{G}_1 \oplus \mathcal{G}_2$ and the corresponding aggregates respectively. However, since we know in advance that $\mathcal{G}_1$ and $\mathcal{G}_2$ use disjoint sets of transitions (each shares some transitions with the medium net's SOG only), the definitions of the aggregate product and of the synchronization of three SOGs is simplified in the following.

Before we define the composition of SOGs, it is important to first show how, using the local attributes of three aggregates a_1, a_2 and a_{12} of $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_{12}$ respectively, one can compute the attributes of the aggregate resulting from their composition.

Definition 30 (The product aggregate) *Let $\mathcal{G}_i$, for $i = 1, 2$, be two SOGs associated with two RCoWF-nets and let $\mathcal{G}_{12}$ be the SOG associated with their medium net. Let a_1, a_2 and a_{12} be three aggregates of these SOGs respectively. The* product aggregate

$a = (a_1, a_{12}, a_2)$ *is defined by:*

1. $a.S = a_1.S \times a_{12}.S \times a_2.S$;

2. $a.l = a_1.l \vee a_2.l$;

3. $\lambda(a) = \{((x \cap y) \cup (x \cap ((Obs_1 \cup \{\tau\}) \setminus Obs_{12}))) \cup ((y \cap z) \cup (z \cap ((Obs_2 \cup \{\tau\}) \setminus Obs_{12}))) \mid x \in \lambda(a_1),\ y \in \lambda(a_{12})$ *and* $z \in \lambda(a_2)\}$;

Note that the l attribute depends only on the corresponding value in a_1 and a_2 since, by definition, a_{12} is a singleton (thus $a_{12}.l = false$). Finally, $Obs_i \cap Obs_{12}$, for $i = 1, 2$, is not empty (because $\mathcal{N}_1$ and $\mathcal{N}_2$ are interface compatible) but Obs_i is not necessarily a subset of Obs_{12}, and that $Obs_1 \cap Obs_2 = \{term\}$. When we compose a_1 and a_2, if a_1 (resp. a_2) can progress in $\mathcal{G}_1$ (resp. $\mathcal{G}_2$) by using local observed transitions (i.e., transitions in $Obs_1 \setminus Obs_{12}$ (resp. $Obs_2 \setminus Obs_{12}$)), the product aggregate a should be able to do the same. If this is not the case, then a has to have the same behavior as a_1 (resp. a_2) and a_{12} conjointly.

Definition 31 (Composition of oWF-nets' SOGs) *Let* $\mathcal{G}_i = \langle \mathcal{A}_i, Obs_i \cup \{\tau\}, L_i, \rightarrow_i, a_{0i}, \Omega_i \rangle, i = 1, 2$ *be two SOGs corresponding to two oWF-nets* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *over* $AP_1 = AP_{L1} \cup AP_{Br1}$ *and* $AP_2 = AP_{L2} \cup AP_{Br2}$ *respectively. Let* $\mathcal{G}_{12} = \langle \mathcal{A}_{12}, Obs_{12}, \rightarrow_{12}, a_{012}, \Omega_{12} \rangle$ *be the interface graph of the medium net* N_{12} *over* AP_{12}. *The composition of* $\mathcal{G}_1$ *and* $\mathcal{G}_2$, *namely* $\mathcal{G}_1 \oplus \mathcal{G}_2 = \langle \mathcal{A}, Obs \cup \{\tau\}, L, \rightarrow, a_0, \Omega \rangle$ *is defined as follows:*

1. $\mathcal{A} \subseteq \mathcal{A}_1 \times \mathcal{A}_{12} \times \mathcal{A}_2$;

2. $Obs = Obs_1 \cup Obs_2$;

3. $L : A \rightarrow 2^{(AP_1 \cup AP_{12} \cup AP_2)}$ *is the labeling function s.t.* $L(a_1 \oplus a_{12} \oplus a_2) = L_1(a_1)|_{AP_{L1}} \cup L_2(a_2)|_{AP_{L2}} \cup L_{12}(a_{12})$

4. $\rightarrow$ *is the transition relation, defined by:*
$\forall (a_1, a_{12}, a_2) \in \mathcal{A},\ \forall (a'_1, a'_{12}, a'_2) \in \mathcal{A},\ (a_1, a_{12}, a_2) \xrightarrow{o} (a'_1, a'_{12}, a'_2) \Leftrightarrow$
$$\begin{cases} a_1 \xrightarrow{o}_1 a'_1 \wedge a_{12} \xrightarrow{o}_{12} a'_{12} \wedge a'_2 = a_2 & \text{if } o \in (Obs_1 \cap Obs_{12}) \\ a'_1 = a_1 \wedge a_{12} \xrightarrow{o}_{12} a'_{12} \wedge a_2 \xrightarrow{o}_2 a'_2 & \text{if } o \in (Obs_2 \cap Obs_{12}) \\ a_1 \xrightarrow{o}_1 a'_1 \wedge a'_{12} = a_{12} \wedge a'_2 = a_2 & \text{if } o \in (Obs_1 \cup \{\tau\} \setminus Obs_{12}) \\ a'_1 = a_1 \wedge a'_{12} = a_{12} \wedge a_2 \xrightarrow{o}_2 a'_2 & \text{if } o \in (Obs_2 \cup \{\tau\} \setminus Obs_{12}) \end{cases}$$

5. $a_0 = (a_{01}, a_{012}, a_{02})$;

6. $\Omega = \Omega_1 \times \Omega_{12} \times \Omega_2$.

The composition of the three SOGs $\mathcal{G}_1$, $\mathcal{G}_{12}$ and $\mathcal{G}_2$ can be obtained by applying Definition 31. Here, the composition of the corresponding RCoWF-nets has been reduced to a synchronous composition involving the medium net. The evolution in $\mathcal{G}_1 \oplus \mathcal{G}_2$ can stand for a local evolution to $\mathcal{G}_1$ (resp. $\mathcal{G}_2$) by using point 3 (resp. 4) of the transition relation in Definition 27, or a simultaneous evolution in $\mathcal{G}_1$ (resp. $\mathcal{G}_2$) and $\mathcal{G}_{12}$ by using point 1 (resp. 2).

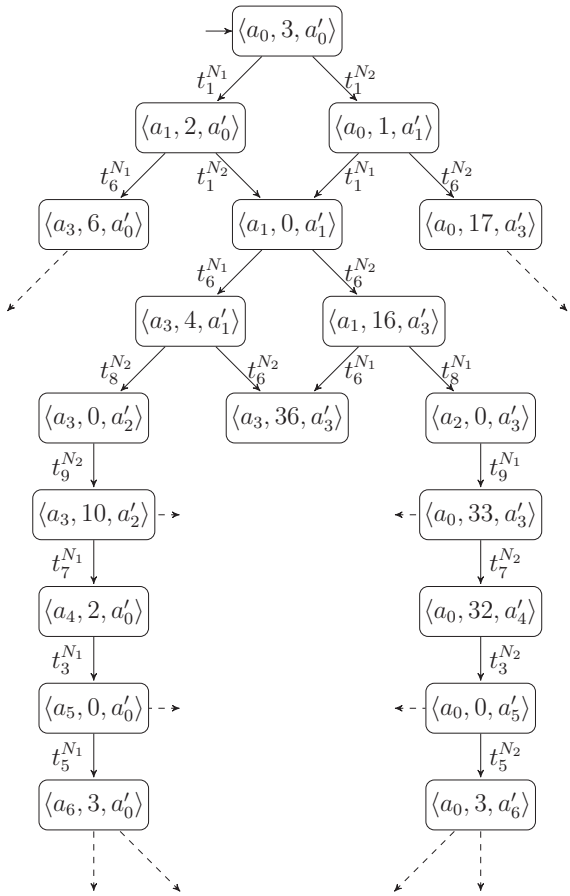


Figure 5.7: The synchronous product of the SOGs of the RCoWF-nets example

We note that the synchronized product $\mathcal{G}_1 \oplus \mathcal{G}_2$ of the two SOGs $\mathcal{G}_1$ and $\mathcal{G}_2$ (illustrated in Figure 5.5) contains 34 aggregates and 52 arcs. For this example, we do not observe only 6 transitions (from 18 transitions) but the size of $\mathcal{G}_1 \oplus \mathcal{G}_2$ represent the half size of the reachability graph of the whole system $\mathcal{N}_1 \oplus \mathcal{N}_2$ composed by models in Figure 5.4, which contains 60 states and 92 arcs. Because of lack of space, only a part of $\mathcal{G}_1 \oplus \mathcal{G}_2$ is given in Figure 5.7. Each aggregate of the composition is a triplet $\langle a, n, a' \rangle$ where a and a' are two aggregates corresponding to N_1 and N_2 respectively, and n is the number corresponding to the binary representation of the marking of the interface places (in this order: $(b_4, b_3, b_2, b_1, r_2, r_1)$. The Figure illustrates some interesting behavior: One can see, for instance through the $t_1^{N_1}.t_6^{N_1}.t_1^{N_2}.t_6^{N_2}$ sequence, that the synchronized product ends in

a deadlock aggregate ($\lambda(\langle a_3, 36, a'_3\rangle) = \{\emptyset\}$). Note finally that the λ attributes of the product aggregates are omitted in this Figure for sake of clarity.

Using the medium net, the computing of a product aggregate's attributes (Definition 30) and the composition of two RCoWF-nets' SOGs (Definition 31), we obtain the following result. It states that the composition of two SOGs corresponding to two RCWF-nets is isomorphic to the SOG of their original composition when observing each transition connected to an interface place (buffer or resource). The proof of this theorem is identical to the proof of Theorem 5.2.1 since the asynchronous composition between two RCoWF-nets is here reduced to the synchronous composition of three nets.

Theorem 5.2.2 *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two RCoWF-nets and let $\mathcal{G}_1$ and $\mathcal{G}_2$ be the corresponding SOGs respectively. Then, $\mathcal{G}_1 \oplus \mathcal{G}_2$ is a SOG of $\mathcal{N}_1 \oplus \mathcal{N}_2$ with respect to $Obs_1 \cup Obs_2$.*

5.3 Modular verification

We follow a bottom-up approach where each component of an IEBP is designed and analyzed independently from the others. Using the composition of SOGs defined in the previous section, we deal in this section with the verification of specific and/or generic properties of IEBPs. A SOG related to an RCoWF-net is built over a set of observed elements. In case we are interested in generic properties (e.g. soundness), this set contains the interface transitions only. However, when we are interested in some (hybrid) $LTL \setminus X$ formula, the SOG is built over the interface transitions and the elements (atomic propositions and transitions) occurring in the formula.

5.3.1 LTL-based Properties

Given two RCoWF-nets $\mathcal{N}_1$ and $\mathcal{N}_2$ and a (hybrid) local $LTL \setminus X$ formula φ_1 (resp. φ_2) to be checked on $\mathcal{N}_1$ (resp. $\mathcal{N}_2$), our goal is to check whether φ_1 (resp. φ_2) is satisfied by $\mathcal{N}_1 \oplus \mathcal{N}_2$ by analyzing the SOGs of $\mathcal{N}_1$ and $\mathcal{N}_2$ only. Let $\mathcal{G}'_1$ (resp. $\mathcal{G}'_2$) be a SOG of $\mathcal{N}_1$ (resp. $\mathcal{N}_2$) where the set of atomic propositions are those occurring in φ_1 (resp. φ_2) and where the set of observed transitions contains any transition occurring in φ_1 (resp. φ_2) or belonging to the interface. Using the approach described in the previous Chapter, one can check whether φ_1 (resp. φ_2) is satisfied by $\mathcal{N}_1$ (resp. $\mathcal{N}_2$). However this does not guarantee that φ_1 (resp. φ_2) is satisfied by $\mathcal{N}_1 \oplus \mathcal{N}_2$. Based on Theorem 4.2.1 and Theorem 5.2.2, the verification of φ_1 (resp. φ_2) on the whole net can be achieved on $\mathcal{G}'_1 \oplus \mathcal{G}_2$ (resp. $\mathcal{G}_1 \oplus \mathcal{G}'_2$), where $\mathcal{G}_2$ (resp. $\mathcal{G}_1$) is a SOG of $\mathcal{N}_2$ (resp. $\mathcal{N}_1$) over the set of the interface transitions only.

Notice that the SOG $\mathcal{G}'_i$ (for $i = 1 \ldots 2$) coincides with $\mathcal{G}_i$ in two cases: (1) when the formula φ_i involves observed transitions only, and (2) when $\varphi_i = true$ (i.e. when there is no formula φ_i to be checked on $\mathcal{N}_i$).

Theorem 5.3.1 *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two RCoWF-nets, let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two corresponding SOGs over the interface transitions Int. Let φ_1 (resp. φ_2) be an LTL\X involving a set of atomic propositions AP_1 (resp. AP_2) and a set of transitions Obs_1 (resp. Obs_2). Let (for $i \in \{1, 2\}$) $\mathcal{G}'_i$ a SOG of N_i over $Obs_i \cup AP_i \cup Int$. Then,*

- $\mathcal{G}'_1 \oplus \mathcal{G}_2 \models \varphi_1$ *and* $\mathcal{G}_1 \oplus \mathcal{G}'_2 \models \varphi_2 \Leftrightarrow \mathcal{G}'_1 \oplus \mathcal{G}'_2 \models \varphi_1 \wedge \varphi_2$

Proof 6 *The proof is obvious because of the following:*

- $\Rightarrow$ *Given an RCoWF-net $\mathcal{N}$ and an $LTL \setminus X$ formula φ involving the set of atomic propositions AP and the set of observed transitions Obs the SOG of $\mathcal{N}$ over AP' and Obs', for some $AP \subseteq AP'$ and some $Obs \subseteq Obs'$ can be used to check φ instead of the SOG built over $AP \cup Obs$, while leading to an equivalent result. Indeed such a SOG will probably be bigger (since some aggregate might be splitting due to the observation of additional elements) but it contains all the behavior that are necessary to check φ. Thus, the fact that $\mathcal{G}'_1 \oplus \mathcal{G}_2$ satisfies φ_1 and the fact that $\mathcal{G}_1 \oplus \mathcal{G}'_2$ satisfies φ_2 imply that $\mathcal{G}'_1 \oplus \mathcal{G}'_2$ satisfies both φ_1 and φ_2.*

- $\Leftarrow$ *Given an RCoWF-net $\mathcal{N}$ and an $LTL \setminus X$ formula φ involving the set of atomic propositions AP and the set of observed transitions Obs, the SOG of $\mathcal{N}$ over AP and Obs can be reused to check any formula involving a subset of AP and a subset of Obs (see Theorem 4.2.1), while giving the same result. Thus, the fact that $\mathcal{G}'_1 \oplus \mathcal{G}'_2$ is built over $Obs_1 \cup AP_1 \cup Obs_2 \cup AP_2 \cup Int$, it can be used for checking φ_1 on $\mathcal{G}'_1 \oplus \mathcal{G}_2$ and for the checking of φ_2 on $\mathcal{G}_1 \oplus \mathcal{G}'_2$. Indeed $Obs_1 \cup AP_1 \cup Int$ (which is used to build $\mathcal{G}'_1$) and $Obs_2 \cup AP_2 \cup Int$ (which is used to build $\mathcal{G}'_2$) are respectively subsets of $Obs_1 \cup AP_1 \cup Obs_2 \cup AP_2 \cup Int$.*

The following corollary is then obtained from the previous theorem and involves the composite model $\mathcal{N}_1 \oplus \mathcal{N}_2$.

Corollary 1 *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two RCoWF-nets, let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two corresponding SOGs over the interface transitions Int. Let φ_1 (resp. φ_2) be an LTL\X involving a set of atomic propositions AP_1 (resp. AP_2) and a set of transitions Obs_1 (resp. Obs_2). Let (for $i \in \{1, 2\}$) $\mathcal{G}'_i$ a SOG of N_i over $Obs_i \cup AP_i \cup Int$. Then,*

- $\mathcal{G}'_1 \oplus \mathcal{G}_2 \models \varphi_1$ *and* $\mathcal{G}_1 \oplus \mathcal{G}'_2 \models \varphi_2 \Leftrightarrow \mathcal{N}_1 \oplus \mathcal{N}_2 \models \varphi_1 \wedge \varphi_2$

Coming back to the example of Figure 5.4, one can see that both formulae $\varphi_1 = G\ (t_3 \implies F\ t_5)$ and $\varphi_2 = G\ (p_1 \implies F\ t_1)$ are preserved by composition. However, the formula $\varphi_3 = G\ (t_1 \implies F\ t_5)$ is violated after composition (but locally satisfied by both processes N_1 and N_2). In fact, we have an interlock resulting from the firing of the sequence $t_1.t_6$ in N_1 followed by the firing of the sequence $t_1.t_6$ in N_2 (N_1 holds r_1 and waits for r_2 and N_2 holds r_2 and waits for r_1).

Now that we showed how to check LTL formulae using the composition of SOGs, we consecrate the rest of this section to the soundness properties.

5.3.2 Checking Soundness Properties

Soundness is not preserved by composition. For instance, the RCoWF-nets of Figure 5.4 are both locally sound while their composition is not. In fact, the firing of the sequence $t_1.t_6$ in N_1 followed by the firing of the sequence $t_1.t_6$ in N_2 lead to deadlock state. In this section, we will show why the aggregates of a SOG (see Definition 18 of the previous Chapter) must be enriched with new attributes in order to allow the verification of soundness properties using the synchronized product of two (or more) SOGs. From a local point of view, the synchronized product constraints the behavior of a component by taking into account the behavior of its environment. The goal of the remaining part of this section is to measure the consequence of such constraints on the properties that have been proved satisfied locally to each component. Note that, for checking soundness properties, the SOGs of each component of the IEBP is built over the set of the interface transitions only.

5.3.3 Soundness

Based on the current attributes of a SOG's aggregate (see Definition 18), which are the observed behavior λ and the loop attribute l (the set of states belonging to the aggregate are not stored), we note the following: (1) The l attribute is no more useful (only terminal loops, which are covered by the observed behavior, are important regarding to soundness), and (2) the observed behavior is sufficient to check the *option to complete* requirement but it is not sufficient to check the *no dead transitions* reuirement. In fact, the locally reachable aggregates of an RCoWF-net are not necessarily reachable after composition. In this case, we are not able to state whether any unobserved transition, which is enabled by a marking of such an aggregate, is enabled elsewhere (within an other aggregate). Thus, the set $E_t(a)$ (containing the set of enabled transitions in any state inside an aggregate a and defined in Definition 22) has to be kept as an attribute of each aggregate for the composition. We do not consider that publishing this set represents a violation

of the privacy of the underlying process since it could consist of obsolete names which are not necessarily the real names of the transitions. As for the observed behavior (see Definition 26), the value of this attribute for a product aggregate $a = (a_1, a_{12}, a_2)$ can be deduced as follows:

Definition 32 *Let $\mathcal{G}_i$, for $i = 1, 2$, be two SOGs and let $\mathcal{G}_{12}$ be the SOG associated with the medium net. Let a_1, a_2 and a_{12} be three aggregates of these SOGs respectively. The* product aggregate *$a = (a_1, a_{12}, a_2)$ is defined by $a = \langle \lambda, Enable \rangle$: where $E_t(a) = \bigcup_{i=1...2}(E_t(a_i) \setminus (Obs_i \cap Obs_{12})) \cup (E_t(a_i) \cap E_t(a_{12}))$*

Using the E_t attribute and given a local transition t in $\mathcal{N}_1$ (for instance) one can check whether it remains enabled (by some product aggregate) after composition or not. Indeed, t is proved to be not dead in the SOG composition as soon as it belongs to the E_t attribute of some product aggregate. If this condition is satisfied by all the transitions, then the *no dead transition* requirement is preserved by the composition.

Using the observed behavior $\lambda(a)$ and the set of enabled transitions $E_t(a)$ as sole attributes of an aggregate a of a SOG, we are able to completely characterize the soundness of the composition of two SOGs. The *option to complete* requirement is guaranteed as long as there is no interlock and each product aggregate leads to a final aggregate. Note that, as for the non modular approach (the previous Chapter), and under the fairness assumption, one can use our modular LTL model checker in order to check this requirement, expressed by an LTL formula. It is satisfied as soon as the LTL formula $\phi = G\ F\ term$ is verified by the $\mathcal{G}_1 \oplus \mathcal{G}_2$ ($term$ being shared by components). Note finally that the *proper completion* requirement is directly preserved by composition.

Definition 33 *Let $\mathcal{N}_1 \oplus \mathcal{N}_2$ be a composite RCoWF-net. Let $\mathcal{G}_i = \langle \mathcal{A}_i, a_{0i}, \rightarrow_i, \mathcal{F}_i \rangle$ ($i = 1, 2$) be two SOGs corresponding to $\mathcal{N}_1$ and $\mathcal{N}_2$ and let $\mathcal{G}_1 \oplus \mathcal{G}_2 = \langle \mathcal{A}, a_0, \rightarrow, \mathcal{F} \rangle$ be their composition*

- *if $\mathcal{G}_1$ and $\mathcal{G}_2$ are both sound then $\mathcal{G}_1 \oplus \mathcal{G}_2$ is sound iff:*
  - $\forall a \in \mathcal{A}$, $\emptyset \notin a.\lambda \wedge \exists a_f \in \mathcal{F} \mid a_f \in R(a)$.
  - $\bigcup_{a \in \mathcal{A}} E_t(S) = \bigcup_{i=1...2} \bigcup_{a_i \in \mathcal{A}_i} E_t(a_i)$.

Then, the equivalence between checking the soundness property of a composite RCoWF-net $\mathcal{N}_1 \oplus \mathcal{N}_2$ and checking soundness of a composite SOG $\mathcal{G}_1 \oplus \mathcal{G}_2$ is established in the following theorem.

Theorem 5.3.2 *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two RCoWF- nets and let $\mathcal{N}_1 \oplus \mathcal{N}_2$ be the corresponding composite RCoWF-net. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two SOGs corresponding to $\mathcal{N}_1$ and $\mathcal{N}_2$ respectively.*

- *if $\mathcal{N}_1$ and $\mathcal{N}_2$ are both sound, then $\mathcal{N}_1 \oplus \mathcal{N}_2$ is sound $\Leftrightarrow \mathcal{G}_1 \oplus \mathcal{G}_2$ is sound,*

Proof 7 $\Rightarrow$ *Assume that $\mathcal{N}_1 \oplus \mathcal{N}_2$ is sound.*

1. *Let a be an aggregate of $\mathcal{G}_1 \oplus \mathcal{G}_2$ and let us suppose that $\emptyset \in a.\lambda$. This means that there exists a dead marking $m \in a.S$. Since $\mathcal{G}_1 \oplus \mathcal{G}_2$ is a SOG of $\mathcal{N}_1 \oplus \mathcal{N}_2$ (Theorem 5.2.2) the marking m is reachable in $\mathcal{N}_1 \oplus \mathcal{N}_2$ as well, which would mean that $\mathcal{N}_1 \oplus \mathcal{N}_2$ is not deadlock free which is contrary to the hypothesis. Thus $\mathcal{G}_1 \oplus \mathcal{G}_2$ is deadlock free.*

2. *Let a be an aggregate of $\mathcal{G}_1 \oplus \mathcal{G}_2$ and let m be a marking in $a.S$, since $\mathcal{N}_1 \oplus \mathcal{N}_2$ is sound then there exist a sequence σ leading from m to a final marking $m_f = m_1 \oplus m_{12} \oplus m_2$ where m_1, m_{12} and m_2 are the projections of m_f on the places of $\mathcal{N}_1$, N_{12} and $\mathcal{N}_2$ respectively. Using Theorem 5.2.2, $\sigma|_{Obs_1 \cup Obs_2}$ is enabled by a and its firing leads to an aggregate a_f s.t. $m_f \in a_f.S$. Now using the proper completion property of $\mathcal{N}_1 \oplus \mathcal{N}_2$ m_{12} is the zero vector which the final marking of N_{12}. Thus there exists a path from the aggregate a to a final aggregate a_f of $\mathcal{G}_1 \oplus \mathcal{G}_2$.*

3. *Obvious using Theorem 5.2.2 and Lemma 4.2.2 (see previous Chapter). Indeed, the first guarantees that $\mathcal{G}_1 \oplus \mathcal{G}_2$ is a SOG of $\mathcal{N}_1 \oplus \mathcal{N}_2$ while the second ensures that each path in reachability graph of $\mathcal{N}_1 \oplus \mathcal{N}_2$ corresponds to a path in $\mathcal{G}_1 \oplus \mathcal{G}_2$. Thus, for any transition t in $\mathcal{N}_1 \oplus \mathcal{N}_2$, for any reachable state s_t enabling t ($\mathcal{N}_1 \oplus \mathcal{N}_2$ is sound) one can find a corresponding path leading to an aggregate a_t containing s_t where t (observed or not) is necessarily enabled.*

$\Leftarrow$: *Assume that $\mathcal{G}_1 \oplus \mathcal{G}_2$ satisfies the three requirements of Theorem 15.*

1. *option to complete*
   Let m be a marking of $\mathcal{N}_1 \oplus \mathcal{N}_2$. Assume that there is no path leading from m to a final marking. This would mean that either m is a dead state (i.e. $\lambda(m) = \emptyset$), or $\lambda(m) \neq \emptyset$ and m belongs to a terminal strongly connected component C. In the first case, Theorem 5.2.2 ensures that there exists an aggregate a in $\mathcal{G}_1 \oplus \mathcal{G}_2$ s.t. $m \in a.S$. This would mean that $\emptyset \in \lambda(a)$ which is contrary to the hypothesis. In the second case, Theorem 5.2.2 ensures that there exists a terminal strongly connected component $\overline{C}$ whose aggregate cover the set of markings involved in C. However, by hypothesis, for each aggregate, especially those of $\overline{C}$ there exists a path in $\mathcal{G}_1 \oplus \mathcal{G}_2$ leading to a final aggregate. Thus, the option to complete requirement is satisfied by $\mathcal{N}_1 \oplus \mathcal{N}_2$.

2. *proper completion*
   Assume this requirement is not satisfied by $\mathcal{N}_1 \oplus \mathcal{N}_2$. Then there exists a marking

$m = m_1 \oplus m_{12} \oplus m_2$, where m_1, m_{12} and m_2 are the projections of m on places of $\mathcal{N}_1$, N_{12} and $\mathcal{N}_2$ respectively, and there exists a final marking m_f s.t. $m > m_f$. Let m_{f_1}, $m_{f_{12}}$, and m_{f_2} be the projections of m_f on the places of $\mathcal{N}_1$, N_{12} and $\mathcal{N}_2$ respectively. Since $\mathcal{N}_i$ (for $i \in \{1, 2\}$) is sound, and since that m_i is reachable in $\mathcal{N}_i$ (obvious because $\mathcal{N}_i$ is a subnet obtained by removing its interface places, the transitions and the places of $\mathcal{N}_j$, for $j \neq i \wedge j \in \{1, 2\}$), then $m_i = m_{f_i}$. Assume then that $m_{12} > m_{f_{12}}$. This would mean that there is still some tokens in the interface places while $\mathcal{N}_1$ and $\mathcal{N}_2$ are in their final marking already. Let a be the aggregate of $\mathcal{G}_1 \oplus \mathcal{G}_2$ s.t. $m \in a.S$, and let a_f be the final aggregate reachable from a. Note that the final markings of $\mathcal{N}_1$ and $\mathcal{N}_2$ are terminal and note also that each final marking belonging to a_f has no token in each interface place (interface place). Thus, the only way to reach a final marking from m is to consume the tokens present in the interface places. By doing so, the marking of some place in $\mathcal{N}_1$ or $\mathcal{N}_2$ will be incremented which is not possible since these markings are sound (satisfy the proper completion requirement).

3. *no dead transitions*
   Let t be a transition in $\mathcal{N}_1 \oplus \mathcal{N}_2$.

   - *if $t \in Obs_1 \cup Obs_2$*
     Since there exists an aggregate a in $\mathcal{G}_1 \oplus \mathcal{G}_2$ enabling t and by using Theorem 5.2.2, one can deduce that there exists a marking $m \in a.S$ which is reachable in $\mathcal{N}_1 \oplus \mathcal{N}_2$ and thus enabling t.

   - *if $t \notin Obs_1 \cup Obs_2$*
     Assume that t is a local transition of, for instance, $\mathcal{N}_1$. Since local transitions, especially t, remain enabled after composition, there exists an aggregate $a = (a_1, a_{12}, a_2)$ and there exists a marking $m = m_1 \oplus m_{12} \oplus m_2$ in $a.S$, where m_1, m_{12} and m_2 are the projections of m on places of $\mathcal{N}_1$, N_{12} and $\mathcal{N}_2$ respectively, s.t. $m_1 \xrightarrow{t}$. Theorem 5.2.2 ensures that m is also reachable in $\mathcal{N}_1 \oplus \mathcal{N}_2$. Since t is local to $\mathcal{N}_1$, it is also enabled by m.

5.3.4 Relaxed, Weak and Easy Soundness

Let us start by weak and easy soundness since the observed behavior of an aggregate of the SOG's synchronized product is sufficient to reduce the verification of these properties to the analysis of this product. In fact, the weak soundness is satisfied when there is no interlock and when each aggregate allows to reach a final aggregate. The easy soundness is satisfied as soon as a final aggregate is reached from the initial aggregate. The verification

of the relaxed soundness is however more complex. Indeed, let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two local processes whose SOGs are $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively, as soon as a local aggregate a_i (for $i = 1 \ldots 2$) is proved to be reachable in $\mathcal{G}_i$ but not reachable in $\mathcal{G}_1 \oplus \mathcal{G}_2$, one is no longer able to decide whether a local transition enabled by a state in a_i still belongs, in the $\mathcal{G}_1 \oplus \mathcal{G}_2$, to a path leading to the final state. Implicitly, the value of the T_f predicate of the predecessor aggregates of a_i are no longer trustable and must be updated during the composition process. However, the computing of the new value of T_f is done by the corresponding process which has the knowledge of the underlying model. Thus, in the following, we first define how to update this attribute, then how to use it in order to deduce the corresponding value of a product aggregate, and finally we characterize the relaxed soundness property (and the weak and easy soundness) on the SOG's composition.

Definition 34 *Let $G_1 \oplus G_2$ be the synchronized product between two SOGs G_1 and G_2 associated with two RCoWF-nets N_1 and N_2 respectively. Let $a_1 \oplus a_2$ be a product aggregate of $G_1 \oplus G_2$. Let $E_{sync}(a_i) = (E_t(a_i) \cap UnObs_i) \cup (E_t(a_i) \cap E_t(a_j))$ for $j \neq i$ be the set of transitions that are enabled by a_i locally and still be enabled after composition. When $E_{sync}(a_i) \neq E_t(a_i)$, the $M_f(a_i)$ and $T_f(a_i)$ must be updated as follows:*

- $M_f(a_i) = \bigcup_{t \in E_{sync}(a_i)} SaturatePre(Pred(M_f(Succ(a_i.S, t)), t), a_i)$ *where* $Pred(S, t) = \{s \mid \exists s' \in S \colon s \xrightarrow{t} s'\}$ *the set of states leading to any state of S by the firing of t;*

- $T_f(a_i) = \{t \in UnObs_i \mid Succ(M_f(a_i), t) \cap M_f(a_i) \neq \emptyset\} \bigcup_{t_i \in E_{sync}(a_i) \setminus UnObs} \{t_i, \mid \exists a \in AggSucc(a_i, t_i)$ *s.t.* $M_f(a) \neq \emptyset\}$ *where* $AggSucc(a, t) = \{a' \in A \mid a \xrightarrow{t} a'\}$ *the set of aggregates reachable from a by the firing of t.*

Note that the $SaturatePre(S, a)$ has been used in Algorithm 3 and allows to saturate from the set of states S within the aggregate a by firing unobserved transitions only.

Thus, the set $T_f(a)$ has to be kept as an attribute of each aggregate for the composition, and it is updated on-the-fly, during the composition of SOGs. As for the E_t attribute (see previous subsection), we do not consider that publishing this set represents a violation of the privacy of the underlying process since it could consist of obsolete names which are not necessarily the real names of the transitions. The T_f attribute of a product aggregate $a_1 \oplus a_2$ is then deduced from the $T_f(a_1)$ and $T_f(a_2)$ (possibly updated on-the-fly) as follows: $T_f(a_1 \oplus a_2) = T_f(a_1) \cup T_f(a_2)$. Now that we showed how the relaxed, the weak and the easy soundness properties can be analyzed on the SOG's synchronized product, we define, in the following, these properties directly on the SOG product.

Definition 35 *Let $\mathcal{N}_i$, for $i = 1, 2$, be two RCoWF-nets. Let $\mathcal{G}_i = \langle \mathcal{A}_i, a_{0i}, \rightarrow_i, \Omega_i \rangle$ be the SOG corresponding to $\mathcal{N}_i$.*

- *if* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *are relaxed sound, then* $\mathcal{G}_1 \oplus \mathcal{G}_2$ *is relaxed sound iff* $\bigcup_{a \in \mathcal{A}} T_f(a) = \bigcup_{i=1...2} \bigcup_{a_i \in \mathcal{A}_i} T_f(a_i)$.

- *if* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *are weak sound then* $\mathcal{G}_1 \oplus \mathcal{G}_2$ *is weak sound iff* $\forall a \in \mathcal{A}$, $\emptyset \notin a.\lambda \wedge \exists a_f \in \Omega \mid a_f \in R(a)$.

- $\mathcal{G}_1 \oplus \mathcal{G}_2$ *is easy sound iff* $\Omega \neq \emptyset$.

Let us consider the example of Figure 5.4 where N_1 and N_2 are both sound locally. The general and the weak soundness are both violated after composition. This is due to the interlock which is visible in the SOG's synchronized product of Figure 5.7 (aggregate $\langle a_3, 36, a_3' \rangle$). However, the relaxed and the easy soundness are preserved by composition in spite of the existence of the interlock.

Then, the equivalence between checking all the three variants of soundness property of a composite RCoWF-net $\mathcal{N}_1 \oplus \mathcal{N}_2$ and checking them on the composite SOG $\mathcal{G}_1 \oplus \mathcal{G}_2$ is established in the following theorem. Its proof is direct if we take into account Theorem 5.2.2, the way of computing the attributes (λ and T_f) of a product aggregate and Defintion 35.

Theorem 5.3.3 *Let* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *be two RCoWF- nets. Let* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *be two SOGs corresponding to* $\mathcal{N}_1$ *and* $\mathcal{N}_2$.

- *if* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *are relaxed sound then* $\mathcal{N}_1 \oplus \mathcal{N}_2$ *is relaxed sound* $\Leftrightarrow$ $\mathcal{G}_1 \oplus \mathcal{G}_2$ *is relaxed sound.*

- *if* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *are weak sound then* $\mathcal{N}_1 \oplus \mathcal{N}_2$ *is weak sound* $\Leftrightarrow$ $\mathcal{G}_1 \oplus \mathcal{G}_2$ *is weak sound.*

- *if* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *are easy sound then* $\mathcal{N}_1 \oplus \mathcal{N}_2$ *is easy sound* $\Leftrightarrow$ $\mathcal{G}_1 \oplus \mathcal{G}_2$ *is easy sound.*

5.4 Conclusion

We addressed in this Chapter the problem of abstracting and checking the correctness of IEBP modularly. First, we showed that, depending on the properties to be checked, and depending on the collaboration nature (synchronous/asynchronous/sharing resources), a revisited SOG is defined. Such a SOG still continue to be suitable to preserve the privacy of each component, while containing the sufficient and the necessary information to allow the verification of the whole IEBP. Our bottom-up approach has been presented for specific properties expressed with the LTL logic and for soundness properties.

Chapter 6

Implementation and Experimental Results

Contents

6.1 Introduction **89**

6.2 Verification of Soundness Properties **91**

6.3 Verification of LTL Property **95**

6.3.1 Implementation 95

6.3.2 Experimental results 95

6.4 Conclusion **102**

6.1 Introduction

In Chapter 4 and Chapter 5, we presented SOG-based approaches and algorithms for the abstraction and the verification of business processes from both local and global point of views respectively. The modular approach is illustrated by Figure 6.1 (left hand side). The implementation of the corresponding tools, written in $c++$, follow the same bottom up scheme. In a top-down approach, the whole BP model is available and a SOG-based verification can be processed using the right hand side of Figure 6.1. The input of this approach is the whole RCoWF-net (composed by two RCoWF-nets) from which the SOG can be built, even without computing first the corresponding reachability graph, and analyzed equivalently. Since, in our approach, the whole model is not available, each component is supplied separately from the others. In our implementation, we can either supply the RCoWF-net of each component or the corresponding SOGs. The RCoWFnets can be represented in different format languages (e.g. PROD [2], PNML [15], or GrML [7]). We note that we do not provide any graphical user interface for painting a Petri net. However, several graphical Petri net modeling tools are able to export/import files that can be read by our tool or translated into one of the supported formats. Once parsed and proved to be correct syntactically, these inputs are processed leading to the

synchronized product of the corresponding SOGs. This SOGs product is built on the fly and over the appropriate set of observed elements depending on the property to be checked. If we are interested in soundness properties, then the SOGs are built over the interface transitions only. Otherwise (for LTL properties), the appropriate atomic propositions and transitions are observed in addition to the interface transitions. In order to increase the efficiency of our approach, each aggregate is encoded with a BDD and all the operations manipulating the aggregates are based on sets operations. For that, we use an existing Binary Decision Diagram library, named *buddy* [1], which offers highly efficient vectorized BDD operations, dynamic variable reordering, automated garbage collection, and a C++ interface with automatic reference counting. In our tool, we handle all kinds of communication (synchronous, asynchronous and sharing of resources) and the number and/or types of the inputs change accordingly. Once the property is checked and the analysis is finished, our tool reports some important results of the analysis (e.g. the SOG construction time) and allows for a textual description of the whole explored SOG. Notice that although our modular tool can be used for the local verification, we still use the non modular version of the tool when we deal with the local verification until we adapt the modular one for this particular case.

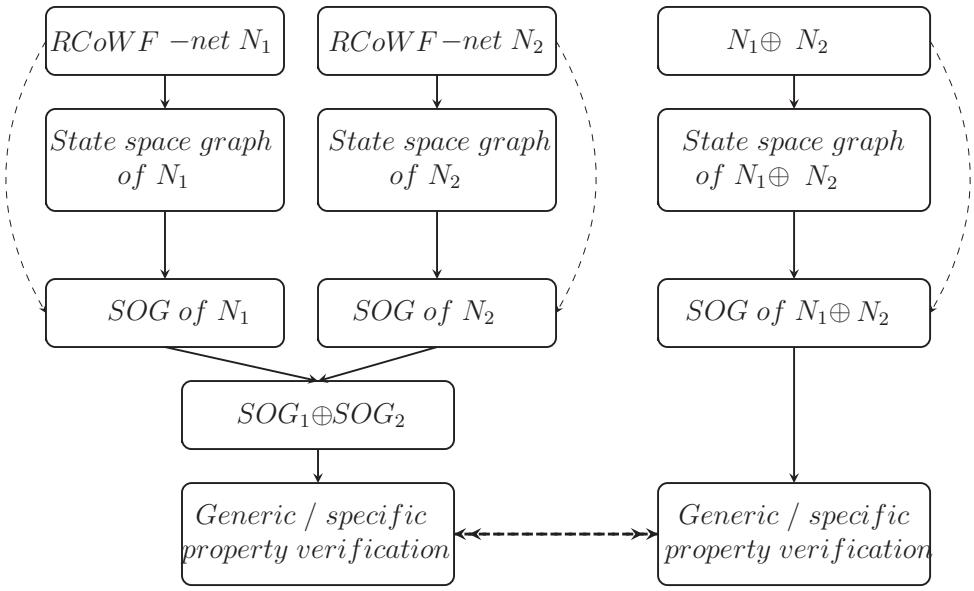


Figure 6.1: Schema of our approach

In the following we first describe our implementation for checking soundness properties and then we detail the LTL model checking implementation. For both kinds of properties, we discuss some preliminary experimental results compared to those obtained by related tools. Since the compared tools are not modular i.e. take as input the whole model, and in order to have a fair comparaison, we give our experimental results following both modular

and non modular approaches. The first are given to show the feasibility and the efficiency of our modular approach.

6.2 Verification of Soundness Properties

We conducted some preliminary experimentations. To the best to our knowledge, there is no available tool allowing the checking of the different variants of soundness presented in this work. Thus, we compare our results to two well known tools with respect to the general soundness property. The first tool is LoLA [113], which is a Petri net model checker which decides about numerous properties for a given Petri net. The second tool is Woflan [121, 122], which verifies the correctness of the soundness property by using structural Petri nets reductions.

Table 6.2 and Table 6.1 provide the obtained experimental results for our tool and the Woflan and LoLA ones. In Table 6.1, the following modular case studies are considered:

- Two versions of a model of Subcontractor composed with Contractor model (C+CS) taken from [68];
- An example of reservation in travel agency (Res);
- An example of reservation of trips (ResTrip) in Figure 5.1 of Chapter 5,
- A workflow (interOrg) taken from [122] that involves four business partners: a customer, a producer and two suppliers;
- A route planning service (Planning) that acts as mediator and a customer's service taken from [89];
- The electronic bookstore (BookStor) which is a contract between the customer, the bookstore, the publisher, and the shipper taken from [131];
- The example (Registra) of a contract organizing the registration process for a passport or an ID card taken from [126].

Table 6.1 illustrates the size of the considered oWF-net models (in terms of number of places and transitions) as well as the size of their reachability state graphs RG (in terms of number of nodes (S) and arcs (E)). To check soundness, LoLA performs with two runs (represented by two rows in the LoLA column): the first run checks for local deadlock states, and the second run checks for lack of synchronization. Since LoLA is a behavioral approach (based on the traversal of the state space), it uses several state-space reduction techniques to make the state space inspection feasible. Concerning the Woflan tool, it

poses syntactic restrictions on the Petri nets it can analyze. Especially, it imposes that each net must have a unique initial place and a unique terminal place (workflow net). Woflan is based on a structural approach which explores a reduced state space resulting from prior application of structural reduction rules. Note that as for our approach, both tools performs on-the-fly (the checking process is stopped as soon as the soundness is proved to be false). When the model does not satisfy the soundness property only the size of the built part of the graph is given. In addition to the size of the SOG of the original composite service, Table 6.1 gives the size of the SOG obtained by composition. We note that the modular SOG is equivalent to the non-modular one in terms of size (see Theorem 5.2.1 and Theorem 5.2.2). In Table 6.2, other case studies are considered. The corresponding models are not modular (that is why there is no column for modular SOGs) and can be found at the IBM WebSphere Business Modeler tool [46]. These models are taken from different domains (financial services, automotive, telecommunications). Except for the modular SOG, this table contains the same information as Table 6.1.

All the three approaches perform similarly fast, but the SOG is always (at least for the tested examples) smaller than the LoLA's and Woflan's graphs and this is regardless the satisfaction or not of the soundness property (column Sound = T). Both the non-modular SOGs and the modular SOGs are smaller than those of LoLA and Woflan.

Model				Modular SOG				Non Modular SOG				LoLA			Woflan	
Nom	Sound	Places	Trans	Obs	S	E	T(s)	Obs	S	E	T(s)	S	E	T(s)	S	T(s)
C+SC1	F	25	16	4/4	9	8	<1	8	9	8	<1	22	26	<1	18	<1
												1	0	<1		
C+SC2	V	28	23	4/4	11	12	<1	8	11	12	<1	23	25	<1	21	<1
												23	25	<1		
Res	V	28	33	8/8	17	17	<1	16	17	17	<1	19	21	<1	19	<1
												15	17	<1		
ResTrip	F	19	21	4/4	10	12	<1	8	10	12	<1	24	27	<1	16	<1
												1	0	<1		
interOrg	T	39	26	5/9/2/2	18	26	<1	18	18	26	<1	16	15	<1	28	<1
												58	59	<1		
Planning	T	21	13	2/2/4	10	9	<1	8	10	9	<1	10	10	<1	28	<1
												19	21	<1		
Bookstore	T	47	31	6/9/5/6	39	53	<1	26	39	53	<1	48	70	<1	41	<1
												30	32	<1		
Registra	T	24	18	8/8/2	17	18	<1	18	17	18	<1	17	18	<1	21	<1
												17	18	<1		

Table 6.1: *Experimental results: modular SOG*

Model				RG		SOG				LoLA			Woflan	
Nom	Sound	Places	Trans	S	E	Obs	S	E	T(s)	S	E	T(s)	S	T(s)
b1.1	F	37	26	147	331	5	6	6	<1	25	26	<1	30	<1
										3	2	<1		
b2.1	F	67	56	70	60	22	8	7	<1	18	17	<1	11	<1
										1	0	<1		
a.1	T	33	19	20	19	3	4	3	<1	19	18	<1	22	<1
										19	18	<1		
a.2	T	21	15	15	15	9	9	9	<1	17	16	<1	32	<1
										13	12	<1		
a.3	T	35	19	19	19	9	9	9	<1	18	18	<1	19	<1
										12	12	< 1		
b1.2	T	32	19	492	1869	3	4	3	<1	19	18	<1	24	<1
										4	3	<1		
b2.2	T	28	18	34	27	8	13	16	<1	32	32	<1	53	<1
										48	49	<1		
b3.1	F	100	64	345	656	31	31	30	10	109	109	30	192	<1
										34	34	30		

Table 6.2: *Experimental results: non-modular*

6.3 Verification of LTL Property

6.3.1 Implementation

Our implementation regarding the verification of LTL formulae on IEBPs is based on the automata theoretic approach to LTL model checking (see Figure 6.2). The inputs are an LTL formula φ and the description of the IEBP given through its different components' RCoWF-nets $N_1 \dots N_n$ (the different corresponding SOGs can be given alternatively). The model checking problem is reduced to an on-the-fly emptiness check processed on the synchronized product of the Büchi automaton $A_{\neg\varphi}$ corresponding to the negation of the formula, and the automaton A_{SOG} of the SOGs product (checking whether $L(A_{\neg\varphi} \oplus A_{SOG}) = \emptyset$ or not). The later is directly obtained by the synchronization of the components' SOGs. Using our approach (Chapter 5), we implemented the different steps of the right hand side part of Figure 6.2. If the emptiness check returns true, then the formula is proved to be satisfied by the IEBP. Otherwise, a counterexample (a possible run that violate φ) is supplied to the user. The left hand side part is realized using Spot [40]: an object-oriented model checking library written in C++ which offer a set of building blocks to experiment with and to develop a model checker program. Spot proposes different algorithms for both the translation of an LTL formula into an Büchi automaton and the emptiness check problem.

The implemented tool offers several options to the user via the a command-line interface.

- $-aALGO$: this option lets the user choose the emptiness check algorithm ALGO to be applied
- $-c$: this option displays the number of states and edges of the reachability graph
- $-e$: this option displays a sequence (if any) of the net satisfying the formula (implies -f or -F)
- $-g$: this option displays the SOG graph.

6.3.2 Experimental results

In order to experiment our modular LTL model checker, we consider here three parametrized examples. In the first the parameter represents the number of resources initially available while, in the two next examples, the parameter represents the number of collaborating components. As the obtained results for these three examples are homogenous, we give a common interpretation at the end of this section.

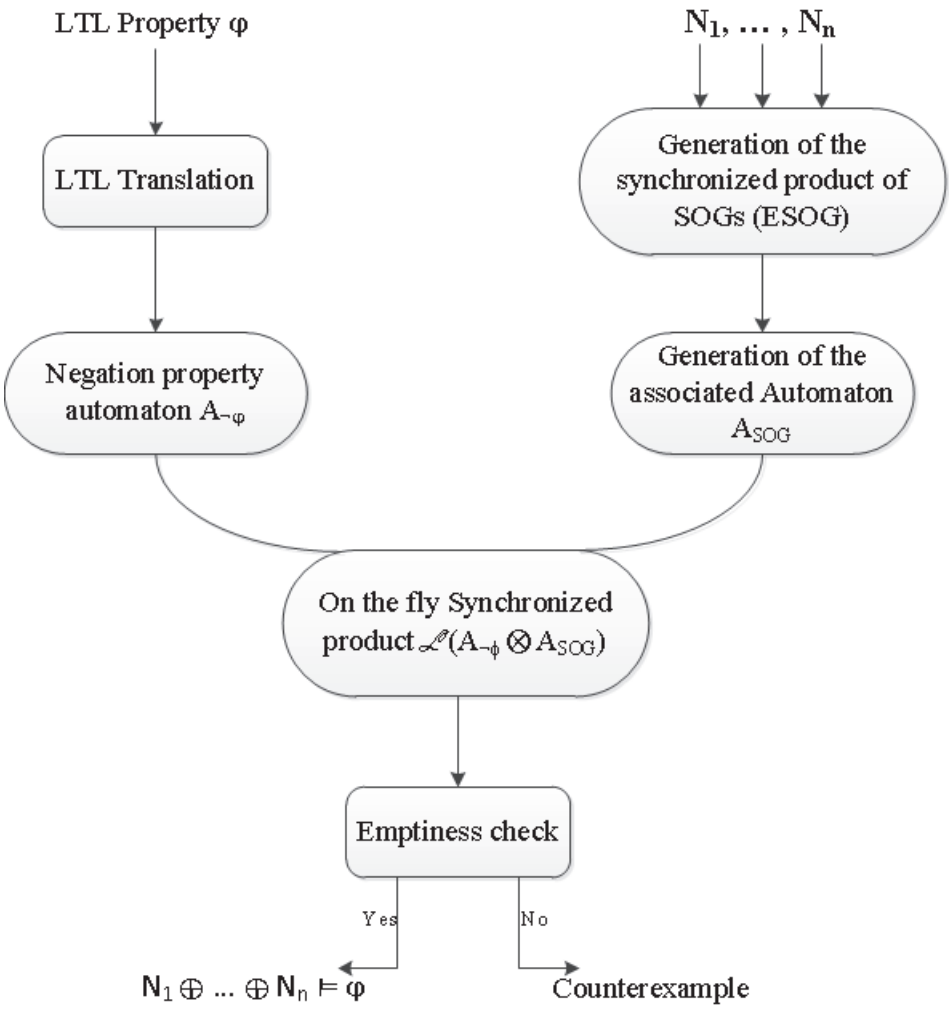
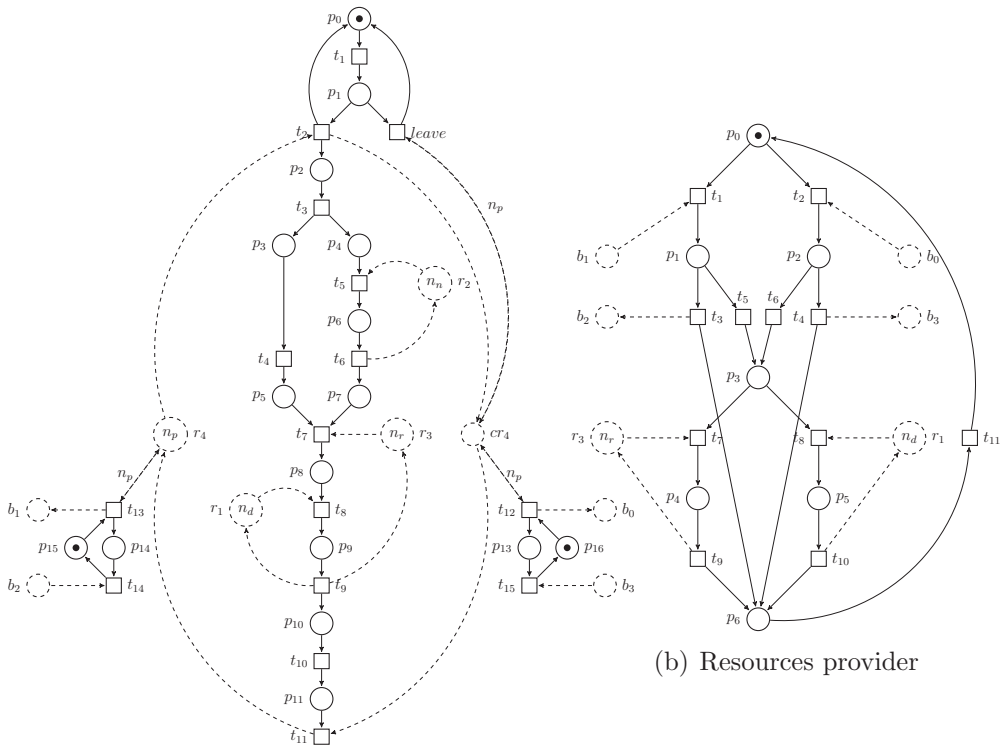


Figure 6.2: Illustration of our Model Checker

Emergency Service Example

Figure 6.3 illustrates the RCoWF-net model of an emergency care service (our design follows the textual description given in [119]). In the first model (Figure 6.3(a)), the injured people will be taken to emergency rooms for immediate treatment. Resources to be modeled include physicians, nurses, and examining rooms, as well as the resource consumers, the patients. When a patient first arrives at emergency room (transition t_1), he/she proceeds to check in. Then, the receptionist checks the resources and number of current patients to determine a waiting time. If the waiting time is bigger than a certain value, then the newly arrived patient would not be admitted into the emergency room (transition *leave*). The responsable of the service can decide to ask for more resources by firing transition t_{12}. Otherwise, the patient would check into the emergency room and be given paperwork to fill out (transition t_3). After the patient completes the



(a) Emergency medical care service

(b) Resources provider

Figure 6.3: Sharing resources in an emergency medical care service

paperwork, he/she would wait to be treated (transition t_4) and an available nurse would start processing the paperwork (transition t_5). When the nurse completes the process (transition t_6) and there is an examining room available for the patient, then the patient would enter the room (transition t_7). When a physician becomes available, he/she would start examining the patient in the examining room (transition t_8). After the completion of the treatment (transition t_9), the patient proceeds to check (transition t_{10}) out and leave (transition t_{11}) the emergency room. When there is no client waiting, the responsable of the service can decide to contact the resources provider, depicted in Firgue 6.3(b), (by firing transition t_{13}) to offer a resource (a room or a doctor) in case other services need it. The resource provider, shares with the any service of the hospital (in particular the workflow of Figure 6.3(a)) the rooms and physicians resources (r_3 and r_1 respectively), and communicates with it asynchronously via buffers b_0 and b_1. A message in b_0 is an authorization from the emergency service to the provider to use its resources and to give them to an other service which needs it.

The choice of the number of initially available resources (the initial marking) allows to make the size of the state space larger and check how our implementation behaves against complex systems. To the best of our knowledge, there is no existing tool allowing to

Model		Modular SOG		Non Modular SOG		LoLA	
N	RG	States	Time(s)	States	Time(s)	States	Time(s)
(1,1,1,1)	2844	73	3	7	1	17	1
(5,3,4,2)	73200	20	3	9	3	38	1
(2,2,3,3)	157816	48	8	13	3	49	1
(2,2,1,4)	850928	119	3	24	2	58	1
(4,3,2,4)	1.06101 e^{+6}	15	2	24	4	62	1

Table 6.3: *Model checking of an unsatisfied formula* ($G(t_7 \implies F\ t_9)$)

Model		Modular SOG		Non Modular SOG		LoLA	
N	RG	States	Time(s)	States	Time(s)	States	Time(s)
(1,1,1,1)	2844	706	3	10	3	1422	1
(5,3,4,2)	73200	93	5	1697	7	9150	1
(2,2,3,3)	157816	3840	20	14	3	39454	1
(2,2,1,4)	850928	8090	26	37	3	106366	1
(4,3,2,4)	1.06101 e^{+6}	53975	300	122	14	132626	1

Table 6.4: *Model checking of a satisfied formula* ($G(t_7 \implies F\ (t_9 \vee t_{12} \vee leave)$)

check LTL properties modularly (as we do), in the sense that the knowledge of the whole model is necessary to use the existing LTL model checkers. Thus, in order to validate our approach and to show that our results are comparable in terms of performances to existing tools, we give the results obtained in both modular and non modular ways. As for the soundness property, we compare our prototype's results to LoLA tool.

We check two LTL properties on our model: the first one, expressed with the LTL formula is $G(t_7 \implies F\ t_9)$ expresses that each time resource r_3 is hold, then it will be eventually released in the future. This first formula is not satisfied. In fact, for instance for the case where the initial marking of each resource place is 1, the firing of the infinite run $t_2^1.t_7^1.(t_{12}^1.t_2^2.t_4^2.t_{15}^1)^\omega$ (where t_i^1 is the transition t_i of the left hand side model, and t_i^2 is the transition t_i of the right hand side model) does not satisfy the formula. The second LTL formula is $G(t_7 \implies F\ (t_9 \vee t_{12} \vee leave)$ and expresses that the firing of t_7 is always followed in the future by either the firing of t_9, or the firing of t_{12} or the firing of *leave*. This formula is satisfied by our model. However, since LoLA allows to check state-based LTL formulae, these two event-based LTL formulae are not expressible with this logic. The checked formulae with LoLA are obtained by replacing each involved transition by the set of its output places. This way is possible here because of the simplicity of the model, but is not obvious in general.

Table 6.3 and Table 6.4 present the results we obtained when checking these two LTL formulae respectively. Each table contains four multi-columns: the first one gives

information about the parameterized model i.e., the initial marking of the resources places (r_1, r_2, r_3, r_4) and the number of reachable markings. Then, three approaches are compared: first the modular approach where we consider three modules (the two to be composed and the interface medium) and we build and compose their SOGs. In this case, there are 15 observed transitions in total. The second part processes the model checking of the formula on the SOG of the composite model. In this case, only the transitions occurring in the formula are observed (2 transitions and 4 transitions respectively). The last part of each table gives the results obtained with LoLA tool. For each verification line, we compare the number of states that are visited during the verification, and the verification time. Note, that, in the second table, this number corresponds to the size of the whole synchronized product between the reachability graph and the Büchi automaton of (the negation of) the formula.

Producer/Consumer and Reservation Trip Example

Here, we consider two parameterized models representing the reservation trip example (Figure 5.1 of Chapter 5) and a well known toy model representing the producer consumer. The obtained results are presented in Table 6.5 and Table 6.6 respectively. Each table is divided into two sections: the first one concerns the experimental results for a satisfied formula and the second one concerns a non satisfied formula. In Table 6.5 (resp. Table 6.6) the number of customer collaborating with the reservation trip model (resp. the number of consumers communicated with the producer) is given in the first column. Increasing the value of this parameter implies a bigger size for both the model and the formula to be checked. Indeed, the chosen formula involves all the added components. These tables contain the same information as those presented previously for the soundness property.

Interpretation of the experimental results

Comparing to LoLA, The obtained results for the LTL model checking process show that the SOG is always smaller than LoLA's graph and this is regardless the satisfaction or not of the LTL property. However, our approach consumes more time than LoLA in both modular and non modular approach. In addition to the fact that LoLA uses several reduction techniques, this can be explained by the following: It is possible that a marking belongs to different aggregates which can lead to compute some sets of states several times. Moreover, in the case of the modular approach, the fact that the number of observed transitions is greater than the one used in the non modular approach, leads to a bigger number of aggregates, and hence bigger consumption of time. We can improve, the consumption of time of the modular construction by taking as inputs the SOGs (a priori

computed) of the components to be composed. In addition, we recall that a global model checker such as LoLA is simply unusable in our context since the whole model is supposed to be unavailable. Finally, the advantage of our approach is that we handle *hybrid* LTL properties which is, to the best of our knowledge, not allowed by the existing verification tools (including LoLA).

Model					Modular SOG			Non Modular SOG			LoLA		
N	Places	Trans	RG	EG	Obs	States	Time(s)	Obs	States	Time(s)	States	Edges	Time(s)
RT 1	29	30	43	74	12	19	3	4	6	2	34	137	1
RT 2	43	33	780	895	24	341	3	8	54	3	820	7415	1
RT 3	53	54	11419	8833	36	4495	21	12	531	12	11419	176439	1
RT 4	63	66	163393	79533	48	50569	282	16	5195	100	137015	3201710	4
RT 5	73	78	1.4938 e^6	679581	60	507931	5h2m	20	31244	210	1237490	40804030	85
RT 1	29	30	43	76	12	19	2	2	8	2	22	22	1
RT 2	43	33	780	937	24	99	3	4	10	2	198	272	1
RT 3	53	54	11419	9355	36	762	6	6	24	2	2373	4261	1
RT 4	63	66	163393	84799	48	6242	52	8	106	4	26558	57644	1
RT 5	73	78	1.4938 e^6	727663	60	50660	698	10	694	124	220973	555332	1

Table 6.5: *Experimental results: Reservation Trip*

Model					Modular SOG			Non Modular SOG			LoLA		
N	Places	Trans	RG	EG	Obs	States	Time(s)	Obs	States	Time(s)	States	Edges	Time(s)
PR 1	31	25	123	241	10	29	2	5	7	2	123	879	1
PR 2	47	40	1440	2495	20	194	3	9	61	3	1440	15090	1
PR 3	62	54	12772	21166	30	435	3	13	610	3	12772	183901	1
PR 4	77	68	101232	163886	40	2038	13	17	6155	43	101232	1993552	6
PR 5	92	82	754480 e^6	1203874	50	9553	77	21	64194	151	754480	18101399	58
PR 1	31	25	123	241	10	24	2	2	7	2	23	22	1
PR 2	47	40	1440	2495	20	119	2	4	15	2	44	43	1
PR 3	62	54	12772	21166	30	617	4	6	45	2	65	64	1
PR 4	77	68	101232	163886	40	3114	9	8	179	7	86	85	1
PR 5	92	82	754480 e^6	1203874	50	5924	45	10	15	3	107	106	1

Table 6.6: *Experimental results: Producer-Consumer*

6.4 Conclusion

We presented in this Chapter our implemented SOG-based approach dedicated to the abstraction and the verification of IEBPs. All necessary algorithms detailed in Chapter 4 and Chapter 5 have been implemented. Regarding to both soundness properties and LTL formulae, the preliminary experimentations show that our approach outperform both LoLA and Woflan tool in terms of size of the explored graph. However, for LTL properties, although our implemented tool consumes more time, it have the advantage of dealing with the *hybrid* LTL properties unlike the other tools. Moreover, we aim to confront our approach to some realistic domains in order we will be able to confirm its efficiency. on specific domains.

Chapter 7

General Conclusion and Perspectives

Contents

7.1 Summary 103

7.2 Future Work 105

7.1 Summary

Formal verification is hot research topic since three decades. It is performed at design time and ensures the correction of a system's model with respect to some desired properties. It is highly recommended in critical fields (Transport, Hardware, Communication protocols, ...) where a bug in the system can lead to disasters. Nowadays, the formal verification is however increasingly widespread and used in several other domains (Business processes, Web services, ...). In this thesis, we deal with formal verification, and especially model checking, in the context of IEBPs. We are indeed convinced that developing such approaches for a specific domain can be highly useful from two points of view: First, one can take into account the specific properties/constraints of the target domain in order to design new efficient approaches. Second, domain-specific approaches could bring new ideas to improve the verification in the general case. This would ideally create a virtuous circle where general and specific-domain verification approaches enrich each other.

An IEBP can be seen as the composition of severals independent business processes, defined as a flow of related activities that together create a customer value. Collaboration between companies are considered necessary in a business environment, where each company focuses on its competitive advantage, performs only those functions, for which it has expert skills. The different involved companies must operate in a network in order to complement their offering through partners and suppliers, and hence to leverage (resp. compensate) their strength (resp. weakness). Typically, there are n business partners which are involved in one 'global' IEBP. Each of the partners has its own 'local' business process (designed separately) which is private, i.e., each component has no knowledge about the local process of the partners. The whole IEBP model being unavailable, the sole possible approach to check its correction is necessarily a bottom-up one i.e. the

IEBP model is obtained by composing the individual components' models. However, this is not possible because each component wants to hide the trade secrets of its service (private view) for privacy reasons. Such a bottom-up approach should be then built on an appropriate public view (instead of the private view) of each component allowing both the respect of the privacy constraint and the verification of the whole IEBP. The question how to formalize such a public view (abstract model) and how to use it to check the correctness of an IEBP is the core of the work presented in this thesis.

The abstract model we propose in this work is based on the Symbolic Observation Graphs (SOG). Originally [55, 75], the SOG has been defined as a hybrid structure abstracting the state graph of a system, and has been used for model checking linear time properties. We propose to represent each component of the IEBP by a SOG such that only the behavior regarding the interface (the collaborative activities) of each component is public. This is ensured by hiding the local behavior (the execution of the local/non collaborative activities) inside the nodes (aggregates) of the SOG. The correction criterium of the IEBP is dealt with in this work regarding two kinds of properties: generic and specific properties. As generic properties, we studied the well known soundness property and some of its variants. Specific properties, however, were defined here as LTL logic formulae expressed over concrete elements of the BP model. For both kinds of properties, the contributions of this thesis were to revisit the SOG structure in order to reach the following goals: (1) reduce the complexity of the verification process locally to each component by using the corresponding public view (the SOG) instead of the original explicit state space, and (2) reduce the verification of the whole IEBP (whose model is not available) to the analysis of the composition of its components' SOGs. The main difficulty of the second goal is the non preservation of the considered correctness properties by composition. Indeed, the fact that each component is correct does not guarantee that the composite model is correct. We established which are the sufficient and necessary information, to be computed (locally) and added to the aggregates of a SOG, allowing a modular SOG-based verification.

Another important contribution is the extension of our approach to deal with *hybrid* LTL logic (where a formula could involve both state- and event-based atomic propositions). The semantics of this extended logic allows to mix the state-based and the even-based semantics which are interchangeable (an event can be encoded as a change in state variables, and likewise one can equip a state with different events to reflect different values of its internal variables). However, it is not fairly trivial to switch from one representation to the other, and it can lead to a significant state space enlargement because of the size of the translated formula.

Finally, our goal to make general and domain-specific verification enrich each other is

somehow reached. Indeed, our modular verification approach can be used in any other domain and especially for loosely coupled concurrent systems.

Our approach for checking the correctness of IEBP w.r.t. soundness and LTL formulae, for both local and global point of views, has been implemented within a prototype and has been validated through examples from different business domains. The obtained results are encouraging and open several improvement issues. A part of the prototype is already integrated in the CosyVerif (http ://www.CosyVerif.org) and we are currently studying the integration of the whole implementation.

7.2 Future Work

There are several open issues of the work presented in this thesis. From technical point of view, our implementation needs to be revisited in order to achieve the following objectives: First, we plan to integrate the different functionalities of our tool in order to have a unique framework allowing to check the correctness of an IEBP process, and where the user can choose, according to his desire, the appropriate module. In this unique framework, we would like to generalize the possibility to give, as input, either the RCoWF-nets of the different IEBP components, or the corresponding SOGs which have been built independently. We think that this feature will lead to a significant improvement of the consumed time during the construction and the verification of the SOGs synchronized product. This task will facilitate the confrontation of our approach to realistic applications, thing we could not do during the thesis (although we tried to have significant examples in terms of models and behavior). Second, several orthogonal reduction techniques could be envisaged in order to improve the performances of our approach (in terms of memory and time consumption) such as partial order reduction (e.g., [13, 120]), or distributed/parallel model checking (following for instance the approaches presented in [45, 79]. In the first perspective, the independence between the observed transitions of the SOG can be exploited in order to avoid to build the synchronized product entirely. The second perspective allows to distribute the construction and the verification tasks on several machines/cores.

Now, from theoretical point of view, our approach can be extended in two directions: The first direction is related to the application domain and consists in adapting our approach regarding a new domain with specific requirements/constraints. Although it is not detailed in this manuscript, we already applied our approach in order to check compatibility between Web services and hence to check the correctness of composite services ([69, 78, 71, 74]). We are currently studying ([73]) the extension of our approach to processes in a Cloud environment where several challenging specific properties have

to be studied. First, our verification modular approach can help in managing Cloud resources that are shared by different Cloud service-based BPs, by checking, at design time, some fairness LTL properties on the resources. Second, we plan to study two specific Cloud properties, namely *elasticity* and *multi-tenancy*. Elasticity in the Cloud ensures the provisioning of necessary and sufficient resources in such a way that a Cloud process continues running smoothly even when the number of its uses scales up or down, thereby avoiding under-utilization and over-utilization of resources. The multi-tenancy can be defined at different levels (resources, process instances, process, ...) and expresses the fact that different Cloud processes share some features.

The second direction we plan to investigate in the future concerns the models we use to specify the underlying processes. A same bottom-up approach can be designed for enriched (extended) Petri nets. For instance, on can consider the extension of the approach to Colored Petri nets [63](e.g. when the desired properties require to distinguish the identity of the process instances) or to Timed/Temporal Petri nets [94, 108] (when the process and the properties involves time explicitly).

Bibliography

[1] Buddy: A bdd package. `http://buddy.sourceforge.net/manual/main.html`.

[2] Prod 3.4.01 an advanced tool for efficient reachability analysis. `http://www.tcs.hut.fi/Software/prod/`.

[3] Wil M. P. van der Aalst. Verification of workflow nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, ICATPN '97, pages 407–426, London, UK, UK, 1997. Springer-Verlag.

[4] Wil M. P. van der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 161–183, London, UK, UK, 2000. Springer-Verlag.

[5] Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors. *Business Process Management, Models, Techniques, and Empirical Studies*, London, UK, UK, 2000. Springer-Verlag.

[6] Jean-Raymond Abrial, Matthew K. O. Lee, David Neilson, P. N. Scharbach, and Ib Holm Sørensen. The b-method. In *VDM '91 - Formal Software Development, 4th International Symposium of VDM Europe, Noordwijkerhout, The Netherlands, October 21-25, 1991, Proceedings, Volume 2: Tutorials*, pages 398–405, 1991.

[7] etienne Andre, Benoit Barbot, Clement Démoulins, LomMessan Hillah, Francis Hulin-Hubard, Fabrice Kordon, Alban Linard, and Laure Petrucci. A modular approach for reusing formalisms in verification tools of concurrent systems. In Lindsay Groves and Jing Sun, editors, *Formal Methods and Software Engineering*, volume 8144 of *Lecture Notes in Computer Science*, pages 199–214. Springer Berlin Heidelberg, 2013.

[8] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing ltl semantics for runtime verification. *J. Log. and Comput.*, 20(3):651–674, June 2010.

[9] Jörg Becker, Martin Kugeler, and Michael Rosemann, editors. *Process management: a guide for the design of business processes.* 2003.

[10] Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. Meteor: A successful application of b in a large project. In JeannetteM. Wing, Jim Woodcock, and Jim Davies, editors, *FM99, Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 369–387. Springer Berlin Heidelberg, 1999.

[11] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and MASSIMO MECELLA. Automatic service composition based on behavioral descriptions. *INTERNATIONAL JOURNAL OF COOPERATIVE INFORMATION SYSTEMS*, 14:2005, 2005.

[12] Yves Bertot, Pierre Casteran, Gerard (informaticien) Huet, and Christine Paulin-Mohring. *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Texts in theoretical computer science. Springer, Berlin, New York, 2004. Donnees complementaires http://coq.inria.fr.

[13] Girish Bhat and Doron Peled. Adding partial orders to linear temporal logic. *Fundam. Inf.*, 36(1):1–21, January 1998.

[14] Armin Biere, Edmund M. Clarke, and Yunshan Zhu. Multiple state and single state tableaux for combining local and global model checking. In *IN CORRECT SYSTEM DESIGN*, pages 163–179. Springer, 1999.

[15] Jonathan Billington, Søren Christensen, Kees Van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian Stehno, and Michael Weber. The petri net markup language: Concepts, technology, and tools. In *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets*, ICATPN'03, pages 483–505, Berlin, Heidelberg, 2003. Springer-Verlag.

[16] Graham Birtwistle and P. A. Subrahmanyam, editors. *Current Trends in Hardware Verification and Automated Theorem Proving*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[17] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[18] C. E. Brown. *Automated Reasoning in Higher-order Logic: Set Comprehension and Extensionality in Church's Type Theory*. College Publications, 2007.

[19] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[20] J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr .)*, pages 1–11. Stanford Univ. Press, Stanford, Calif., 1962.

[21] Tevfik Bultan, Jianwen Su, and Xiang Fu. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1):18–25, 2006.

[22] S. Cheikhrouhou, S. Kallel, and M. Jmaiel. Toward a verification of time-centric business process models. In *WETICE Conference (WETICE), 2014 IEEE 23rd International*, pages 326–331, June 2014.

[23] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. W3c note, March 2001.

[24] Theodore H. Clark and Donna B. Stoddard. Interorganizational business process redesign: Merging technological and process innovation. *J. Manage. Inf. Syst.*, 13(2):9–28, September 1996.

[25] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, April 1986.

[26] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, UK, 1982. Springer-Verlag.

[27] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *25 Years of Model Checking*, pages 196–215, 2008.

[28] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Form. Methods Syst. Des.*, 1(2-3):275–288, October 1992.

[29] Jean-Michel Couvreur. On-the-fly verification of linear temporal logic. In JeannetteM. Wing, Jim Woodcock, and Jim Davies, editors, *FM: Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 253–271. Springer Berlin Heidelberg, 1999.

[30] D. Cyrluk, S. Rajan, N. Shankar, , and M.K. Srivas. Effective theorem proving for hardware verification. In *Theorem Provers in Circuit Design (TPCD '94)*, volume 901 of *Lecture Notes in Computer Science*, pages 203–222, Bad Herrenalb, Germany, sep 1994. Springer-Verlag.

[31] Thomas H. Davenport. *Process Innovation: Reengineering Work Through Information Technology*. Harvard Business School Press, Boston, MA, USA, 1993.

[32] Thomas H. Davenport and James E. Short. The new industrial engineering: Information technology and business process redesign. *Sloan Management Review*, 31(4):11–27, 1990.

[33] Juliane Dehnert and Peter Rittgen. Relaxed soundness of business processes. In *International Conference on Advanced Information Systems Engineering*, volume 2068 of *LNCS*, pages 157–170. Springer-Verlag, 2001.

[34] Jörg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, New York, NY, USA, 1995.

[35] Gregorio Diaz, Juan-José Pardo, María-Emilia Cambronero, Valentín Valero, and Fernando Cuartero. Automatic translation of ws-cdl choreographies to timed automata. In *Proceedings of the 2005 International Conference on European Performance Engineering, and Web Services and Formal Methods, International Conference on Formal Techniques for Computer Systems and Business Processes*, pages 230–242, Berlin, Heidelberg, 2005. Springer-Verlag.

[36] Andries Van Dijk. Contracting workflow and protocol patterns. In *in Proc. of Business Process Management Int. Conf*, pages 152–167. Springer, 2003.

[37] B. F. Van Dongen and H. M. W. Verbeek. Verification of epcs: Using reduction rules and petri nets. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE05), volume 3520 of Lecture Notes in Computer Science*, pages 372–386. Springer, 2005.

[38] Yanhua Du, Xitong Li, and PengCheng Xiong. A petri net approach to mediation-aided composition of web services. *IEEE T. Automation Science and Engineering*, 9(2), 2012.

[39] Ziyang Duan, Arthur Bernstein, Philip Lewis, and Shiyong Lu. A model for abstract process specification, verification and composition. In *Proceedings of the 2Nd International Conference on Service Oriented Computing*, ICSOC '04, pages 232–241, New York, NY, USA, 2004. ACM.

[40] Alexandre Duret-Lutz and Denis Poitrenaud. Spot: an extensible model checking library using transition-based generalized büchi automata. In *IN PROC. OF MASCOTS'04*, pages 76–83. IEEE Computer Society.

[41] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proceedings of the 7th Colloquium on*

Automata, Languages and Programming, pages 169–181, London, UK, UK, 1980. Springer-Verlag.

[42] Robert Engel, WilM.P. van der Aalst, Marco Zapletal, Christian Pichler, and Hannes Werthner. Mining inter-organizational business process models from edi messages: A case study from the automotive sector. In *Advanced Information Systems Engineering*, volume 7328 of *Lecture Notes in Computer Science*, pages 222–237. Springer Berlin Heidelberg, 2012.

[43] Rik Eshuis, Paul W. P. J. Grefen, and Sven Till. Structured service composition. In *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, pages 97–112, 2006.

[44] Kousha Etessami. Stutter-invariant languages, omega-automata, and temporal logic. In *Proceedings of the 11th International Conference on Computer Aided Verification*, CAV '99, pages 236–248, London, UK, UK, 1999. Springer-Verlag.

[45] Sami Evangelista, Lars Michael Kristensen, and Laure Petrucci. Multi-threaded explicit state space exploration with state reconstruction. In *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, pages 208–223, 2013.

[46] Dirk Fahland, Cédric Favre, Jana Koehler, Niels Lohmann, Hagen Völzer, and Karsten Wolf. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data Knowl. Eng.*, pages 448–466, 2011.

[47] Kathi Fisler, Ranan Fraer, Gila Kamhi, Moshe Y. Vardi, and Zijiang Yang. Is there a best symbolic cycle-detection algorithm? In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2001, pages 420–434, London, UK, UK, 2001. Springer-Verlag.

[48] Melvin Fitting. *First-order Logic and Automated Theorem Proving (2Nd Ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[49] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting bpel web services. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 621–630, 2004.

[50] Rob Gerth, Doron Peled, Moshe Y. Vardi, R. Gerth, Den Dolech Eindhoven, D. Peled, M. Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *In Protocol Specification Testing and Verification*, pages 3–18. Chapman Hall, 1995.

[51] Yolanda Gil. Workflow composition: Semantic representations for flexible automation, 2006.

[52] Bernard C. Glasson, Igor Hawryszkiewycz, Alan Underwood, and Ron Weber, editors. *Business Process Re-Engineering: Information Systems Opportunities and Challenges, Proceedings of the IFIP TC8 Open Conference on Business Re-engineering: Information Systems Opportunities and Challenges, Queensland Gold Cost, Australia, 8-11 May, 1994*, volume A-54 of *IFIP Transactions*. Elsevier, 1994.

[53] Ursula Goltz, Ruurd Kuiper, and Wojciech Penczek. Propositional temporal logics and equivalences. In W.R. Cleaveland, editor, *CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*, pages 222–236. Springer Berlin Heidelberg, 1992.

[54] Daniela Grigori, Juan Carlos Corrales, and Mokrane Bouzeghoub. Behavioral matchmaking for service retrieval: Application to conversation protocols. *Inf. Syst.*, 33(7-8):681–698, November 2008.

[55] Serge Haddad, Jean-Michel Ilié, and Kais Klai. Design and evaluation of a symbolic and abstraction-based model checker. In *ATVA'04*, LNCS. 196–210, Springer-Verlag, 2004.

[56] Michael Hammer and James Champy. Reengineering the corporation: A manifesto for business revolution. *Business Horizons*, 36(5):90–91, 1993.

[57] Henri Hansen, Wojciech Penczek, and Antti Valmari. Stuttering-insensitive automata for on-the-fly detection of livelock properties. *Electronic Notes in Theoretical Computer Science*, 66(2):178 – 193, 2002. FMICS'02, 7th International {ERCIM} Workshop in Formal Methods for Industrial Critical Systems (ICALP 2002 Satellite Workshop).

[58] Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 514–529. Springer, 1996.

[59] Sebastian Hinz, Karsten Schmidt, and Christian Stahl. Transforming bpel to petri nets. In *BPM'05, vol 3649 of LNCS*. Springer-Verlag, 2005.

[60] Yigal Hoffner, Heiko Ludwig, Ceki Gülcü, and Paul W. P. J. Grefen. An architecture for cross-organizational business processes. In *Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000), Milpitas, California, USA, June 8-9, 2000*, pages 2–11, 2000.

[61] Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, first edition, 2003.

[62] Michael G. Jacobides and Stephan Billinger. Designing the boundaries of the firm: From "make, buy, or ally" to the dynamic benefits of vertical architecture. *Organization Science*, 17(2):249–261, 2006.

[63] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 2*. Springer-Verlag, London, UK, UK, 1995.

[64] Matjaz B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2Nd Edition*. Packt Publishing, 2006.

[65] Roope Kaivola and Antti Valmari. The weakest compositional semantic equivalence preserving nexttime-less linear temporal logic. In *Third International Conference on Concurrency Theory*, CONCUR '92, pages 207–221, London, UK, 1992. Springer-Verlag.

[66] Yonit Kesten, Amir Pnueli, and Li on Raviv. Algorithmic verification of linear temporal logic specifications. In *Proc. 25th Int. Colloq. Aut. Lang. Prog., volume 1443 of Lect. Notes in Comp. Sci*, pages 1–16. Springer-Verlag, 1998.

[67] Ekkart Kindler, Axel Martens, and Wolfgang Reisig. Inter-operability of workflow applications: Local criteria for global soundness, 2000.

[68] Kais Klai and Jörg Desel. Checking soundness of business processes compositionally using symbolic observation graphs. In *FMOODS/FORTE*, volume 7273 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2012.

[69] Kais Klai and Hanen Ochi. Checking compatibility of web services using sogs. In *2012 IEEE 19th International Conference on Web Services (ICWS), Honolulu, HI, USA, June 24-29, 2012*, pages 670–671, 2012.

[70] Kais Klai and Hanen Ochi. Modular verification of inter-enterprise business processes. In *eKNOW*, pages 155–161, 2012.

[71] Kais Klai and Hanen Ochi. Checking compatibility of web services behaviorally. In *FSEN*, pages 267–282, 2013.

[72] Kais Klai and Hanen Ochi. A bottom-up approach to check the correctness of interorganisational workflows. In *To appear in TASE*, 2015.

[73] Kais Klai and Hanen Ochi. LTL model cheking of service-based business processes in the cloud. In *39th Annual Computer Software and Applications Conference, COMPSAC Workshops 2015, Taichung, Taiwan, July 1-5, 2015*, pages 398–403, 2015.

[74] Kais Klai, Hanen Ochi, and Samir Tata. Formal abstraction and compatibility checking of web services. In *ICWS*, pages 163–170, 2013.

[75] Kais Klai and Denis Poitrenaud. MC-SOG: An LTL model checker based on symbolic observation graphs. In *Petri Nets*, 2008.

[76] Kais Klai, Samir Tata, and Jörg Desel. Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes. In *BPM'09 of LNCS*, pages 294–309, 2009.

[77] Kais Klai, Samir Tata, and Jörg Desel. Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes. *Data Knowl. Eng.*, 70(5):467–482, 2011.

[78] Kais Klai, Samir Tata, and Hanen Ochi. Generic and specific compatibility criteria for web service composition: Formal abstraction and modular verification approach1. *Int. J. Web Service Res.*, 9(4):45–68, 2012.

[79] Alfons Laarman, Mads Chr. Olesen, Andreas Engelbredt Dalsgaard, Kim Guldstrand Larsen, and Jaco van de Pol. Multi-core emptiness checking of timed büchi automata using inclusion abstraction. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 968–983, Berlin, Heidelberg, 2013. Springer-Verlag.

[80] Florian Lautenbacher and Bernhard Bauer. A survey on workflow annotation & composition approaches. In Martin Hepp, Knut Hinkelmann, Dimitris Karagiannis, Rdiger Klein, and Nenad Stojanovic, editors, *SBPM*, volume 251 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[81] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. Analyzing interacting bpel processes. In *BPM'06, LNCS*, pages 17–32. Springer-Verlag, 2006.

[82] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. Analyzing interacting WS-BPEL processes using flexible model generation. *Data Knowl. Eng.*, 64(1):38–54, 2008.

[83] Shuailiang Ma, Li Zhang, and Jimei He. Towards formalization and verification of unified business process model based on pi calculus. In *Software Engineering*

Research, Management and Applications, 2008. SERA '08. Sixth International Conference on, pages 93–101, Aug 2008.

[84] Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil M. P. Van Der Aalst. Monitoring business constraints with linear temporal logic: an approach based on colored automata. In *BPM'11*, pages 132–147. Springer-Verlag.

[85] Fabrizio Maria Maggi, Michael Westergaard, Marco Montali, and Wil M. P. van der Aalst. Runtime verification of ltl-based declarative process models. In *RV'11*, pages 131–146. Springer-Verlag, 2012.

[86] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[87] A. Martens. Usability of web services. In *ICWISEW*, pages 182–190. IEEE Computer Society, 2003.

[88] Axel Martens. On compatibility of web services. *Petri Net Newsletter, Special Interest Groups on Petri Nets and Related Systems Models, Gesellschaft fur Informatik e.V.*, 65, 2003.

[89] Axel Martens. Analyzing web service based business processes. In *Proceedings of the 8th International Conference, Held As Part of the Joint European Conference on Theory and Practice of Software Conference on Fundamental Approaches to Software Engineering*, FASE'05, pages 19–33, Berlin, Heidelberg, 2005. Springer-Verlag.

[90] Axel Martens. Simulation and equivalence between bpel process models. In *In Proc. of Intl. Conference DASD'05*, 2005.

[91] Axel Martens, S. Moser, A. Gerhardt, and K. Funk. Analyzing compatibility of bpel processes. In *AICT/ICIW*, page 147. IEEE Computer Society, 2006.

[92] Axel Martens and Simon Moser. Diagnosing SCA components using wombat. In *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, pages 378–388, 2006.

[93] Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An operating guideline approach to the soa. *Annals Of Mathematics, Computing and Teleinformatics*, 1:35–43, 2005.

[94] Philip M. Merlin and David J. Farber. Recoverability of modular systems. *Operating Systems Review*, 9(3):51–56, 1975.

[95] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[96] OMG. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1, August 2011.

[97] Object Management Group (OMG). Business process model and notation (bpmn) version 2.0. Standard, Object Management Group (OMG), jan 2011.

[98] Victor Pankratius and Wolffried Stucky. A formal foundation for workflow composition, workflow view definition, and workflow normalization based on petri nets. In *Conceptual Modelling 2005, Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), Newcastle, NSW, Australia, January/February 2005*, pages 79–88, 2005.

[99] Lawrence C. Paulson. *Isabelle : a generic theorem prover*. Lecture notes in computer science. Springer-Verlag, Berlin, New York, 1994.

[100] Cesare Pautasso and Gustavo Alonso. The jopera visual composition language. *J. Vis. Lang. Comput.*, 16(1-2):119–152, February 2005.

[101] M. Pesic. Decserflow: Towards a truly declarative service flow language. In *International Conference on Web Services and Formal Methods, volume 4184 of Lecture Notes in Computer Science*. Springer-Verlag, 2006.

[102] M. Pesic, M. H. Schonenberg, and N. Sidorova. Constraint-based workflow models: Change made easy. In *In CoopIS*, 2007.

[103] M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops*, BPM'06, pages 169–180, Berlin, Heidelberg, 2006. Springer-Verlag.

[104] Maja Pesic, Helen M. Schonenberg, and Wil Van Der Aalst. Declare demo: A constraint-based workflow management system.

[105] C. A. Petri. Concepts of net theory. In *MFCS'73*. Mathematical Institute of the Slovak Academy of Sciences, 1973.

[106] Antti Puhakka and Antti Valmari. Weakest-congruence results for livelock-preserving equivalences. In JosC.M. Baeten and Sjouke Mauw, editors, *CONCUR'99 Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 510–524. Springer Berlin Heidelberg, 1999.

[107] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, UK, 1982. Springer-Verlag.

[108] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. Technical report, Cambridge, MA, USA, 1974.

[109] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Proceedings of the First International Conference on Semantic Web Services and Web Process Composition*, SWSWPC'04, pages 43–54, Berlin, Heidelberg, 2005. Springer-Verlag.

[110] Ronald Read and Derek Corneil. The graph isomorphism disease. In *Graph Theory*, pages 339–363, 1977.

[111] Andreas Rogge-Solti, Ronny Mans, Wil M. P. van der Aalst, and Mathias Weske. Repairing event logs using timed process models. In *On the Move to Meaningful Internet Systems: OTM 2013 Workshops - Confederated International Workshops: OTM Academy, OTM Industry Case Studies Program, ACM, EI2N, ISDE, META4eS, ORM, SeDeS, SINCOM, SMS, and SOMOCO 2013, Graz, Austria, September 9 - 13, 2013, Proceedings*, pages 705–708, 2013.

[112] August-Wilhelm W. Scheer. *Aris-Business Process Frameworks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2nd edition, 1998.

[113] Karsten Schmidt. Lola: a low level analyser. In *Proceedings of the 21st international conference on Application and theory of petri nets*, ICATPN'00, pages 465–474. Springer-Verlag, 2000.

[114] Dennis M. M. Schunselaar, Eric Verbeek, Wil M. P. van der Aalst, and Hajo A. Reijers. A framework for efficiently deciding language inclusion for sound unlabelled wf-nets. In *Joint Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'13) and the International Workshop on Modeling and Business Environments (ModBE'13), Milano, Italy, June 24 - 25, 2013*, pages 135–154, 2013.

[115] Roberto Sebastiani, Stefano Tonetta, and Moshe Y. Vardi. Symbolic systems, explicit properties: on hybrid approches for ltl symbolic model checking. In *In Proc. of CAV'05, volume 3576 of LNCS*, pages 350–363. Springer, 2005.

[116] Juliane Siegeris and Armin Zimmermann. Workflow model compositions preserving relaxed soundness. In *Proceedings of the 4th International Conference on Business*

Process Management, BPM'06, pages 177–192, Berlin, Heidelberg, 2006. Springer-Verlag.

[117] Robert Singer. Business process management in small- and medium-sized enterprises: An empirical study. In *Proceedings of the 7th International Conference on Subject-Oriented Business Process Management*, S-BPM ONE '15, pages 9:1–9:8, New York, NY, USA, 2015. ACM.

[118] Fabio Somenzi, Kavita Ravi, and Roderick Bloem. Analysis of symbolic scc hull algorithms. In *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design*, FMCAD '02, pages 88–105, London, UK, UK, 2002. Springer-Verlag.

[119] William Tepfenhart, Jiacun Wang, Daniela Rosca, and Anni Tsai. Resource-constrained and decision support workflow modeling. *International Journal of Intelligent Control And Systems*, 12(1):15–23, 2007.

[120] Antti Valmari. A stubborn attack on state explosion. In *Proceedings of the 2Nd International Workshop on Computer Aided Verification*, CAV '90, pages 156–165, London, UK, UK, 1991. Springer-Verlag.

[121] W. M. P. van der Aalst. Woflan: a petri-net-based workflow analyzer. *Syst. Anal. Model. Simul.*, 35(3), May 1999.

[122] Wil van der Aalst. Loosely coupled interorganizational workflows: Modeling and analyzing workflows crossing organizational boundaries. *Inf. Manage.*, 37(2):67–75, March 2000.

[123] Wil van der Aalst, Kees van Hee, Arthur ter Hofstede, Natalia Sidorova, H.M.W. Verbeek, Marc Voorhoeve, and Moe Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing: applicable formal methods*, 23(3), 2010.

[124] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[125] Wil M. P. van der Aalst. Making work flow: On the application of petri nets to business process management. In *Applications and Theory of Petri Nets 2002, 23rd International Conference, ICATPN 2002, Adelaide, Australia, June 24-30, 2002, Proceedings*, pages 1–22, 2002.

[126] Wil M. P. van der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106, 2010.

[127] Wil M. P. van der Aalst, Christian Stahl, and Michael Westergaard. Strategies for modeling complex processes using colored petri nets. *T. Petri Nets and Other Models of Concurrency*, 7:6–55, 2013.

[128] Wil M. P. van der Aalst, Kees M. van Hee, Arthur H. M. ter Hofstede, Natalia Sidorova, H. M. W. Verbeek, Marc Voorhoeve, and Moe Thandar Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Asp. Comput.*, 23(3):333–363, 2011.

[129] Wil M. P. van der Aalst, Kees M. van Hee, and Robert A. van der Toorn. Component-based software architectures: a framework based on inheritance of behavior. *Sci. Comput. Program.*, 42(2-3):129–171, 2002.

[130] WilM.P. van der Aalst, KeesM. van Hee, ArthurH.M. ter Hofstede, Natalia Sidorova, H.M.W. Verbeek, Marc Voorhoeve, and MoeT. Wynn. Soundness of workflow nets with reset arcs. In *Transactions on Petri Nets and Other Models of Concurrency III*, volume 5800 of *Lecture Notes in Computer Science*, pages 50–70. Springer Berlin Heidelberg, 2009.

[131] WilM.P. van der Aalst and Mathias Weske. The p2p approach to interorganizational workflows. In KlausR. Dittrich, Andreas Geppert, and MoiraC. Norrie, editors, *Advanced Information Systems Engineering*, volume 2068 of *Lecture Notes in Computer Science*, pages 140–156. Springer Berlin Heidelberg, 2001.

[132] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The prom framework: A new era in process mining tool support. In *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets*, ICATPN'05, pages 444–454, Berlin, Heidelberg, 2005. Springer-Verlag.

[133] Kees M. van Hee, Natalia Sidorova, and Marc Voorhoeve. Resource-constrained workflow nets. *Fundam. Inform.*, 71(2-3):243–257, 2006.

[134] Wil vanderAalst and Kees vanHee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2004.

[135] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995.

[136] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344, 1986.

[137] H. M. W. Verbeek, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. Verifying workflows with cancellation regions and or-joins: An approach based on relaxed soundness and invariants. *Comput. J.*, 50(3):294–314, 2007.

[138] H. M. W. Verbeek, Moe Thandar Wynn, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. Reduction rules for reset/inhibitor nets. *J. Comput. Syst. Sci.*, 76(2):125–143, 2010.

[139] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing workflow processes using woflan. *THE COMPUTER JOURNAL*, 44:2001, 1999.

[140] Moe Thandar Wynn, H. M. W. Verbeek, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Business process verification - finally a reality! *Business Proc. Manag. Journal*, 15(1):74–92, 2009.

[141] Moe Thandar Wynn, H. M. W. (Eric) Verbeek, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Reduction rules for YAWL workflows with cancellation regions and or-joins. *Information & Software Technology*, 51(6):1010–1020, 2009.

[142] PengCheng Xiong, Yushun Fan, and MengChu Zhou. A petri net approach to analysis and composition of web services. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, pages 376–387, 2010.

[143] Jian Yang and Mike P. Papazoglou. Service components for managing the life-cycle of service compositions. *Inf. Syst.*, 29(2):97–125, April 2004.

[144] Dongsong Zhang. Web services composition for process management in e-business. *Journal of Computer Information Systems XLV*, pages 83–91, 2004.