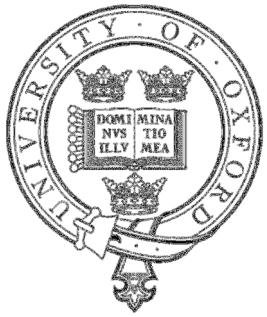
A Framework of Trust in Service Workflows



Wattana Viriyasitavat

Hertford College

University of Oxford

A dissertation is submitted for the degree of

Doctor of Philosophy

Trinity Term 2013

Abstract

The *everything as a service* concept enables dynamic resource provisions to be seen and delivered as services. Their proliferation nowadays leads to the creation of new value-added services composed of several sub-services in a pre-specified manner, known as *service workflows*. The use of service workflow appears in various domains, ranging from the basic interactions found in several e-commerce and several online interactions to the complex ones such as Virtual Organizations, Grids, and Cloud Computing. However, the dynamic nature in open environments makes a workflow constantly changing, to be adaptable to the change of new circumstances. How to determine suitable services has becomes a very important challenge. Requirements from both workflow owners and service providers play a significant role in the process of service acquisition, composition, and interoperations. From the workflow owner viewpoint, requirements can specify properties of services to be acquired for tasks in a workflow. On the other hand, requirements from service providers affect trust-based decision in workflow participation.

The lack of formal languages to specify these requirements poses difficulties in the success of service collaborations in a workflow. It impedes: (1) workflow scalability that tends to be limited within a certain set of trusted domains; (2) dynamicity when each service acts in an autonomous and unpredictable manner where any change might affect existing requirements; and (3) inconsistency in dealing with the disparate representations of requirements, causing high overhead for compliance checking.

This thesis focuses on developing a framework to overcome, or at least alleviate, these problems. It situates in inter-disciplinary areas including logics, workflow modelling, specification languages, trust management, decision support system, and compliance checking. Two core elements are proposed: (1) a formal logic-based requirement

specification language, namely Trust Specification (TS), such that the requirements can be formally and uniformly expressed; and (2) compliance checking algorithms to automatically check for the compliance of requirements in service workflows. It is worth noting that this thesis contains some proofs of logic extension, workflow modelling, specification language, and compliance checking algorithms. These might raise a concern to people focusing deep on one particular area such as logics, or workflow modelling who might overlook the essence of the work, for example (1) the application of a formal specification language to the exclusive characteristics of service workflows, and (2) bridging the gap of the high level languages such as trust management down to the lower logic-based ones.

The first contribution of the framework is to allow requirements to be independently and consistently expressed by each party where the workflow participation decision and acquisition are subject to the compliance of requirements. To increase scalability in large-scale interoperations, the second contribution centres on automatic compliance checking where TS language and compliance checking algorithms are two key components. The last contribution focuses on dynamicity. The framework allows each party to modify existing requirements and the compliance checking would be automatically activated to check for further compliance. As a result, it is anticipated that the solution will encourage the proliferation of service provisions and consumption over the Internet.

Acknowledgement

The last four years have been some of the most challenging, yet exciting of my academic research. I have learnt so much, developed many skills and met some fantastic people along the way. Without the help and guidance of the following people, this thesis could not have been possible.

First of all, I would like wholeheartedly to thank my supervisor, Dr. Andrew Martin, for the continued support, direction, and encouragement he has provided. Secondly, I would like to thank my parents who provide me this great opportunity to pursue a D.Phil. They have continually provided this support of my studies through this period. Thank to my sisters Nok and Lek for all the helps and supports. I would also like to thank Professor Li Da Xu for the opportunities and advice for publications. I have enjoyed collaborating with him on several papers and I am very grateful for the ideas he has provided.

I have been lucky to meet several great people in Oxford. My thanks need to be paid for John, Lyle, Cornelius Namiluko, Andy Cooper, Shamal Faily, Ronald Kainda, Joe Loughry, Ivan Flechais, Michael Goldsmith, and the rest of the people in Security Reading Group for aiding, advising and ultimately resolving several issues. I hope we will meet and perhaps collaborate in some works together in the near future.

This D.Phil is funded by Chlualongkorn University for tuition fee and living expense. Thank you for all supports to make this thesis possible.

Finally, I must thank my lovely girlfriend, Pleum, for all she has done for me.

Thank you very much to every one again.

Table of Content

1 Introduction **1**

1.1 Overview 1
1.2 Motivation 2
1.3 A Brief Overview 3
1.4 Thesis Statement 5
1.5 Methodology 6
1.6 Novel Contributions and Publications 7
1.7 Thesis Structure 9

2 Background **11**

2.1 Chapter Introduction 11
2.2 Defining Service Workflow Trust 12
2.3 Relationships of Trust Properties and Workflow Characteristics . . 14
2.4 Literature Reviews 18
2.4.1 Trust in Grid Contexts 19
2.4.1.1 Policy- and Credential-based Trust in Grids . . . 19
2.4.1.2 Trust and Resource Management in Grids 20
2.4.2 Trust in VOs 21
2.4.3 TC in Service Workflows 22
2.4.4 Formal Trust Models in Service Workflow 23
2.4.5 Trust in Service-Oriented Workflow 23
2.4.6 Trust in Provenance-based Service Workflows 25
2.4.7 Other Aspects 26
2.4.7.1 Trust and Access Control in Service Workflows 26

2.4.7.2 Reputation- and Behavioral-based Trust 27
2.4.7.3 Service Delegation and Trust Transitivity 27
2.4.7.4 Trust in Service Workflow Structure 28
2.5 Chapter Discussion 29

3 Trust Specification Language in Service Workflows 31
3.1 Chapter Introduction 31
3.2 Introduction to Specification Language in Service Workflows 34
3.3 Criteria for Specification Languages in Service Workflows 35
3.4 Related Works 37
3.5 Service Workflow Net (SWN) 41
3.5.1 Why Petri Net? 42
3.5.2 Reduction Rules and Binding Priorities 43
3.6 Requirement Specification Formalisation for SWN 45
3.6.1 Vector TMC 47
3.6.2 Trust as a Bounded Value [-1..1] 47
3.6.3 TS Algebra 48
3.6.4 Satisfiability Relations 57
3.7 The Application Example 58
3.7.1 Travel Agent Workflow Scenario 58
3.7.2 Applying TS Formulas 59
3.8 The Example of Calculation Method 62
3.9 Basic Analysis of TS Formulas 63
3.10 Chapter Discussion 64

4 Syntax and Semantics of TS Language 67
4.1 Chapter Introduction 67
4.2 Background 68
4.3 Syntax of TS Language 69
4.4 Semantics of TS Language 74
4.4.1 Composite Formulas 74
4.4.2 Path Formulas 78

4.4.3 Direction Formulas 79
4.5 Algebraic Properties, Soundness, and Completeness 79
4.6 Expressiveness of TS Language in Workflow Systems 83
4.7 Important Semantics Equivalences of TS Formulas. 85
4.7.1 Direction Formulas 85
4.7.2 Path Formulas 85
4.7.3 Composite Formulas 85
4.7.4 Formulas Transformation 86
4.8 Practical Patterns of TS Language 87
4.9 Subsets of SWN and Partially Satisfied 88
4.10 Chapter Discussion 88

5 Algorithms for Compliance Checking 91

5.1 Chapter Introduction 91
5.2 Background 94
5.2.1 Model Checking and Specification Languages 94
5.2.2 Reasoning Algorithms 95
5.3 Part I: Composite Formulas, Boolean Representations, Compliance Checking Algorithms, and Comparisons 96
5.3.1 Pre-processing TS Formulas 96
5.3.2 CTT 98
5.3.3 EDNF 100
5.3.4 RCOBDDs 102
5.3.4.1 COBDDs Definitions and Theorems 102
5.3.4.2 Algorithms for RCOBDDs 106
5.3.4.2.1 1st Class Algorithms 106
5.3.4.2.2 2nd Class Algorithms 111
5.3.4.3 Analysis of Algorithms for RCOBDDs 114
5.3.5 Comparison of CTT, EDNF, and RCOBDD Algorithms . . 116
5.4 Part II: Compliance Checking for Path Formulas 116
5.4.1 A Quick Glance on Simplifying SWNs 117

5.4.2 Labelling Algorithm with Constrained Tasks 119
5.4.3 Analysis of Labelling Algorithm 125
5.5 Part III: Compliance Checking for Direction Formulas 125
5.6 Handling the Results between Satisfied and Partially Satisfied 126
5.7 Chapter Discussion 127

6 A Scenario 131
6.1 Chapter Introduction 131
6.2 Scenario Overview 131
6.3 Task Necessity, Dependencies, and Specification 135
6.4 Service Composition, Separation, and Specification 136
6.4.1 Service Composition 137
6.4.2 Service Separation 139
6.4.3 Requirement Specifications 142
6.5 Characteristics Elicitation of Travel Agent Workflow 150
6.6 Expressiveness of TS Formulas 151
6.7 Chapter Discussion 152

7 Prototype Implementation and Evaluation 153
7.1 Chapter Introduction 153
7.2 Prototype Implementation 154
7.2.1 Usage Scenario 154
7.2.2 System Implementation 155
7.3 Evaluation and Analysis 157
7.4 Chapter Discussion 165

8 Conclusion and Discussion 167
8.1 Conclusion 167
8.2 Discussion 170
8.2.1 Potential Threats and Vulnerabilities 170
8.2.2 Petri Net as a Workflow Language 171
8.2.3 Conflict Discovery Enhancement 175

8.2.4 Performance Optimisation 176
8.2.5 Scalability, Usability, and Extendability 176
8.2.6 Service Explosion Problem 178
8.2.7 Expressiveness and Expensiveness 178

Bibliography **181**

Appendix A: Theorem Proofs for TS Properties **195**

Appendix B: TS Semantics for Partially Satisfied **201**

Appendix C: Taylor's Approximation **205**

Appendix D: Theorem, Proofs, and Algorithms **207**

Appendix E: Travel Agent Workflow Description **213**

Appendix F: Programme Codes **237**

Acronyms

ADL	Architecture Description Languages
ATS	Association between Tasks and Services
BDDs	Binary Decision Diagrams
BNF	Backus–Naur Form
BPT	Backward Path Trust
CNF	Conjunctive Normal Form
COBDDs	Constrained Ordered Binary Decision Diagrams
CTL	Computational Tree Logic
CTR	Concurrent Transaction Logic
CTT	Constraint Truth Table
DAG	Direct Acyclic Graph
EaaS	Everything as a Service
EDNF	Exclusive Disjunctive Normal Form
EoS	Enforcement of Sequences
ERP	Enterprise Resource Planning
ET	Existence Trust
FDR	Flexible Degrees of Restriction
FIDIS	Future of Identity in the Information Society
FOL	First Order Logic

FSM	Finite State Machine
FTS	Flow- and Task-oriented Specification
GSI	Grid Security Infrastructure
GTS	Grid Trust Service
HPT	Henceforth Path Trust
IAS	Integrated Authentication Service
ILR	Interoperability with Local Requirements
LTL	Linear Time Logic
NuSMV	New Symbolic Model Verifier
OBDDs	Ordered Binary Decision Diagrams
P2P	Peer-to-Peer
PDF	Probability Density Function
QoS	Quality of Service
RBAC	Role-based Access Control
RCOBDDs	Reduced Constrained Ordered Binary Decision Diagrams
SOA	Service Oriented Architecture
SoD	Separation of Duty
SRN	Service Resource Net
SSL	Secure Socket Layer
SWN	Service Workflow Net
TC	Trust Computing
TMC	Trust Model Computation
TpC	Temporal Constraints
TPM	Trusted Platform Module

TRMS	Trust-aware Resource Management System
TS	Trust Specification
TVD	Trusted Virtual Domains
VO	Virtual Organisation
WFMS	Workflow Management System
WSTO	Web Service Trust Ontology

List of Figures

Figure 2.1: Inter-service and Inter-domain Trust Relationship 16

Figure 3.1: Three Ways of Service Workflow in Practice 32

Figure 3.2: Workflow Example with HPT, BPT and ET 45

Figure 3.3: Examples of (a) Service-join and (b) Service-split 51

Figure 3.4: Graphical Illustration Examples of Algebraic Operators 57

Figure 3.5: The Travel Agent Workflow 60

Figure 4.1: Graphical Illustration of Service-join and Service-split Types 74

Figure 4.2: Examples of Non-equivalence Formulas 86

Figure 5.1: The Tree Structure of $\mathcal{BP}\mathrm{A}_{\mathrm{t}}(\mathrm{q}_1 \sqcup (\mathrm{q}_2 \boxplus (\mathrm{q}_3 \sqcap \mathrm{q}_4))$ 97

Figure 5.2: Pseudo Code for CTTSat Algorithm 99

Figure 5.3: Graphical Representation of Efficiency Complexity Calculation 100

Figure 5.4: Pseudo Code for Algorithm EDNFSat 101

Figure 5.5: Examples of COBDD Trees 104

Figure 5.6: Some Examples of Reduced COBDD Trees 106

Figure 5.7: Pseudo Code for Algorithm ForSomeSat1 107

Figure 5.8: Pseudo Code for Algorithm ForAllSat1 108

Figure 5.9: Pseudo Code for Algorithm reachable 109

Figure 5.10: An Example Execution for Algorithm Reachable 111

Figure 5.11: Pseudo Code for Algorithm ForSomeSat2 111

Figure 5.12: Pseudo Code for Algorithm ForAllSat2 112

Figure 5.13: Pseudo Code for Algorithm LogicalCheck 113

Figure 5.14: Pseudo Code for Algorithm EtractPathCheck 113

Figure 5.15: A High Level of How Algorithms for Composite Formula Run 114

Figure 5.16: An Example of Simplifying SWN 117

Figure 5.17: The Example of the Labelled SWN 118

Figure 5.18: The Example of Labelling States of $\forall_{t_1} \Diamond S_1 \wedge \exists_{t_2} \Diamond S_2$ 120

Figure 5.19: The Iteration Step for Labelling States with Formula $\exists_{t_3} \odot \phi_1$ 121

Figure 5.20: The Iteration Step for Labelling States with Formula $\exists_{t_3} [\phi_1 \Cup \phi_2]$ 122

Figure 5.21: The Iteration Step for Labelling States with Formula $\forall_{t_6} \Diamond \phi_1$ 122

Figure 5.22: Pseudo Code for TSSat that Calls $Sat_{\exists\odot}$, $Sat_{\forall\Diamond}$, and $Sat_{\exists\Cup}$ 123

Figure 5.23: Pseudo Code for SATEX 124

Figure 5.24: Pseudo Code for SATEU 124

Figure 5.25: Pseudo Code for SATAF 124

Figure 5.26: Pseudo Code for $pre_{\exists}$ 125

Figure 5.27: Pseudo Code for $pre_{\forall}$ 125

Figure 5.28: Pseudo Code for the DirectionSat Algorithm 126

Figure 5.29: Running Sequences of Unsatisfied, Partially Satisfied, and Satisfied 127

Figure 6.1: Task Coordination of the Travel Agent Workflow 134

Figure 6.2: Task Coordination with Acquired Services 137

Figure 6.3: RCOBDD Form of Services for FlightSearching 138

Figure 6.4: The RCOBDDs Form of $Start.f = (\psi, split)$ 141

Figure 7.1: System Architecture 154

Figure 7.2: The Flowchart of Activation of Functions 156

Figure 7.3: Performance of the Twenty-one Requirements in Section 6.3.3 159

Figure 7.4: Comparison of Algorithm RCOBDDSat, EDNFSat, and CTTSat 159

Figure 7.5: Performance Evaluation of 10, 20, 50, 100, and 1,000 Formulas 160

Figure 7.6: Performance Evaluation of 10, 20, 50, and 100 Services Involved 162

Figure 7.7: Comparison of the 1st and 2nd Class Algorithms 162

Figure 7.8: The Comparison of CTTSat, EDNFSat, and RCOBDDSat 164

Figure 8.1: Two Example Snapshots of SWNs 183

Figure B.1: Example of $\mathcal{M}' \subseteq \mathcal{M}$ 206

Figure D.1: Some Examples of Reduced COBDD Trees 212

Figure D.2: Algorithm for COBDD-to-RCOBDD Transformation 214

Figure D.3: Example of COBDD-to-RCOBDD Transformation 213

Figure D.4: Pseudo Code for Algorithm SWN2FSM 215

List of Tables

Table 2.1: Various Definitions of Trust 13

Table 2.2: Relationships between Workflow and Trust Characteristics 15

Table 2.3: A Summary of Trust in Service Workflows 18

Table 3.1: Comparison between Existing Works on Criteria 39

Table 3.2: Constrained Truth Table for ϕ AND ψ 44

Table 3.3: Constrained Truth Table for ϕ XOR ψ 44

Table 3.4: Constrained Truth Table for ϕ OR ψ 44

Table 3.5: The List of Services and Tasks Associated 60

Table 3.6: Examples of TS Formulas of Case 1 to 5. 61

Table 5.1: Constrained Truth Table for A XOR (B OR (C AND D)) 99

Table 5.2: Complexity Comparison of Reachability Query Approaches 107

Table 5.3: Comparing Efficiency of Three Representation and the Algorithms 115

Table 6.1: TS Formulas and Their Results 148

Table 7.1: Confidence Interval of Each Experiment 158

Table E.1: Tasks Necessity, Dependencies, and Specifications 217

Table E.2: Association of Service and Tasks in the Travel Agent Workflow 218

Table E.3: Logical Relations of Acquired Services 219

Table E.4: Services Glossary 222

Chapter 1

Introduction

1.1 Overview

With the power of Internet technologies, several service-based online interactions have become common over the past few years [1]. At present, they appear in various domains, such as within an organisation where services are used as a building block to streamline business processes and its concept is realised in many Enterprise Resource Planning (ERP) solutions; in e-commerce where the interaction simply occurs between service providers and customers; and in decentralised collaborative environments such as Peer-to-Peer (P2P), Grids, Virtual Organizations (VOs), and Cloud Computing [2], where services have become a fundamental element of collaborations. A number of Internet-based technologies and standards such as Service-Oriented Architecture (SOA) and Web Services are key factors that enable several service-oriented applications to provide more than simple file sharing or resource access, but rather ubiquitous service provisions. Nowadays, the economies in industrialised sector across the world have become more service oriented. The workforce employed in the service sector continued to grow during the second half of the 1980s and the 1990s. At the beginning of 2000, in Organisations for Economic Co-operation and Development (OECD) countries, about three quarters of employees were working in the service sector [2]. Nowadays, the world's biggest source of employment is in the service sector, rather than the agricultural and industrial sectors. It is predicted that in

future the service sector would provide more than half of global employment opportunities. Since the proliferation of service provisions enables services with complement functionalities to be composed in a pre-specified manner, known as service workflows, several challenges arise in this context, turning the attention of researchers into this area.

1.2 Motivation

Internet services are increasingly being used in the form of service workflows. Currently, one of the most important challenges is how to determine suitable services to be part of a workflow, especially when services are dynamically composed to perform sophisticated tasks. Since a service workflow consists of many tasks where each task can be executed by one, or several, services, this leads to more complicated service composition than the previous traditional composition approaches. Workflow Management Systems (WFMSs) have been introduced to manage this complexity, offering service composition, service execution, and workflow interoperation facilities. Currently, many WFMSs are extended to support dynamicity by coordinating disparate autonomous services that may be made available at distributed sites [3]. However, the lack of formal specification languages in service composition in the workflow domain poses the major problem involving scalability, dynamicity, and inconsistency of disparate representations of requirements. This makes the process of requirement verification in workflow composition very complicated, resulting in high overhead, and therefore hindering the success of service workflow collaborations.

There is a need for a high-level approach to specify and manage these requirements, which are general enough to be integrated into applications and used in any platform. So far, there have been neither generic patterns nor standards in specifying requirements that fit into every, or at least the general, context of service workflows [4]. This poses a major difficulty in solving the aforementioned problems. It impedes workflow scalability that tends to be limited within a trusted domain, fails to address high degree of dynamicity when services act in an unpredictable manner, and results in inconsistency when requirements from each service and a workflow owner are represented in different formats.

The main motivation of this thesis is to solve, or alleviate, these obstacles. A formal requirement specification language that is applicable to the nature of service workflows is presented. With its well-defined syntax and semantics, compliance checking algorithms can be developed to support scalability. Finding a solution would reduce inefficiency in requirement verification and therefore leverages scalability and security overall.

1.3 A Brief Preview

This section details a high-level application example to demonstrate the importance of requirement specification languages in service workflows. Suppose that there is a travel agent business (this workflow will be extended to the full scenario and used as the application example throughout this thesis) providing flight, hotel, and car bookings. Sequences of tasks are coordinated and represented by a workflow. Each task is executed by a service(s). One example of an end-to-end sequence begins with searching, booking, and payment where each task can be executed by a single or multiple services. For instance, a searching task can be done by integral results from several services. Conversely, one competent service might be capable of executing several tasks such as a service responsible for executing both searching and booking. To some extent, choices are available, for example, a payment task can be executed by a service either verified by VISA or MASTER. All of these aspects including service composition and task coordination are abstracted by our service workflow model, namely Service Workflow Net (SWN) which will be presented in Chapter 3.

In this circumstance, services involved in a workflow may have never known each other before. A service X at the booking task may not know what services are responsible for payment. This creates vulnerability as its booking information can be transferred to unwanted services such as the services from its business competitors. To avoid this situation, service X imposes a requirement that all bookings made will eventually go to payment and the services for payment is trusted if they do not belong to its competitors. Traditionally at the time when this report is written, service X can specify this requirement to the workflow owner who will guarantee for valid execution which in this case the

booking information will not be transferred to the service X's competitors. This becomes impractical when there is a large number of services and requirements involved in a workflow that creates high overhead for the owner to check validity for every execution. Consequently, too much reliance is put on the workflow owner and security can be assured only if he is prudent and benign.

The first part of this thesis (Chapters 3 and 4) is the introduction of the Trust Specification (TS) language with its syntax and semantics to formally express such requirements. It serves as a fundamental element for automatic compliance checking which will be detailed in the second part (Chapters 5). The language allows each party to uniformly express its own requirements. Applied to the above example, the requirement of service X can be expressed in a formal way, thus the trust of a payment service can be evaluated from the result of compliance.

It is important to note that the requirements are used to support trust-based decision making, instead of computing a trust value of a service, or a workflow. For instance, service X trusts the services for payment if they are certified by VISA, and service X will join the workflow if this requirement is satisfied. In some cases, relative properties, which can be described as the properties of integral services, may also have an influence on the trust-based participation decision. For example, service X additionally requires that the booking information must be encrypted by Secure Socket Layer (SSL) along the trace, which implies that the intermediate services along the path until reaching the payment services must operate on SSL technology.

Here, some other examples of requirements in service workflows that can be captured by TS language are given (later, the below examples will be encoded by TS formulas in Chapter 4, Section 4.8.)

- A private Weather Forecast Company only trusts services that belong to BBC before providing weather information.
- In order to share information in a P2P network, one node requires that the subsequent nodes to which information will be transferred must possess a reputation score not less than 0.8 (out of 1).

- In e-commerce, a booking service requires that once the booking is made all paths must lead to the payment service eventually.
- In addition to the above example, a payment service requires that at least one booking must be made before executing payment.
- A booking service requires that all subsequent services must operate on SSL technology until the payment is done.
- In e-research, all subsequent services are trusted if they belong to Microsoft. One example is that the University of Oxford's Computer Laboratory signs an agreement to provide research results only to Microsoft.

1.4 Thesis Statement

The thesis statement is as follows:

> *"Considering the prevalence of Internet services, where a service workflow can be dynamically composed of several services, trust-based service participation decision and acquisition requirements are important. The formal requirement specification for trust and automatic compliance checking are used to stimulate the success of service workflows in large-scale open environments."*

Several aspects can be elicited from this statement. Firstly, it points out the dynamic characteristics of service workflows where changes can be made regularly [5]. Since the nature of dynamic workflow interoperations allows participating services to freely enter, upgrade, degrade, or terminate, in an unpredictable manner, it results in constant changes in service workflow and requirements. This causes a significant impact on the success of interoperations. Secondly, coordinating disparate services, which may be available at distributed sites, poses a significant challenge to requirement verification [3]. Thirdly, requirements in this context are tightly adhered to the unique characteristics of service workflows. Finally, since compliance of requirements is a key driver for decision making in service acquisition and workflow participation, this implies a consequent need of automatic compliance checking. To this end, successful service interoperations in service workflows

require the amalgamation of formal and system approaches. TS language is a formal way to consistently express requirements. Based on its formal syntax and semantics, the system that facilitates compliance checking processes is developed.

1.5 Methodology

The purpose of this thesis is the development of a framework for trust-based decision making based on compliance checking of requirements. The solution is aimed to solve the inherent interoperation problems, including scalability, dynamicity, and inconsistency. The work contains two major parts described below.

1. TS language and its syntax and semantics are developed such that requirements can be formally and uniformly expressed. It allows requirements to be independently and consistently defined by autonomous services, enabling any service to make a trust-based decision in workflow participation. The syntax and semantics are two fundamental elements for reasoning about service workflows.
2. The algorithms for compliance checking between TS formulas and a service workflow are developed. They are used to automate labour-intensive and time-consuming manual checking that could negate scalability and efficiency in large-scale dynamic interoperations.

The Petri Net semantics is extended for our service workflow modelling as its expressive power in both graphical and mathematical representations facilitates both designing process and applying formal method into the workflow semantics (justifications are provided in Chapter 3, Section 3.6.1, and Chapter 8, Section 8.2.1). Although the right choice of workflow modelling languages is an open on-going argument, Petri Net itself is sufficient in a way that our TS language can be reasoned. Furthermore, other modelling languages including pi calculus, UML activity diagrams, and other graph-based languages can be semantically translated into Petri Net [6]. Developing a specification language based on Petri Net would yield a good benefit as it is able to be extended to the broader areas.

TS language employs and extends the existing First Order Logic (FOL) and Computational Tree Logic (CTL) to capture the unique characteristics of requirement

specification in service workflow (see Chapter 2, Section 2.3). We use the semantic extension from FOL to address service composition specification and to enhance cardinal constraint on task over CTL to make it specific to the nature of service workflows that contain braches and sequence of tasks. Developing a language based on formal logics yield several advantages. One obvious advantage is that it provides strong backgrounds with plenty of definitions, theorems, and proofs. It also bridges the gap of the high-level specification, usually found in the languages with newly invented predicate terms such as those in trust management, to the low-level logic-based specification.

For the proof of concept, the prototype is implemented to illustrate the practicality in terms of applicability and scalability. Applicability is illustrated by the translation of narrative statements into TS formulas and scalability is evaluated by the time performance of the compliance checking algorithms from the small to large number of TS formulas and services. Although our work is evaluated in the set-up scenario, it to some extent provides the identification of the possibility that it can be deployed in real-world environments.

To this end, the solution will leverage interoperability by increasing the degree of automation that directly affects scalability and security, while still preserving autonomy of the involved services in a workflow. As a result, it will encourage the proliferation of online service provisions, consumption, and interoperation over the Internet.

1.6 Novel Contributions and Publications

This section presents a number of contributions of the framework.

1. The framework with the formal specification language is a means to allow requirements to be independently and consistently expressed by each party where the workflow participation decision and acquisition are subject to the compliance of requirements.
2. With its well-defined syntax and semantics, the language serves as a key driver for reasoning between requirements in the form of TS formulas and service workflows.
3. Since requirements are uniformly expressed, it has no need to devise distinct complicated algorithms to deal with unrelated formats. Therefore, it supports

automatic compliance checking which leverage scalability, and simultaneously increase efficiency of security management in large-scale interoperation.

4. The forth contribution focuses on dynamicity. The framework allows each party to modify existing requirements and the compliance checking would be automatically activated to check for further compliance. As a result, it is anticipated that the solution will encourage the proliferation of service provisions and consumption over the Internet.
5. Finally, this language equips workflow owners with a tool in the process of service acquisition and composition, which regulate how a workflow is constructed in accordance with their requirements.

There are a few publications involving some conference papers and journals presented in the following list.

1. W. Viriyasitavat, and A. Martin, Formal Trust Specification in Service Workflows, In Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on, pp. 703-710, 2010. (From Chapter 3)
2. W. Viriyasitavat, and A. Martin, Formalizing Trust Requirements and Specification in Service Workflow Environment, In Runtong Zhang, Jos Cordeiro, Xuewei Li, Zhenji Zhang and Juliang Zhang, editors, ICEIS, vol. 3, pp. 196–206. SciTePress. 2011. (From Chapter 3, and some parts of Chapter 4, and 5)
3. W. Viriyasitavat, D. X. Li, and A. Martin, SWSpec: The Requirement Specification Language in Service Workflow Environments, In IEEE Transactions on Industrial Informatics, Special Issue in Information Systems with Industrial Applications, vol. 8, no. 3, pp. 631-638, 2012. (From Chapter 3 and 4)
4. W. Viriyasitavat, P. Ruchikachorn, Compliance Checking Algorithms for Service Workflow Interoperations, to be appeared in Decision Support System, Elsvier (accepted). (From Chapter 5)

1.7 Thesis Structure

The remaining part of this thesis is organised as follows. In Chapter 2, the comprehensive backgrounds and related works on trust and service workflows are presented. The relation between trust and service workflows is first synthesised with important workflow characteristics and trust properties. This chapter ends with the review of the existing literature in this area. Chapter 3 introduces TS language as a means to express requirements for trust in service workflows. Three modes of trust and algebraic operators are defined to form TS formulas. A travel agent business workflow is used as an example to illustrate the applicability of these formulas. To provide a strong foundation of the language, syntax and semantics are explained with algebraic properties and proofs in Chapter 4. Chapter 5 focuses on the development of algorithms for compliance checking between requirements expressed by TS formulas and service workflows. In Chapter 6, the full scenario of the travel agent workflow is described, which is evaluated by the prototype implementation in Chapter 7. Finally, we conclude and discuss important issues and future development of the framework in the last chapter.

Chapter 2

Background

2.1 Chapter Introduction

To provide an overview of trust and service workflows, the following sections define and survey existing literature on these subjects. The literature survey is substantial and spreads across several research boundaries. The purpose is to lay the ground for comprehensive understanding, and to serve as justifications for the need of requirement specification languages. This chapter consists of three major parts.

The first part summarises the ideas of the proponents of trust definitions in many disciplines. A wide-ranging of definitions and analysis are provided which to some extent describes the major works on the concept of trust. Typically, each of them involves specialisation in one area which leads to specific discussions confined within such area. Then, the conclusion of trust definition specific to service workflows is drawn.

The second part extends the idea of trust in a way that helps to clarify types of trust situated in service workflows. *Workflow characteristics* and *trust properties* are described, providing those who work with trust with a useful reference for clear discussions in service workflows. TS language to be presented in Chapter 3 is developed in a way to address these *characteristics* and *properties*.

The last part surveys the existing literature on trust specific to service workflows. It gives comprehensive information of what work has been done so far, and deeper justifications of the characteristics and properties for the consideration presented in the second part. From this survey, it can be concluded that a formal requirement specification language in service workflows is lacking. Investigating this area would yield a great contribution.

2.2 Defining Service Workflow Trust

In most of current researches, how trust is defined falls into various categories and a solid generic definition is quite elusive [7]. Trust has received attention in several areas. Each discipline approaches trust with its own perspectives where the definition of trust tends to reflect the view of a particular discipline. For example, while sociologists tend to see trust as relationship in nature ([8] and [9]), some psychologists view it as a personal view/attribute [10]. Social psychologists are more likely to consider trust an interpersonal phenomenon [11], whereas economists are more inclined to view trust as a rational choice mechanism to increase utility [12]. This section summarised some important definitions on trust and then the definition specific to service workflows is drawn (A summary of these definitions is presented in Table 2.1.). For a full detailed survey, trust classifications, and analysis of trust definitions, please refer to [4].

The Oxford English Dictionary's definition of trust includes terms such as *reliability, truthfulness, ability*, or *strength*. Other dictionary definitions relate trust to *confidence*, *belief*, *competence*, and *anticipation*. These terms are common in many usages, and pertain to other terms like *expectation*, *predictability*, and the motivational relevance of someone or something [13]. None of these concepts can accurately describe trust, since it is an abstract concept rather than a concrete number, which combines multiple complicated factors.

From a number of definitions in [4], we can see that trust is used as an integral component in many kinds of agents' interactions, allowing agents to act under uncertainties and with the risk of negative consequences [14]. From the security viewpoint, it can be observed that humans are trusted because they are believed to be honest whereas systems

are trusted because they are believed to be secure [15], such as providing protection to mitigate potential risks.

Table 2.1: Various Definitions of Trust [4]

Category	Definitions	Purpose	Focus and Analysis
Belief	Deutsch [13]	To capture the connotations of everyday usages	These definitions regard trust as a qualitative notion which is difficult to compare trust between two parties for a similar event.
	Kini and Choobineh [16]	Trust of a system in the context of e-commerce	
	Grandison and Sloman [19]	Trust of an entity based on its competence as quality	
Risk and Vulnerabilities	Mayer et al. [12]	Including trust based on risks of a particular action, and co-operation between organisations	Trust related to risks and vulnerabilities is a qualitative notion that is difficult to measure.
	Lau and Whyte [20]	Risk-related trust in the context of VOs	
Subjective Probability	Gambetta [21]	Trust bound to the subjective probability an individual assigns to the occurrence of some set of future events	Trust (or, symmetrically, distrust) is a particular measureable level of the subjective probability with which an agent will perform a particular action for a specified context.
	Mui et al. [22]	Using Statistical method for computing reputation	
	Olmedilla et al. [18]	Trust of a party for a particular service	
	Jøsang et al. ([23] and [24])	Reputation as trustworthiness based on past experience and recommendation	

Despite there being a claim that there is considerable confusion around the terminology in describing trust in multiple meanings ([16] and [17]), the notion of trust is still being used effectively in many contexts. From our perspective, the important definition from Olmedilla et al. [18] refers to the relationship of a service provider and consumer, which is closely related trust in our work. They define trust as

"Trust of a party A to a party B for a service X is the measurable belief of A in that B behaves dependably for a specified period within a specified context (in relation to service X)."

In the area of service workflow where trust has an influence on service acquisition and can be used for decision making in workflow participation, the definition of trust in the context of service workflows is defined in [4].

"Trust in a workflow is a subjective, possibly mutually measurable, relationship between (direct and indirect) interacting services, or between services and a workflow owner, that affects the willingness of collaboration in a given situation within a specific context and time."

Direct interaction takes place between two adjacent services in a workflow path, while the indirect one occurs when they are not immediately connected. The mutual trust relationship describes the bidirectional measurable relationship exhibited between interacting services. Establishing trust in both directions is independent. For example, a preceding service may need to verify the trustworthiness of its subsequent services before passing information, while a subsequent service perhaps requires trust in an outcome that is originated from its trusted preceding services. This definition reflects trust-based decision depending on the context, situation, and temporal notions, as trust is bound to time.

2.3 Relationships of Trust and Workflow Characteristics

Before delving into the literature in this area, it is important to describe workflow characteristics with respect to trust properties (see Table 2.2) [4]. This information enables researchers to be appropriately aware of trust implications in the context of service workflow. From the work in [25], we summarise the characteristics and trust properties in the following paragraphs. The enumerations C# and T# are used for workflow characteristics and trust properties.

Transient (C1) and Persistent (C2) Workflow Characteristics

A workflow is *transient*, *ad hoc*, or *short-lived*, if services are collaborated in a short-term basis. The workflow in short-lived VOs and ad hoc communications in a network [26] exemplify this characteristic. On the other hand, a workflow is considered *persistent* if it involves repeated processes to complete repeated objectives over time. For example, a workflow of salary paying is repeatedly executed every month. Each characteristic requires different types of trust. A *transient* workflow frequently interacts with new services, where trust relationship tends to be *temporary* (T1), whereas a *persistent* workflow usually

incorporates with previously known services, which requires the *longer* trust relationship (T2).

Table 2.2: Relationships between Workflow and Trust Characteristics [25]

Workflow Characteristics	Related Trust Properties and Characteristics
1. Transient (*C1*) vs Persistent (*C2*)	Temporary (*T1*) and Long-term (*T2*) Trust Relationship*
2. Static (*C3*) vs Dynamic (*C4*)	Hard-security-based (*T3*) and Soft-security-based (*T4*) Trust*
3. Workflow Formation (*C5*) and Enactment (*C6*)	Hard-security-based (*T3*) and Soft-security-based (*T4*) Trust*
4. Global (*C7*) vs Local (*C8*) Management	Centralised (*T7*) and Distributed (*T8*) Trust
5. All Types of Service Workflows	Inter-domain (*T5*) and Inter-service (*T6*) Workflow Trust Characteristics Service Delegation (*T9*) and Trust Transitivity (*T10*) Direct (*T11*) and Indirect (*T12*) Trust in Workflow Paths Trust in Singleton (*T13*) and Composite Services (*T14*)

The characteristics marked with * are not provided by dedicated sections, but implicitly discussed along the way in following sections.

Dynamic (C3) and Static (C4) Workflow Characteristics

A *static* workflow is described when there is no change, or a few changes, in its components including the sequences of tasks and services invocation, and services involved. A *dynamic* workflow on the other hand deals with constant changes throughout its lifetime ([27] and [28]). As a result, trust based on *hard* and *soft* securities seem to be suitable in *static* and *dynamic* workflows respectively.

Workflow Formation (C5) and Enactment (C6)

A workflow can be distinguished into *formation* and *enactment* stages. A number of activities during the formation stage are related to workflow planning and designing [29] while the latter involves the interpretation of these elements during execution which is considered more dynamic. Dissimilarity between these two stages leads to the different needs of trust. Since most activities in the *formation* stage are relatively *static*, *hard* security seems to be appropriate. Workflow *enactment* on the other hand encounters more *dynamic* issues, which is more appropriate to *soft* security.

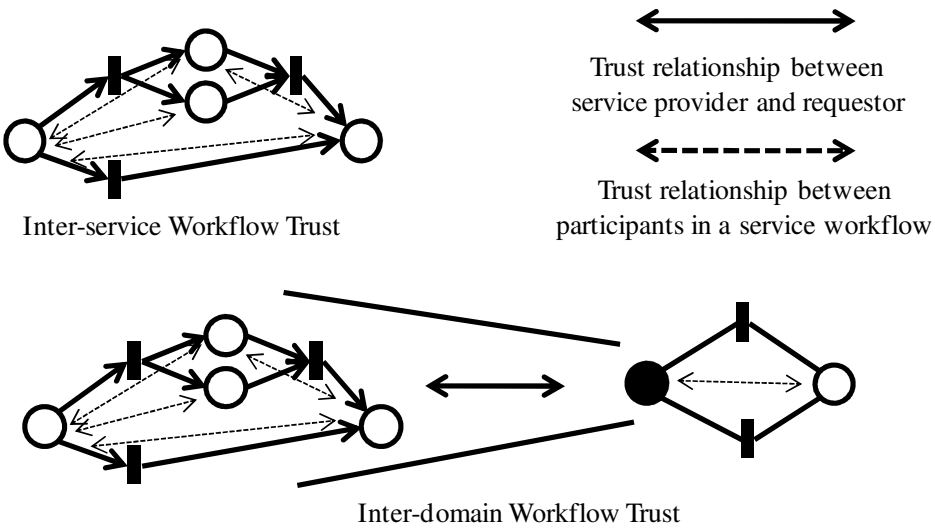


Figure 2.1: Inter-service and Inter-domain Trust Relationship

Global (C7) and Local (C8) Workflow Management

In the *global* workflow management, a centralised engine is responsible for managing services, and enforcing security throughout a workflow [30]. In a more complex situation, however, WFMSs may encounter several services that may be available at distributed sites [31]. This complexity cripples efficiency as well as security enforcement by using the centralised approach. Localising workflow management permits each service to isolate itself from global enforcement. As a result, *global* and *local* characteristics regard trust in different ways. Complying with *global* requirements in the *centralised* approach (T7) results in implicitly being trusted by other services, while, in the *local* management, satisfying local *distributed* (T8) requirements affects trust in workflow participation.

Inter-service (T5) and Inter-domain (T6) Trust Characteristics

Trust situated horizontally among services inside a workflow domain is called *inter-service* trust (see dash lines in Figure 2.1), where the scope of relationship is confined within a single workflow domain. Another perspective aims at the *inter-domain* level (see solid lines in Figure 2.1) where trust relationship is vertically established between a workflow owner and participating services.

Centralised (T7) and Decentralised (T8) Trust Characteristics

In the *centralised* approach, the common way to obtain trust of a target is to consult a central trusted third party or guarantor that has information about the target and can vouch for its trustworthiness. However, the major problems of the *centralised* approach involve scalability and flexibility. Currently, a number of works are proposed to address trust in *decentralised* workflows ([32], [33], [34], and [35]) to overcome these problems. These two characteristics can be employed for all types of service workflows.

Service Delegation (T9) and Trust Transitivity (T10) Properties

Many studies have shown that these properties are essential and exist widely in theoretical and practical aspects [23]. Recently, a great deal of research has explored the notions of *service delegation* ([32], [36], and [37]) and *trust transitivity* (propagation) ([38], [39], [40], [41], [42], and [43]). The notion of *delegation* is meaningful in service workflows, where one service can be hierarchically composed of sub-services, and these services might be delegated to other service providers.

Direct (T11) and Indirect (T12) Trust Characteristics

Interactions in a workflow path can generally be examined in two aspects: *direct* and *indirect*. The interaction that occurs between immediate services is considered *direct*. *Indirect* interactions occur between services that are not immediately connected, but there is a path connected from one to another. All of these characteristics reflect the different aspects of requirement specification.

Trust in Singleton (T13) and Composite Service (T14) Workflow

The proliferation of services has led to the creation of composite services. A task in a service workflow might be executed by multiple services. This characteristic entails a challenging problem of how to evaluate trust in a composite service in accordance with trust from each sub-service. Without the strong foundation on trust of an individual service, it is difficult to justify or derive trust values of composite services, or service workflows.

So far, we have reviewed the relation between workflow characteristics and trust properties although some might argue that these specifically depend on applications and contexts in concern. However, the intention is to provide generic information and connections that would be valuable for workflow and trust researchers. The following section provides a short survey of the literature which implicitly relates to workflow characteristics and trust properties.

Table 2.3: A Summary of Trust in Service Workflows

Trust in Grid Contexts
- Policy- and Credential-based Trust in Grids
  - (GSI, 1998) [48] (C1,4,7) (T6,8)
  - (Tie-Yan et al., 2003) [55] (C1,4,7) (T7,8)
  - (Globus, 1998) [53][54] (N/A)
  - (GTS, 2007) [56] (C4,7) (T7)
  - (Naqvi, 2006) [57] (C4,7) (T8)
  - (Naqvi, 2006, 2009) [58][59] (C4,7) (T7)
- Trust in Resource Management
  - (TRMS, 2002) [60] (C7) (T5,8)
  - (Sedrakian et al., 2007) [61] (C5,7) (T5,8)
  - (SARAH, 2004) [62] (C5) (T7, 8)
  - (SeGO, 2005) [63][64] (C5) (T7,8)

Trust in Virtual Organisations
- (Mayer et al., 1995) [14] (N/A)
- (Olmedilla et al., 2005) [18] (C4,5,6,8) (T5,7,8)
- (Griffin et al., 2005) [66] (N/A)
- (TrustCom, 2004) [67] (C1,4,8) (T5,7,8,13,14)
- (Svirskas et al., 2005) [68] (C4,6,7) (T5,8)

Trusted Computing in Service Workflows Contexts
- (Song et al., 2008) [70] (T3,7)
- (Yau et al., 2008, 2009) [71][72] (C1,6,7) (T1,7)
- (Kuntze et al., 2008) [73] (C7,8) (T5,11,12,13)
- (FIDIS, 2008) [74] (T5,7)

Formal Trust Models in Service Workflow Contexts
- (Carbone el al., 2003) [77] (C4) (T9,10)

Trust in Service-Oriented Workflow Contexts
- (Dimitrakos, 2003) [79] (C1,4) (T7,8,9,10)
- (Arenas et al, 2005) [80] (C1,4,5,6)
- (WSTO, 2007) [81] (C7) (T5,7,8)
- (Maximilien and Singh, 2004) [82] (C4,5,6) (T8,13)
- (Kim and Doh, 2008) [83] (C4) (T8,13,14)
- (Yang et al., 2005) [84] (C4) (T8,13,14)
- (Lui, 2005) [85] (T8,10)

Trust in Provenance-based Service Workflows
- (Moreau et al., 2004) [88] (C1,6) (T4,6,11,12,13,14)
- (Lynch,2001) [89] (T7, 8)
- (Rajbhandari et al., 2006) [97][99][110] (C3,7) (T8,11,12)
- (SECPROV, 2008) [100] (C3,6) (T5,6,8)
- (Lyle and Martin, 2010) [101] (C3,6,7) (T8)

Other Aspects of Trust in Service Workflow Contexts
- ISO 7498-2 (T3)
- (Ahmed and Tripathi, 2002) [112] (N/A)
- (Ramchurn et al., 2003) [5] (C5,6) (T4,8)
- (FIRE, 2004) [113] (C5,6) (T3,4,8)
- (W-RBAC, 1996)[114] (C2,3,7) (T3,7,9,10)
- (DW-RBAC, 2007) [115] (C2,3,7) (T3,7,9,10)
- (HEN, 2007) [117] (C4,8) (T4,5,9,10)
- (HEN+, 2009) [118] (C4,8) (T4,5,9,10)
- (Altunay et al., 2005) [119] (C1,4,5) (T3,5,6,7,11,12)

2.4 Literature Review

Since service workflows are inherent in many activities such as business cooperation, academic organisations, scientific applications, e-commerce, and day-to-day interactions, various aspects and meanings of trust span across many disciplines. This section summarises the works in this area. We begin with trust centred on Grids that ultimately utilises workflows for coordinating distributed resources, followed by the general works in

multiple domains. Although some parts overlap conceptually, the survey, rather focusing on individual contributions, provides a reference of trust exhibited in service workflows (see the roadmap in Table 2.3). For the complete survey, please refer to [4].

2.4.1 Trust in Grid Contexts

Grid is a mainstream technology that requires the coordination of resources to deliver large-scale services. Given its unique characteristics [44], it highlights the exclusive security and trust challenges. Much research has been motivated by Howard Chivers' [45] observation.

> *"Workflow systems are often interposed between users, data and services without considering the trust responsibilities that this design imposes on planning and enactment systems."*

A document survey by the CoreGrid project ([46] and [47]) differentiates trust from security, and presents deployable concepts and technologies pertaining to trust and security in Grids. The following subsections provide other aspects of trust in Grids based on various research areas.

2.4.1.1 Policy- and Credential-based Trust in Grids

Security in early Grids primarily relied on conventional security ([48], [49], and [50]) utilising authentication, access control, integrity, privacy, and non-repudiation, where the security specifications are explicitly defined by security policies ([51] and [52]). This concept leads to policy- and credential-based trust, pertaining to the process of verifying identity and determining access rights for authentication and authorisation. Trust relationship is initially evaluated from the supplied credentials, reflecting access rights granted. Several applications of this concept are found in the Globus project ([53] and [54]) implemented on Grid Security Infrastructure (GSI) [48]. The early GSI borrows conventional security and SSL technology, focusing mainly on user authentication based on X.509 certificate. Several extensions have been built on top of GSI middleware with various security plug-ins (see the official Globus website[1] for more details). To date, existing proposals on trust attempt to improve GSI in many aspects. Tie-Yan et al. [55]

[1] http://www.globus.org

points out the need for two levels of trust relationships for GSI-based Grids: across VOs and inside a Grid domain. The implementation incorporates a distributed trust approach (T8), using recommendations to compute trust between VOs, and a centralised trust approach (T7) to evaluate trust inside a Grid domain.

In collaborative distributed environments, Langella et al. [56] addresses the complexity and dynamicity of credential management in authenticating and authorising users to other peer-services. They present a Grid Trust Service framework (GTS) for validating users and services against a known set of trusted certificate authorities stated by security policies. The inherent problem of using policies is that they are usually specified informally in the form of narrative statements, which make it difficult to accommodate dynamic changes in requirements. Naqvi et al. [57] studied languages for formal requirement specification in Grid systems and presented rigorous approaches ([58] and [59]) to formally develop security policies elicited from local informal requirements in large-scale distributed environments. Consistency, correctness, and wide applicability are direct benefits from requirement formalism. However, none of them include workflow characteristics in association with trust properties, which consequently makes it difficult to apply the languages into the service workflow area.

2.4.1.2 Trust and Resource Management in Grids

Managing security based on conventional approaches results in high overheads, which negates the performance in large-scaled dynamic Grid environments. As such, trust, commonly referring to QoS and reputation in this context, seems to be an appropriate supplement to these barriers. The early work found in the Trust-aware Resource Management System (TRMS) [60] integrates a reputation-based trust model into the Grid central resource scheduler for resource allocations. The initial trust value varies depending on direct trust and recommendation information, and decays with time. In the CoreGrid[2] project, Sedrakian et al. [61] introduced a solution for jobs mapping to the most suitable resources, in which the performance is determined by reliability, availability, and other application-specific requirements of the nodes in consideration. Song and Hwan [62] developed a fuzzy inference trust management model called SARAH to quantify trust, and

[2] http://www.coregrid.net/

introduce centralised policy management to consolidate trust policies over different sites. The major advantage is to allocate the best performance/cost ratio of collected Grid resources. This concept is implemented in the SeGO ([63] and [64]) scheduler, where a security-binding scheme based on fuzzy logic is used to aggregate trust across Grid sites.

However, all existing works on trust and resource management in Grids rely on global workflow management (C7) through a centralised engine. Therefore, it is unable to address relationship between services inside a workflow.

2.4.2 Trust in VOs

One advanced form of grid services, with the capability to acquire services across distributed administrative organisations, refers to VOs. When coordinating services across multiple locations, trust is a major driver for enabling collaboration among the involved parties. The first proposal of trust in VOs is found in [14]. This work is noted for the effort of an organisational-related trust model correlating with risks in a particular trust relationship. Three factors are identified for evaluating trust: ability, benevolence, and integrity.

According to [47], the strategic research roadmap [65] develops a lifecycle for basic VOs with four phases: (1) identification; (2) formation; (3) operation/evolution; and (4) dissolution. Adopted in [18], Olmedilla et al. relates security and trust requirements with respect to each phase. Two complementary approaches are given: Access Control and Provision of Services, from which trust can be inferred. They identified the requirements to underline trust establishment and evaluation based on credentials, policy, and reputation interposed in the lifecycle. Such requirements are encoded by semantic representations.

Other works can be found in dynamic VOs for cross-organisational workflow interoperations. For example, Trusted Virtual Domains (TVD) [66] are intended to tie together a number of new and previous works on Trusted Computing (TC), policy specification and verification, virtualisation and middleware technologies, and others, in order to establish a secure network from remote trusted domains. The TrustCom Project [67] is developed on the convergence of emerging technologies such as web services and

Grids to support secure collaborative business workflows on demand, and monitor contract-related compliance. In response to this project, Svirskas et al. [68] conducted an analysis on the design and demonstrated how web service specifications and trust are undertaken to promote secure and trustworthy VOs in practice.

Generally, trust relationship in VOs' collaboration is established between two direct interacting entities, each of which manages its own internal workflows that possibly collaborate with multiple services. Currently, there is no effective way to determine the trustworthiness of such services resulting in the difficulty for workflow participation.

2.4.3 TC in Service Workflows

Recently, different usages of the TC technology [69], based on the hardware-based Trusted Platform Module (TPM), have been spread in many security areas for platform trustworthiness. This section covers the major and recent studies on TC in the workflow-related context.

Identification is a core feature of TPM uniquely adhered to a specific platform using public key cryptography as a root of trust (Endorsement key). Song et al. [70] combined TC identity and reporting features with a proposed Integrated Authentication Service (IAS) that permits not only authentication of people's identities, but also platforms and environments.

In service-oriented contexts, the full body on TC adoption to enhance Grid workflow security was introduced by Yau et al. ([71] and [72]), as they best describe the security services TC offers, e.g. resource selection through platform state reporting, confidentiality and integrity of job information using cryptographic keys, and audit data for process provenance based on attestation. Kuntze et al. [73] explains in more general the security and trust implications in business processes and introduced TC as a means to achieve integrity and authenticity among business entities involved in a workflow.

In the aspect of TC in practice, Future of Identity in the Information Society (FIDIS)[3] [74] provides comprehensive studies on the investigation of (1) whether TC features are suitable to support privacy in business processes, and (2) the impact of TC on Identity and

[3] http://www.fidis.net/

Identity Management. The authors conclude that the idea is conceptually promising, but at this moment the TC technology is not widely used or available. An analysis of TC's benefits is also conducted on the effect of dynamicity of organisational workflows that have higher risks of information leakage [75].

At this stage, the realisation of TC in service workflows is still largely in the research community, rather than in industrial practices. It can be used to ensure the trustworthiness of the target services which regarded as properties for trust evaluation. However, as TC is a property-based function, it cannot be solely used to address all aspects of trust in service workflows.

2.4.4 Formal Trust Models in Service Workflow

Although trust has long been investigated, one important topic with less attention is in the area of trust formalism. The first and widely cited work on this area is the doctoral dissertation by Marsh [76]. The key aspect lies in providing social sciences with a tool where the quantitative computational trust value, a continuous value between -1 and 1, can be used to support decision-making. Wagealla et al. [77] attempted to develop a formal model on the correlation of trust and risk in a way to facilitate decision-making in a service-based collaboration. Davulcu et al. [78] devised a formal framework for modelling and reasoning about interactions in virtual enterprises. However, only a few formal languages have been provided in the service workflow area. In the next chapter, the comprehensive details regarding related works on these languages with the introduction of the TS language to capture requirements for trust in service workflows are provided.

2.4.5 Trust in Service-Oriented Workflow

Despite some overlaps in concept with the previous sections, this section describes generally the aspects of trust in service-oriented workflows, rather than being restricted in Grid environments. Nowadays, the fruitfulness of service-based workflow interoperation, largely found in the form of web services, requires effective trust models to capture, measure, and manage trust among participating services. Introductory detail and analysis of trust covering the areas of trust definitions, classification, and properties in SOA was first

offered by Dimitrakos [79]. Several service-oriented trust researchers have been motivated by the fundamental ground of this work.

With respect to the VO's lifecycle [65], a comprehensive framework inspired by Dimitrakos's proposal has been developed in the TrustCom Project [67]. Trust based on access control and reputation (performance) is used in accordance with contract agreements to systematically manage trust between web services. Arenas et al. [80] characterised trust and security requirements, and demonstrated how current web service specifications can meet those requirements throughout the VO's lifecycle. At a higher abstraction, Web Service Trust Ontology (WSTO) [81] is taken in a wider sense as a common knowledge reference that enables parties to semantically describe and use trust requirements with the same understanding. WSTO is an open ontology incorporating different trust approaches that are considered suitable in each particular context.

In dynamic distributed environments, a fundamental challenge of how to evaluate trust emerges in service composition scenarios. Maximilien and Singh [82] extended the classic SOA with an approach for trust-based service selection. An intermediary special-purpose service is introduced for determining the trustworthiness of other potential services in concern. However, this approach regards trust only on a single service, whereas in a more complex scenario a service composed of trusted sub-services is not guaranteed to be trusted. In response to this, Kim and Doh [83] developed QoS-based inference rules to derive trust in a composite service in compliance with each sub-service trust value. The alternative method used to estimate composite service trust utilises statistical techniques [84]. Five types of basic constructs (Sequence, Split-join, Choice, If-Then-Else and Iterate) are used in the Petri-net-based workflow and are parameterised to estimate trust. A more abstract approach for trust in service composition is found in [85]. The author presented a conceptual framework that allows trust to be evaluated from a variety of societal conventions, including personalised service recommendation and distributed trust propagation based on referral networks. However, the success relies on the assumption that proportionally-large benign agents are involved, so that malicious rating attempts become insignificant.

Several web service standards exist in SOA security. WS-Security [86] defines the basic security detail using digital signature (integrity) and encryption (confidentiality), for secure messaging. WS-Trust [87], an extension of WS-Security, enables trust in message exchange by using security tokens, a set of claims, e.g. ID, name, key, etc, as a proof against WS-Policy [86]. One of the major limitations of these standards is that they treat trust as a binary decision (trust or not trust) on a particular message and do not support a high level trust relationship beyond the message level.

2.4.6 Trust in Provenance-based Service Workflows

In the past decade, the advent of provenance-based trust has been realised in many e-experiments and commercial workflows. Provenance is a technique to enable users to determine the trustworthiness of the selected outcome, especially essential when highly valuable data is involved. The position paper [88] of provenance-based trust exhibited in service workflows is a good starting point for basic understanding. Clifford A. Lynch [89] addresses interesting issues about trust in information retrieval from diverse distributed sources on the Internet. The work indicates the trustworthiness of a piece of information evaluated from an identity of a particular source, and trusting the source is dominated by personal preferences. In this work, the perspective of trust and provenance is identical to what is described in the semantic web, where metadata is annotated in such a way that trust can then be inferred ([90] and [91]).

There is a growing amount of attention on trust and provenance data in scientific workflows (for example, Chimera [92], ^{my}Grid [93], CMCS ([94] and [95]) and Trio [96]), as to assist e-scientists in evaluating the level of trust to be placed in an outcome. Key works in this area are suggested by Rajbhandari et al. ([97], [98], and [99]). They proposed a probabilistic-based framework to exploit automatically-recorded provenance data of a workflow for trust assessment, and used a decision tree model for a trust judgment on the results generated from scientific experiments. The model is extended by integrating fuzzy inference rules [3] into the framework, where trust is regarded as a subjective opinion represented in qualitative scales (high, medium, low). However, there are two obvious shortcomings: (1) the use of Beta Probability Density Function (PDF) to compute trust fails

to capture the recent improvement of a service that will suffer from punishment of its long-history of failures; and (2) no protection is addressed for sensitive provenance data. The latter is fulfilled by SECPROV [100] that attempts to formalise and specify security at a granular level based on access control mechanisms. Lyle and Martin [101] proposed a conceptual idea in employing TC in so-called trusted provenance systems. The difficulty arises in provenance recording when workflows are executed in distributed different-control domains. Cruz et al. [102] explored this characteristic and presented a Matrioshka architecture to globally keep track of provenance. However, fine-grained collaboration is needed for provenance format conformity, although semantic web technology can assist with this, as it has shown to be suitable in recording provenance in the form of semantic representation.

Provenance is a means for trust evaluation in service workflows as its records contain trust-related information such as the rating score of a particular service. Its role serves as an information container, providing the information pedigree of a specific outcome. Existing works however do not indicate precisely the formal requirements for trust with respect to workflow characteristics that the provenance technique can satisfy.

2.4.7 Other Aspects

2.4.7.1 Trust and Access Control in Service Workflows

Security is a fundamental part of WFMSs to ensure workflow executions in a secure way. One of the earliest motivations in securing workflows involved the implementation of access control ([29], [102], [103], [104], [105], and [106]), frequently specified by security policies to gain access to a required object through authorisation models. Recently, a number of later proposals ([107], [108], [109], and [110]) attempt to deploy access control in distributed environments. Access control mechanisms continue to advance in accordance with the advent of new technologies, such as the semantic web, web services, and trust. One prominent example is the work of trust based on access control models [111]. To address dynamic nature in distributed environments, a trust establishment framework is integrated into WFMS, enabling dynamic user-role assignments when not all users and

associated tasks are known in advance, whilst preserving those security benefits that typical access control provides.

However, the integration of trust and access controls seems to be too narrow from the security research outlook. Security in workflows should be wider from a number of different perspectives, as stated in the ISO 7498-2 standard identifying five requirements for information security of workflow systems: identification and authentication, confidentiality, non-repudiation, access control, and integrity. In addition, Ahmed and Tripathi [112] provided more comprehensive security requirements in distributed workflow environments and discussed the limitations and challenges in relations to policy-based access control models, e.g. RBAC.

2.4.7.2 Reputation- and Behavioural-based Trust

Many recent models symbolise trust by a means of measuring confidence and reputation, normally based on a rating derived from one's past experience, history of interactions and recommendation. Given an existing trust measure, approaches exist to recommend (transfer) trust to other agents. This plays an important role in dynamic open environments, where each individual acts in an autonomous manner and interaction usually takes place between previously unknown parties. Ramchurnk et al. [5] differentiates confidence and reputation from which trust can be inferred, and shows how the result influences the process of selecting reliable partners to form a workflow. FIRE [113] combines four different types of trust, (1) direct experience, (2) role-based inter-relationship, (3) referral, and (4) testimonial evidence, in order to manage trust in multi-agent systems. One disadvantage is the reliance on the assumption that rating information is always trustworthy, whereas this is hardly possible in the real-world circumstance. Thus, it is necessary to build a reputation mechanism to enforce truthful recommendation elicitation.

2.4.7.3 Service Delegation and Trust Transitivity

Several researchers ([32], [36], [37], [38], [39], [40], [41], [42], and [43]) have explored the notions of service delegation and trust transitivity as the essential properties recognised in many usages. One obvious example is the use of certificate chaining, where the derivation of identity trust is based on trust transitivity [24]. However, there is less support for these

properties exhibited in service workflows nowadays, which makes them inflexible and impractical in large-scale environments. This section is devoted to summarising the state-of-the-art of these properties found in the workflow-related context.

Initially, delegation is strongly adhered to access control mechanisms where a set of access rights and tasks are delegated to other authorised parties. The same approach is realised in workflows by enhancing workflow tasks with *-based access control models ([114] and [115]). The interpretation of delegation policies and constraints refers to the trust level given to an authorised user [116], which is frequently evaluated by the validation of credentials (certificates or chained certificates).

In distributed open environments, one challenging problem arises in the process of service/workflow composition, where delegation constraints become difficult to enforce due to the lack of knowledge of how the task is being delivered from others. Moreover, there remains confusion regarding how to evaluate trust in a service and a chain of delegated services. In response to this, the HEN [117] architecture, a delegation-based model for distributed trust, is first introduced to formulate delegation based on QoS with three parameters: success rate, duration, and price. HEN+ [118] generalises this framework by capturing broader aspects, including: (1) the introduction of mutual subjective trust between a parent and its sub-services, (2) initial trust stemming from policies and the compliance of requirements, (3) trust evaluation based on QoS, (4) better control of service delegation, and (5) trust propagation on a chain of delegated services. Further details on service delegation and trust transitivity are outside the scope of service workflows which can be found in ([19], [37], and [46]).

2.4.7.4 Trust in Service Workflow Structure

The current security progress in WFMSs centres on workflow enactment, when attacks are most visible. However, without security awareness during workflow formation might pose unnecessary risks, which become visible during run-time exploits. Integrating trust into workflow formation is crucial in regulating the processes of service acquisition, composition, and executions. A service that violates security policies might be replaced, while conflicts among participating services in a workflow might result in workflow

structure reformation, constraint relaxation, or service substitution. This makes a service owner aware of trust at the early stage, consequently reducing risks during workflow enactment. The idea of integrating trust into workflow formation was first presented by Altunay et al. [119]. The work indicates trust relationship between services, both directly and indirectly connected, in a workflow path.

Although several aspects of trust have been proposed, the absolute trust solution is far from being accomplished. As a result, many researchers tend to combine existing approaches, as their relative strengths and weaknesses complement, for a specific use. This brings about more challenges, for example, which trust models are suitable and how to elaborate trust when one type might dominate or influence over the others.

2.5 Chapter Discussion

From the above survey, it can be concluded that trust is a driving factor for service workflow collaboration. Trust can be determined from a number of elements depending on the context of use. These elements can be considered properties of services. One way to specify the properties is to use requirement specification languages. Compliance of requirements reflects trust-based decision of workflow collaboration. More specifically, the compliance of requirements from a workflow owner establishes trust of services in the process of service acquisition and composition, while the compliance of requirements from services affects trust-based decision in workflow participation.

One problem that prevents today's technologies from realising this vision is that there is no formal way to capture requirements for trust in service workflows. From this survey, there is no uniformed approach to address requirements for trust, which results in inefficiency for requirement specification. In addition, it can be anticipated when advanced technologies ease the processes of service collaboration, enabling any service to be able to participate in service workflow as long as it satisfies pre-defined conditions. As such, there is a consequent need for automation in compliance checking of these requirements. This

requires formal foundation and machine readable specification language to express requirements that can be used for automatic reasoning.

Chapter 3

Trust Specification Language for Service Workflows

3.1 Chapter Introduction

This chapter introduces TS language and its notions that capture wide ranges of workflow characteristics and trust properties described in Chapter 2. Essential logical operators based on FOL and CTL are invented to form TS formulas. The reason of using a logical language is that it is machine readable and verifying properties are naturally reduced to one problem [78]. To overview its application, an example of a travel agent workflow is illustrated at the end of this chapter. The complete scenario of this workflow will be provided again in Chapter 6 and for evaluation in Chapter 7. Euclidean Distance is used as an example method for calculating the closeness of a matched trust values, and its result indicates the strength of the relationship. We note that this method, which is not the focus of this thesis, is only used as an example to illustrate one of many possible ways for trust calculation, while many others can also be applied. For instance, weighted parameters can be used to give more influence of one value than the others.

Everything as a Service (EaaS) concept enables dynamic resource provisions to be seen and delivered as services. The emergent need of workflows to model e-service applications makes it essential that workflow tasks must be associated with services. This consequently

enables a workflow composed of a variety of services to become more popular and support sophisticated activities. From the viewpoint of use, there are three roles of service workflows in practice (see Figure 3.1).

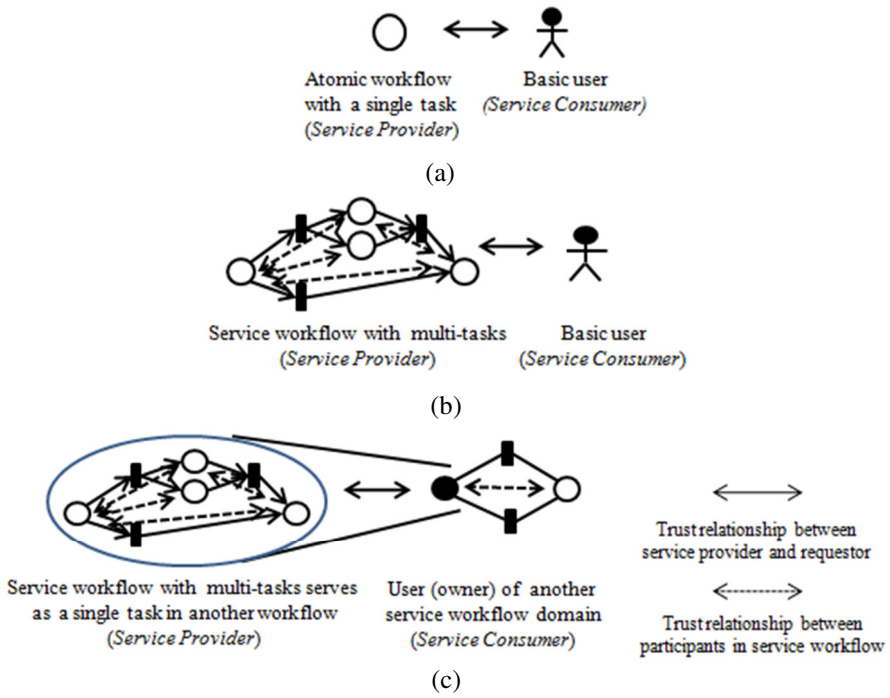


Figure 3.1: Three Ways of Service Workflow in Practice

1. Figure 3.1(a) is the simplest form where a workflow is comprised of a single service for a single task.
2. Figure 3.1(b) illustrates a workflow with internal composite services. A workflow task is not limited to being executed by a single service. In this way, a service consumer only perceives that he interacts with only one service offered by a workflow owner.
3. Figure 3.1(c) presents the concept of a nested workflow; its role is to fulfil a task(s) that belongs to another workflow. This concept increases scalability in distributed environments by permitting a workflow to be hierarchically composed of multiple sub-workflows.

In the context of service workflows, there are two aspects of trust where TS language plays its role. Firstly, it assists a workflow owner to express its requirements in the process

of service acquisition and composition (solid line), and secondly it facilitates services with a tool to express their requirements affecting trust-based participation decision in a workflow (dash line).

It can be observed that these aspects of relationships can be evaluated from several factors such as hard and soft security mechanisms, reputation score, the compliance between policies and credentials, TC, and many more. They are considered properties where the requirements for these properties can be encoded by TS formulas. For example, in order to participate in a workflow, service X requires service Y to have reputation score more than 0.8, and service X requires that service Y must possess an identification card issued from the US government. In this case, reputation score and identification card are two properties of service Y that service X requires, which can be formally encoded by TS formulas. In this context, trust-based decision making in workflow participation is a logical notion which is determined from the compliance of requirements.

As a result, trust-based security among services significantly governs participants' willingness to collaborate and participate in a workflow [119]. The statement is evidenced by the existence of much research, such as web service in business process ([39], [41], [42], [43], [81], [118], [120], [121], and [122]). However, none of them provides sufficient frameworks in specifying requirements for trust-based decision making in service workflows. This chapter attempts to fulfil this gap by introducing TS language which can be used for compliance checking of requirements. It provides four main contributions.

1. This language is a means to formally and uniformly represent requirements across multiple domains. This allows requirements to be independently and consistently expressed, enabling services to interact with others in open environments.
2. With its well-defined syntax and semantics, the language serves as a key driver for reasoning between requirements in the form of TS formulas and service workflows. Since requirements are uniformly expressed, it has no need to devise distinct complicated algorithms to deal with unrelated formats. Therefore, it supports automatic compliance checking which directly leverage scalability, and simultaneously increase efficiency in large-scale interoperation.

3. In rapidly changing environments that may affect the willingness of services to participate in a workflow overtime, the language provides a flexible means allowing the relaxation or restriction of requirements on-the-fly, subjecting to current needs.
4. Finally, this language equips a workflow owner with a tool in the process of service acquisition and composition, which regulate how a workflow is constructed in accordance with his requirements.

The remainder of this chapter is organised as follows. Section 3.2 describes in general the application of TS language in service workflows. Section 3.3 presents the important criteria of specification languages in service workflow. In addition to Chapter 2, Section 3.4 described more specific related works focusing on logical languages and workflow modelling. Section 3.5 introduces the definition of Service Workflow Net (SWN) to mathematically abstract service workflows in a way to be able to reason with TS formulas. Section 3.6 commences on the first step towards formalisation including three modes of TS language and its definitions. To form TS formulas, important operators are proposed. In Section 3.7, an application example of travel agent workflow is provided to illustrate how the language can be employed in practice. Section 3.8 demonstrates an example of trust calculation by using Euclidean Distance for calculating the closeness of a matched value. A basic analysis is conducted in Section 3.9 and then the chapter ends with a brief discussion.

3.2 Introduction to Specification Language in Service Workflows

The fundamental idea of TS language is to allow each party (workflow owners and services) to formally express its requirements that can be used to verify properties of services and workflows. From the owner's viewpoint, he can be assured to some extent that a workflow will operate according to pre-defined requirements. From the service's viewpoint, satisfying requirements reflect the trust level that consequently affects the decision making in workflow participation.

Normally, a workflow consists of several tasks and services. The process of acquiring services for a task nowadays is very dynamic due to the openness of environments that

allow services to leave or join at free will. To reduce the risk of availability, several services with the same functionality are redundantly acquired and will be dynamically activated. For a complicated task, several services with complement capabilities are required for execution in a composite manner.

In order to make a service workflow automatically verifiable by TS language, mathematical representation is required. Our workflow modelling language, namely SWN, is developed based on Petri Net, where place and transition notions are used to represent services and tasks, respectively (see its justifications and formal definition in Section 3.5). The graphical notation of SWN is used to represent high-level dependency of services and tasks, while the mathematical part is used to encapsulate the dependency regarding service composition, service separation, and task coordination. All of these elements are controlled by a basic set of workflow connectives: AND, OR, and XOR. Despite being restricted in the basic ones, they are general enough to address all possible ways of workflow instantiation.

3.3 Criteria for Specification Languages in Service Workflows

In this section, important criteria form [25] that workflow specification languages should address are summarised.

R1: Interoperability with Local Requirements (ILR)

To capture requirements that are maintained locally from distributed services, specification languages should be invented in a way to represent requirements in the same format; therefore being interpreted in the same understanding.

R2: Separation of Duty (SoD)

In the context of service workflows, services with complementary competencies are joined to carry out a sophisticated task. SoD ensures that conflicting services will not occur. Specification languages should be able to address SoD, allowing services to impose their own SoD requirements in a service workflow.

R3: Association between Tasks and Services (ATS)

One capable service is allowed to execute several tasks in a workflow. Requirements of such service are differently determined depending on tasks. Specification languages should be able to differently specify requirements of a service with respect to tasks.

R4: Flow- and Task-oriented Specification (FTS)

The flow-oriented perspective concerns with requirements of services and tasks along an execution flow. The task-oriented perspective on the other hand focuses on the properties of services specific to a task. This is an exclusive requirement of service workflows that should be addressed by specification languages.

R5: Enforcement of Sequences (EoS)

EoS is a special case of the flow-oriented perspective as security of a workflow might depend on a sequence of tasks. One task might require the results from the upstream tasks, as well as to provide its results to the tasks involved at downstream [123]. This is an important aspect for specification language to be able to address EoS in service workflows.

R6: Flexible Degrees of Restriction (FDR)

The absence of the end-to-end visibility of a workflow which often relates to the absence of a single overall process ownership has led workflow researchers to re-examine and find new ways for workflow cooperation [124]. The benefit of varying degrees of visibility is that it enables entities to retain the different levels of privacy, as well as the confidentiality of internal processes. Specification languages should provide a flexible means to code up requirements with several degrees of restriction, depending on the degree of visibility.

R7: Protection of Workflow Data (PWD)

It is important to protect the data especially when it is highly sensitive. This has been sufficiently addressed by the protections offered by traditional security mechanisms and can be additionally supported by soft security approaches. These should be addressed as the

properties of services that are part of the data flow. Specification languages should be able to express the requirements for protecting data in service workflows.

R8: Temporal Constraints (TpC)

This requirement, usually tied up with temporal information such as temporal credential [125], represents fluctuations of trust of a target service(s). As mentioned in the previous chapter, a *transient* workflow frequently interacts with strangers in a short period, where trust relationship is temporary, while a *persistent* workflow requires longer term of trust relationship. Specification language should have capability in specifying TpC-related requirements.

For full details regarding the relationship of these criteria, workflow characteristics, and trust properties, please refer to [25].

3.4 Related Work

Although there are several works in this area [4], they neither provide sufficient formal frameworks nor effective models for specifying requirements specific to service workflows. The development of TS language is related to two research areas: service workflow modelling, and requirement specification languages.

There have been numerous approaches to define and describe workflow models, such as Petri Net ([126] and [127]), Pi calculus [128], UML activity diagram [129], etc. Among these approaches, Petri Net become the focus of research in workflow modelling since it can explicitly model workflows in both control-flow and resource dimensions ([130] and [131]). For example, in the control-flow dimension, Hamadi and Benatallah [132] embraced a Petri-net-based algebra to model control flows for web service composition. In the resource dimension, Tang, et al. [133] proposed Service and Resources Net (SRN) to address more complex demands by adding extra elements adding time, resources, and conditions to Petri Net.

In the area of specification languages, although they have long been studied, one important topic which has been treated less is in the context of service workflows. The following paragraphs summarise the prominent works that might relate to service workflows, and provide justifications why TS language is needed. Table 3.1 shows the comparison between each model towards criteria presented in the previous section.

Formalising Trust as a Computational Concept

Marsh [76] addresses key aspects of trust in providing the social sciences with a tool where the quantitative computational trust value, a continuous value between -1 and 1, can be used to support trusting decisions. This work is widely recognised as the progenitor of trust formalism, providing those who study trust with a means of discussion. The notion $T_a(b)$ is used to denote 'a trusts b' in a general context where the trust value is normalised as $-1 \leq T_a(b) < 1$. If two unknown agents are first met, the trusting agent will be assigned to 0 ($T_a(b) = 0$) . When a situation matters, the notation of situational trust where 'a trusts b in situation s is represented by $T_a(b, s)$. To be able to represent time in this notion, a temporal index can be added, referring to a trust value at a specific point of time, $T_a(b, s)^t$. The contribution of this work encompasses several aspects, including utility, importance, and knowledge to quantify trust, providing a tool for an agent to discuss and reason with and about trust. The use of a single value for trust allows qualitative and quantitative comparisons of different agents' values. Trust is subjective in this context. However, this work is hardly applied to service workflows as it is not comprehensive enough to capture criteria mentioned earlier. For instance, it is unable to address Flow- and Tasks-oriented Specification (R4) and Enforcement of Sequence (R5).

Logic Based Approaches to Workflow Modeling and Verification

Davulcu et al. [78] devised a formal framework based on Concurrent Transaction Logic (CTR) for modelling and reasoning about interactions in virtual enterprises. Workflow is modelled by a Direct Acyclic Graph (DAG). A set of CTR connectives is introduced for flow control. There are five important CTR connectives: (1) $\phi \otimes \psi$ means that ϕ must be executed before ψ; (2) $\phi \mid \psi$ describes ϕ and ψ must be executed in parallel; (3) $\phi \wedge \psi$

indicates ϕ and ψ must be on the same path; (4) ϕ ∨ ψ restricts ϕ or ψ is executed; and (5) ~ ϕ represents the negation of the formula ϕ. Since the framework is rooted in logic, one major advantage is to permit automatic reasoning about the intended behaviour of virtual enterprises to ensure that they function as specified. Although these connectives are sufficient to specify workflow constraints on task coordination, they are not expressive in terms of capturing requirements from each service. For instance, it is partly realise Separation of Duty (R2), and fails to capture Association between Tasks and Services (R3). Furthermore, several other notions are required in order to address workflow characteristics. For example, as a workflow is a tree-like structure, path quantifier operators are very much important.

Table 3.1: Comparison between Existing Works on Criteria

Models	R1	R2	R3	R4	R5	R6	R7	R8
Stephen Paul Marsh	×	×	×	×	×	✓/×	×	✓
Davulcu et al.	×	✓/×	×	✓	✓	✓	✓/×	×
Altunay et al.	✓	✓/×	✓/×	✓	✓	✓	✓	×
HEN+	×	×	×	×	×	×	✓	✓/×
Predicate and Modal Logics	✓	✓	✓/×	×	×	×	✓	✓
Trust Management	✓	✓/×	×	×	×	×	✓/×	✓
TS Language	✓	✓	✓	✓	✓	✓	✓	✓

✓ Direct Support, ✓/× Partial Support, × Not Support

Trust in a Workflow

The closest to our work by Altunay et al. [119] aimed at integrating trust into service workflows. Trust relationships are considered not only between the workflow owner and participating services, but also among services themselves. The relationships are examined in two categories: direct and indirect. The direct relationship occurs between two services that are immediate neighbours, whereas the indirect one is concerned with two services that are not immediately connected. However, this work is only applicable to a workflow modelled by a simple graph, whereas it is far more complex in the real world. Moreover, it neither formally expresses nor comprehensively addresses the essential requirements such as Temporal Constraint (R8).

HEN+

The work on a Petri-net-based framework, namely HENS+ [118], is used to support the characteristic of inter-domain trust relationships, emphasising on service delegation and trust transitivity in dynamic workflow collaboration. This work identifies the necessity for relationships between interacting domains. Even though HENS+ addresses comprehensive trust relationships at the domain level, it cannot specify requirements for trust of interacting services inside a workflow.

Logical Languages for Trust

Logical languages for trust have emerged from access control policies in authentication and authorisation systems [134]. Currently, they are extended to cover broader areas such as virtual community [135], distributed systems [136], and multi-agent systems [137]. The forms of first order predicate logic ([134] and [138]) and modal logic can be used to express possibility, necessity, belief, knowledge, temporal progression and other modalities. The logic for trust is usually in the form of $\mathrm{predicate}(\mathrm{subject}, \mathrm{object}, \mathrm{attr} \ldots)$ where $\mathrm{predicate}$ is an action applied to variable $\mathrm{subject}$ and object with attribute attr. Other propositional operators such as $\wedge$, $\vee$, and $\rightarrow$, connect each term of predicate. Although they are very expressive in a system where subjects, objects, and actions are explicitly known, they are impractical to address the exclusive characteristics in the workflow domain. For instance, they cannot address Flow- and Task-oriented Properties (R4) and Enforcement of Sequence (R5).

Trust Management

Trust management has its roots in authentication and authorization by using digital certificates as a means to establish trust [139]. Policy languages, such as REFEREE [140], Policymaker [141] and Keynote ([142], [143], and [144]), are unified mechanisms providing a common language for policies, credentials, and trust relationships. Trustbuilder [145] addresses the way to establish trust between strangers through credential exchange. However, even though certificates, which are the core of trust management, may provide a means to introduce an initial trust, it cannot guarantee the performance of a target.

Furthermore, a common problem of these works is that they are used to identify a static form of trust, which cannot cover monitoring and evaluation of trust over time [19]. In the area of service workflows, they are unable to address the important criteria such as Flow- and Task-oriented Properties (R4) and Enforcement of Sequence (R5).

Although a number of formal approaches have been proposed, most are incomplete as one approach might be appropriate for expressing trust in general aspects and the others might be appropriate for reasoning. In this chapter, TS language is developed to address all criteria inspired by Davulcu et al. [78], Altunay et al.'s [119], HENS+ [118], and trust management ([140], [141], [142], [143], [144] and [145]).

3.5 Service Workflow Net (SWN)

This section describes a formal definition of SWN which is a Petri Net variant to abstract service workflows in a way to be reasoned by TS formulas. The model is motivated by three existing frameworks, i.e. SRN [131], HENS+ [118] and Workflow Net ([126] and [127]). SWN extends the basic Petri Net with a set of workflow connectives, which provide the basic control of the types of workflow activations. The definition SWN is described below.

Def. 3.1: SWN is a labelled Place/Transition Net, i.e., a tuple $\mathcal{M} = (\mathrm{P}, \mathrm{T}, \mathrm{R}, \mathrm{f}, \mathrm{l})$ where

1. P represents a set of places/services of a workflow (circle);
2. T represents a set of transitions/tasks of a workflow (rectangle);
3. P and $\mathrm{T} \neq \emptyset$; and $\mathrm{P} \cap \mathrm{T} = \emptyset$;
4. $\mathrm{R} \subseteq (\mathrm{P} \times \mathrm{T}) \cup (\mathrm{T} \times \mathrm{P})$ represents directed flows;
5. $\mathrm{f}: (\mathrm{P} \rightarrow (\psi \times \{\mathrm{split}, \mathrm{join}\})) \cup (\mathrm{T} \rightarrow (\psi \times \{\mathrm{split}, \mathrm{join}\})$ is a function containing a workflow formula (ψ) indicating services for a task in either *split* or *join* types. ψ has three basic connectives: AND, OR and XOR. XOR is different from the definition in propositional logic as it restricts only one path to be chosen;
6. $\mathrm{l}: \mathrm{P} \rightarrow \mathrm{A} \cup \{\tau\}$ is a labelling function where A is a set of attributes, and τ denotes a null value. It is used for labelling services' properties.

3.5.1 Why Petri Net?

This section provides the justifications of using Petri Net as a workflow modelling method. Currently, most control-flow workflow languages support the basic connectives of sequence, iteration, splits (AND, OR, and XOR) and joins (AND, OR and XOR) [146], and other complex ones can be implemented either using workaround techniques such as mixtures of basic constructs, or at the higher level through API [147]. Since TS language in is related to control-flow dimension, Petri Net, originally developed for this purpose, is employed for SWN.

Compared to other languages, Petri Net is widely accepted as a workflow modelling method ([129], [148], [149], [150], and [151]) with extensive analysis techniques [152]. It provides advantages in modelling workflow by its expressiveness and Turing completeness, and therefore can do anything in terms of algorithms [147]. There are six reasons for Petri Net (extended from [149] and [153]) as an attractive choice among others including pi Calculus and UML activity diagrams. There reasons are that it

1. has mathematically-based formal semantics despite the graphical nature, making it self-documenting and a powerful design tool [129];
2. is state-based instead of event-based which facilitates compliance checking algorithms;
3. has an abundance of analysis techniques;
4. has the support of a hierarchy net, which is an aggregate of subnets where the higher nets are not basic notions but are derived from their counterparts;
5. sharply distinguishes between places (services) and transitions (tasks); and
6. has graph-like representations that have useful links both to graph theory and algebra.

The mathematical foundation of Petri Net has made it possible to set up a workflow for formal analysis, verification, and validation. In addition to the original definition, SWN is opened to accommodate more types of connectives. The particular reason of using Petri Net rather than using other graph-based languages is that it contains a set of places and transitions corresponding to services and tasks, respectively. This clearly expresses Association of Tasks and Services (R3). Tasks in elementary form are atomic units of work

that can be fulfilled by a single service, and in composite form require several services to complete. This assists security analysis such as Separation of Duty (R2).

In the context of verifying SWN, instead of precisely detailing operations at runtime as in traditional Petri Net does, SWN is a means to represent workflow structure of services and tasks, addressing all possibilities of executions. The reason is that the activation of tasks and services at runtime cannot be known in advance due to nondeterministic factors such as external inputs. From this characteristic, firing transition in Petri Net, which is used to deal with workflow operations deciding which and how services and tasks are going to be activated at runtime [147], are not the part of our verification. This chapter specifically focuses on specifying properties of services in a workflow structure. In dealing with requirements in workflow instantiation, TS language can be extended with additional operators. This issue will be discussed in Chapter 8 which is left for future investigation.

There are some criticisms of Petri Net in supporting advanced connectives. For example, there is no direct support for multiple instances and it is difficult to model advance synchronisation [147]. This problem can be resolved by using workaround techniques such as using API. In our essence, Petri Net together with the basic connectives is sufficient in principle for TS language in terms of reasoning requirements about SWN.

3.5.2 Reduction Rules and Binding Priorities

This section describes reduction rules of workflow formulas for AND, OR, and XOR connectives. Influenced by the work in [154], the importance of these rules is to escalate efficiency of compliance checking algorithms to be presented in Chapter 5. The original purpose of the work in [154] is to apply inference rules to the workflow verification which is used to detect workflow anomalies. However, since workflow formulas are used to reduce the number of executions by ignoring some invalid ones, the complete enumeration of all possible truth assignments is not required.

In this essence, the concept of Constrained Truth Table (CTT) is adopted from [154] by excluding invalid assignments and ignoring false rows as follows (see Tables 3.2, 3.3, and 3.4). Note that ϕ and ψ are workflow formulas and variables in ϕ do not appear in ψ.

Rule 1: For ϕ AND ψ, ϕ and ψ must both be true or false.

Rule 2: For ϕ XOR ψ, ϕ and ψ cannot both be true.

Rule 3: The results of the evaluation of a formula cannot be false.

Here, the application of reduction rules is explained. According to Rule 3, for each row, any assignment that leads the formula to be false is ignored, and the remaining trues indicate possible executions. The true marking for each row indicates the presence of services. For example, only one from several services is chosen when XOR connective is presented in the formula. For instance, all possible assignments of a formula A XOR B XOR C are 100, 010, and 001.

It is very important for a workflow owner to verify that workflow formulas contain at least one possible execution. Specifically, there must be at least one row that evaluates a formula to true in order to prevent any self-conflict that leads the workflow execution into a deadlock state. This verification is outside the scope of this chapter and can be found in [155].

Table 3.2: Constrained Truth Table for ϕ AND ψ

ϕ	**ψ**	**ϕ AND ψ**
1	1	1

Table 3.3: Constrained Truth Table for ϕ XOR ψ

ϕ	**ψ**	**ϕ XOR ψ**
0	1	1
1	0	1

Table 3.4: Constrained Truth Table for ϕ OR ψ

ϕ	**ψ**	**ϕ OR ψ**
0	1	1
1	0	1
1	1	1

The following convention specifies binding priorities on the workflow connectives.

Convention 3.1 (Binding Priorities): The priority of our workflow connectives comes in order of AND, XOR and OR, respectively.

3.6 Requirement Specification Formalisation for SWN

This section describes how TS language is formulated in three modes: Henceforth Path Trust (HPT), Backward Path Trust (BPT), and Existence Trust (ET). These modes are important to indicate where and to which direction TS formulas are applied. HPT and BPT address the direction for preceding and succeeding services, respectively, while ET aims at any point of a workflow, possibly in both directions (see Figure 3.2). Here, two notions of originating and imposing service are revisited. The originating service is the starting service to which a TS formula is applied while the imposing service is the service that imposes a formula. According to the definitions of BTP and HTP, the originating and imposing services must be the same, while this is not necessary in ET. The formal definitions of these modes are given below.

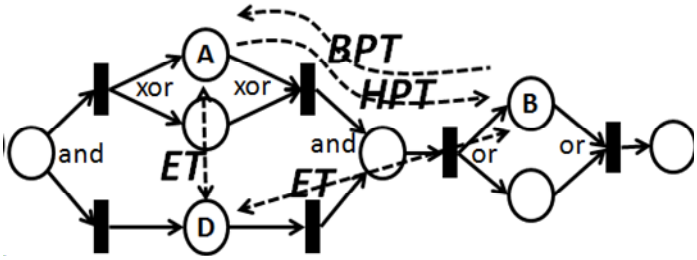


Figure 3.2: Workflow Example with HPT, BPT and ET

Def. 3.2: Let X be an originating point where a TS formula applies and Y be a target which can be SWN, workflow owner, services, or tasks. Trust Model Computation (TMC) is a vector where its elements (tmc_i) indicate the computational trust models used to compute trust values. Path trust is classified into two directions:

1. HPT:

$$\alpha_{TMC}^{X \to Y} = X \succcurlyeq_{TMC=[tmc_1,\ldots,tmc_n]} Y$$

where $\alpha_{TMC}^{X \to Y}$ denotes the value resulting from binary relation $\succcurlyeq$, which indicates the application of TS formulas from X to Y. TMC is a container for computational models for trust of X indicated by tmc_i. $\alpha_{TMC}^{X \to Y}$ is valid if a TS formula addresses the Henceforth direction from X to the subsequent Y along a workflow path.

2. BPT:

$$\beta_{TMC}^{X \to Y} = X \succ_{TMC=[tmc_1,\ldots,tmc_n]} Y$$

where $\beta_{TMC}^{X \to Y}$ denotes the value resulting from binary relation $\succ$, which indicates the application of TS formulas from X to Y. TMC is the same as described above. $\beta_{TMC}^{X \to Y}$ is valid if a TS formula addresses the Backward direction from X to the preceding Y along a workflow path.

Def. 3.3: ET: Let the terms X, Y and TMC be as defined in **Def. 3.2**. The ET definition is given below:

1. ET:

$$E_{TMC}^{X \to Y} = X \trianglerighteq_{TMC=[tmc_1,\ldots,tmc_n]} Y$$

where $E_{TMC}^{X \to Y}$ denotes the value resulting from binary relation $\trianglerighteq$, which indicates the application of TS formulas from X to Y. Because there is no restriction where an imposing service is the same as an originating service, ET is a looser version of Path Trust. However, it is considered more expressive as being able to apply TS formulas at any point in SWN. ET is important when the presence of one service might affect the willingness of others in participating a workflow, regardless of connecting paths.

To understand the distinction of these modes, consider the example of an SWN in Figure 3.2 that conveys how these three modes are applied. In the example, suppose A is an imposing service. The values of HPT and BPT are denoted by $\alpha_{TMC}^{A \to B}$ and $\beta_{TMC}^{B \to A}$, and the two ET values are $E_{TMC}^{A \to D}$ and $E_{TMC}^{B \to D}$.

3.6.1 Vector TMC

From the definitions above, TMC is a vector space referring to the container of the use of trust models which can be selected among several such as reputation rating, credential, provenance information, etc. Since the compliance of requirements produces yes or no answer for each requirement, it must be input to the selected trust models where the final trust value can be determined. At this stage, the metrics are left loose for these different characteristics, merely observing that a vector (rather than a single value) is needed to capture them. In practice, common ontology can be used for the identification of the models that must be agreed among interoperating parties as to be interpreted in the same way.

3.6.2 Trust as a Bounded Value [-1.. 1]

Since a trust value might vary from different perspectives, and some of the models are not designed to be computable, **Def. 3.4** below demonstrates the conversion to address this problem. The results from functions are used as inputs to the example of calculation in Section 3.8.

Def. 3.4: Let the terms X, Y and TMC be as defied in **Def. 3.2**. The Conversion Function is

$$F: [tmc_1, \dots, tmc_n] \rightarrow [a_1, \dots, a_n], \text{ where } a_1, \dots, a_n \in [-1..1], \text{ such that}$$
$$F([tmc_1, \dots, tmc_n]) = [f_1(tmc_1), \dots, f_n(tmc_n)], \text{ and } f_i(tmc_i) \in [-1..1].$$

Below, the use of this interval -1 to 1 is justified. Some trust schemes use discreet and continuous values to measure degree of trust. For example, the work in [156] approaches trust of a certificate in a discrete value [0,1], while the others ([157], [158], [159], and [160]) derive trust into a continuous value [0..1]. After analysing the various metrics, it can be concluded that there is no universal metric generic for all applications. The reason behind trust being reduced to a single numeric value is that trust is a relative factor, so it makes sense to use some unit-less ratio value normalised into the interval $[-1..1]$ instead of using an integer or unbounded values [161]. The intuitive explanation of this range may be from complete distrust to full trust [156]. A value 0 means that one entity has no relationship in another where the conversion function is able to address this knowledge. However, this interval merely represents comparative values and has no strong semantic

meaning in itself. The discussion of the benefits and drawbacks of using this trust value is provided in [76].

Consequently, the result of this function is the value reflecting trust-based decision for service acquisition from a workflow owner and workflow participation from services. Thus, the values of $\alpha_{TMC}^{X\to Y}$, $\beta_{TMC}^{X\to Y}$, and $E_{TMC}^{X\to Y}$ are in the form of $[f_1(tmc_1), \ldots, f_n(tmc_n)]$. We note that how the function is defined and used is context-specific and subjective to a particular domain.

3.6.3 TS Algebra

This section focuses on how TS formulas are constructed by using algebraic operators and propositions. Three types of operators are presented: Composition, Path, and Direction. Composite formulas are built up based on the association between tasks and services; Path formulas based on CTL describes the sequence of tasks and branches in SWN and Direction formulas indicate the direction to which TS formulas apply. We note that TS language is not devised as an alternative logic; but rather being seen as an extension and application of the existing logics to address exclusive criteria (R1-R8) in a service workflow. Throughout this thesis, place (p) and transition (t) from the notions of Petri Net represent services and tasks in an SWN. The algebraic operators are presented in the Backus–Naur Form (BNF) form below.

1. Composite Formulas

$$S ::= \ !S \mid \mathcal{F}E_tZ \mid \mathcal{P}E_tZ \mid \mathcal{F}A_tZ \mid \mathcal{P}A_tZ \mid (S \,\&\, S) \mid (S \,||\, S) \text{ (Quantifier part)}$$
$$Z ::= \ \varepsilon \mid (s, t, o, A) \mid !Z \mid (Z \sqcap Z) \mid (Z \sqcup Z) \mid (Z \boxplus Z) \text{ (Property part)}$$

2. Path Formulas

$$R ::= \ \top \mid \bot \mid S \mid {\sim}R \mid (R \wedge R) \mid (R \vee R) \mid \exists_t \odot R \mid \exists_t \Diamond R \mid$$
$$\exists_t \Box R \mid \exists_t [R \Cup R] \mid \exists_t [R \cup R] \mid \forall_t \odot R \mid \forall_t \Diamond R \mid \forall_t \Box R \mid \forall_t [R \Cup R] \mid \forall_t [R \cup R]$$

3. Direction Formulas

$$W ::= \ \top \mid \bot \mid \mathcal{H}R \mid \mathcal{B}R \mid {\sim}W \mid (W \wedge W) \mid (W \vee W)$$

Any TS formula uses an inductive definition of Composite, Path and Direction; the definition of each depends upon the others. Below, the detailed explanation of each operator is provided.

Composite Operators

Composite operators are invented based on FOL to capture the association between services and tasks. Service composed for each task can be activated in a number of ways restricted by workflow formulas (AND, OR, and XOR). For example, a formula X OR Y has three possible executions with the presence of (1) X, (2) Y, or (3) X and Y. The essence of Composite operators is to specify properties of involved services for a task. For instance, suppose that α is a Composite formula. If X OR Y presents a valid execution, we must have properties of X satisfied by α, properties of Y satisfied by α, and properties of both X and Y satisfied by α. The definition of each operator is described as follows.

$\mathcal{F}$: *Forward* operator addresses properties of services immediately connected in the forward direction (service-split) from a task t indicated by E_t or A_t.

$\mathcal{P}$: *Previous* operator addresses properties of services immediately connected in the previous direction (service-join) from a task t indicated by E_t or A_t.

E_t: *Composite For Some* operator restricts that properties of some services immediately connected to/from a task t must be satisfied.

A_t: *Composite For All* operator restricts that properties of all services immediately connected to/from a task t must be satisfied.

&: *Weak Composite Conjunction* operator is used for property composition of services agglomerated for each execution. It is not restricted to be preceded by the same Composite quantifier operator (E_t or A_t). For example, $\mathcal{P}E_tS$ & $\mathcal{F}A_tS$ indicates that for each execution the properties of the involved services must satisfy $\mathcal{P}E_tS$ and $\mathcal{F}A_tS$.

||: *Weak Composite Disjunction* operator indicates either property separated by || must be satisfied. Similar to &, || is not restricted to be preceded by the same Composite quantifier operator (E_t or A_t). For example, $\mathcal{P}E_tS$ || $\mathcal{F}A_tS$ indicates that for each execution the properties of the involved services must satisfy $\mathcal{P}E_tS$ or $\mathcal{F}A_tS$.

ε: *Null* represents an empty element meaning that no properties of services are required. This operator is important for addressing Enforcement of Sequence (R5).

$(\mathrm{s}, \mathrm{t}, \mathrm{o}, \mathrm{A})$: *Atomic proposition* represents an atomic proposition of Z where s, t and o are service name, type, and owner respectively. A is a set of service attributes. For example, it can be used to address a statement saying that services with CA certificate are trusted ($\mathrm{A} = \{\mathrm{CA}\}$). Note that s, t, and o can be null (ε) and A can be empty, which means that no properties are required. Even though these internal elements are limited in 4 elements, this is not restricted as we allow flexibility for further implementation and extension.

⊓: *Strong Composite Conjunction* operator is used for properties composition that must be satisfied by services immediately connected to/from a task. Composite quantifier operator cannot be distributed into this operator, for example, $\mathcal{P}\mathrm{A}_\mathrm{t}(\mathrm{P} \sqcap \mathrm{Q}) \equiv \mathcal{P}\mathrm{A}_\mathrm{t}\mathrm{P} \sqcap \mathcal{P}\mathrm{A}_\mathrm{t}\mathrm{Q}$ is invalid. This is considered stronger than &.

⊔: *Strong Composite Disjunction* operator indicates that one or both properties must be satisfied by services immediately connected to/from a task. Similar to ⊓, it must be preceded by the same Composite quantifier operator and considered stronger than ||.

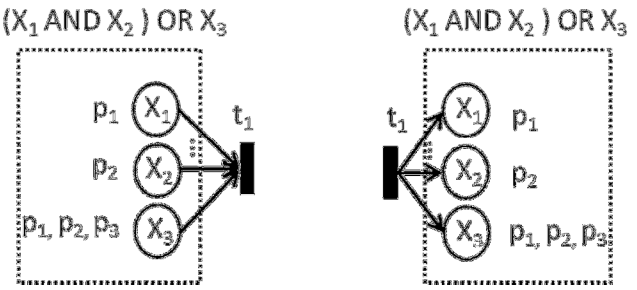
⊞: *Composite Exclusive Disjunction* operator indicates only one property is presented in each execution.

!: *Composite Negation* operator is used for the negation of a property. However, its presence can cause confusion and some inference rules are counter intuitive. For example, $!\,\mathrm{Z} \sqcap \mathrm{Z}$ is not self-conflict as its meaning suggests that there must be one service with Z, and another without Z. In other words, two properties (Z and not Z) must be presented in each execution. This expression cannot be applied by De Morgan's rules as $!\,(\mathrm{Z}_1 \sqcap \mathrm{Z}_2)$ is not equivalent to $!\,\mathrm{Z}_1 \sqcup !\,\mathrm{Z}_2$. Consequently, $!\,\mathrm{Z}$ and Z are considered different properties and the negation can be omitted without losing its meaning. For example, $!\,\mathrm{Z}_1 = (!\,\mathrm{s}, !\,\mathrm{t}, !\,\mathrm{o}, !\,\mathrm{A})$ and $\mathrm{Z}_1 = (\mathrm{s}, \mathrm{t}, \mathrm{o}, \mathrm{A})$ are considered different properties that can be represented by Z_1 and Z_2, instead of $!\,\mathrm{Z}_1$ and Z_1.

To understand the application of these operators, consider the example in Figure 3.3. Suppose that there are three services X_1, X_2, and X_3 with properties p_1, p_2, and p_1 and p_2 and p_3, respectively. The workflow formula used to restrict service activation is

$(X_1 \text{ AND } X_2) \text{ OR } X_3$, such that there are three possible ways of executions with the presence of (1) X_1 and X_2, (2) X_3, and (3) X_1 and $X_2$and X_3. $\mathcal{P}$ indicates service-join presented in Figure 3.3(a), while $\mathcal{F}$ indicates service-split from a task presented in Figure 3.3(b).

To see the difference between E_t and A_t, consider a formula $\mathcal{P}E_{t_1}p_1$. This formula is satisfied since there will always be some services with property p_1 for every execution. However, $\mathcal{P}A_{t_1}p_1$ is partially satisfied as only the second execution satisfies the formula. The first and third executions are invalid because A_{t_1} indicates all services for each execution must have property p_1. In this case, X_2does not have this property. For another example, the formula $\mathcal{P}A_{t_1}(p_1 \sqcap p_3)$ is partially satisfied as only the second and third executions are valid. $\mathcal{P}A_{t_1}(p_1 \sqcup p_2)$ is satisfied for all executions. $\mathcal{P}A_{t_1}(p_3 \boxplus !\,p_3)$ is partially satisfied as only the third execution contains properties p_3 and $!\,p_3$. $\mathcal{P}A_{t_1}p_3 \,\&\, \mathcal{P}E_{t_1}p_2$ is partially satisfied since only the second execution satisfies both $\mathcal{P}A_{t_1}p_3$ and $\mathcal{P}E_{t_1}p_2$, while $\mathcal{P}A_{t_1}p_3 \,||\, \mathcal{P}E_{t_1}p_2$ is satisfied since the first and the third executions satisfy $\mathcal{P}E_{t_1}p_2$, and the second execution satisfies both $\mathcal{P}A_{t_1}p_3$ and $\mathcal{P}E_{t_1}p_2$.



(a) (b)

Figure 3.3: Examples of (a) Service-join and (b) Service-split

Path Formulas

Similar to CTL that applies to verify properties of Finite State Machine (FSM), Path operators are invented to capture the sequence of tasks and paths in an SWN. Tasks in an SWN are considered FSM states regardless of services as they are adhered to each task where the compliance is evaluated by Composite formulas. The Path operators are described below.

$\odot$: *Next* operator specifies the immediately connected task along selected paths.

$\Diamond$: *Future* operator specifies that tasks along selected paths must be eventually reached.

$\Box$: *Global* operator specifies all tasks along selected paths.

$R_1 \Cup R_2$: *Strong Until* operator specifies that R_1 must hold until the presence of R_2 and R_2 must hold in the future.

$R_1 \cup R_2$: *Weak Until* operator is just like $\Cup$, except R_2 is not required to hold.

$\exists_t$: *For Some Path* operator specifies some paths among a set of connected tasks through a task t. If t is omitted, there is no condition on paths specific to tasks and all paths must be considered.

$\forall_t$: *For All Path* operator specifies all paths through a task t. If t is omitted, all paths must be considered.

(R $\wedge$ R, R $\vee$ R, and $\sim$R are similar to the definitions in CTL.)

Direction Formulas

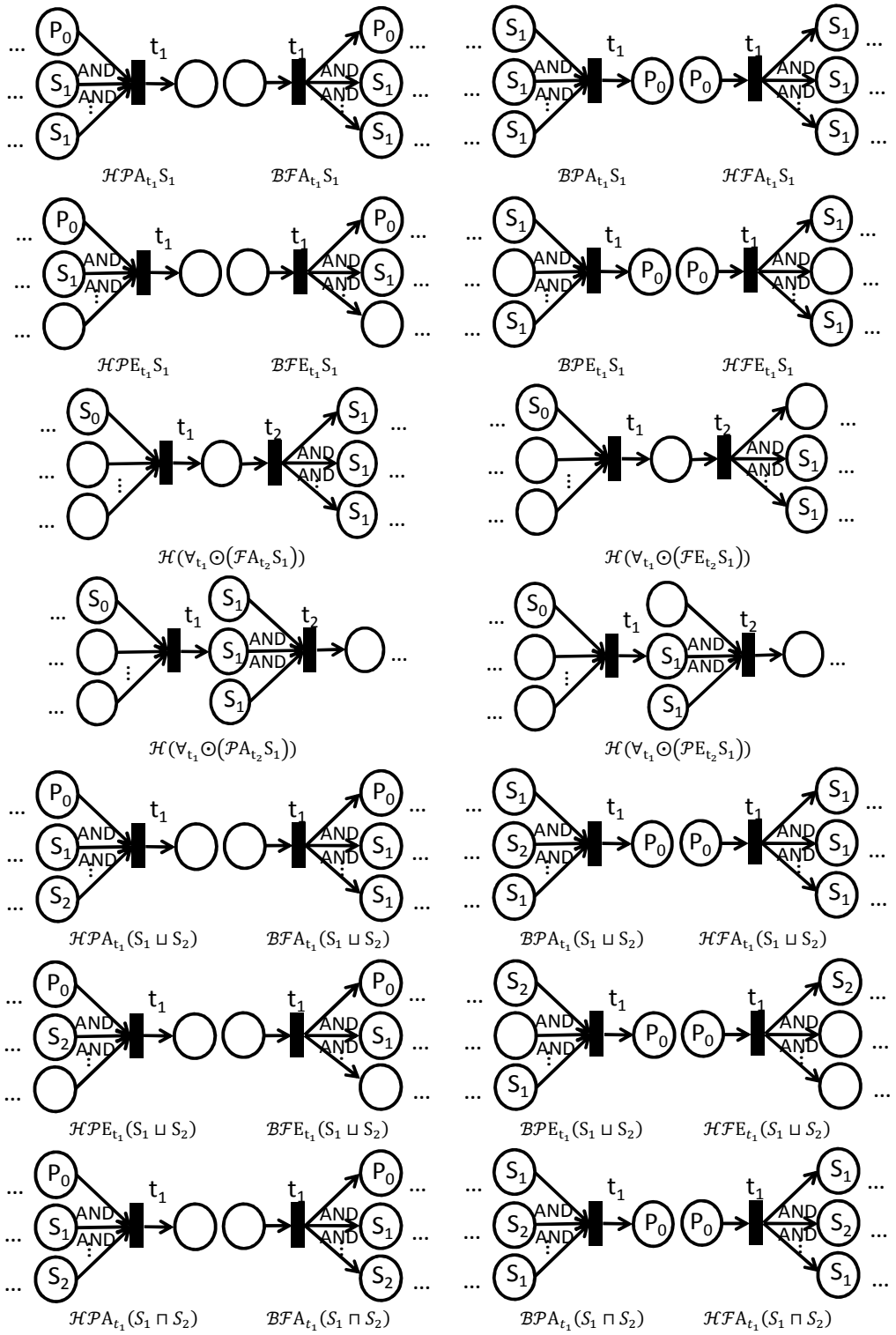
$\mathcal{H}$: *Henceforth* operator specifies the forward direction along a workflow path. This operator is important to construct HPT.

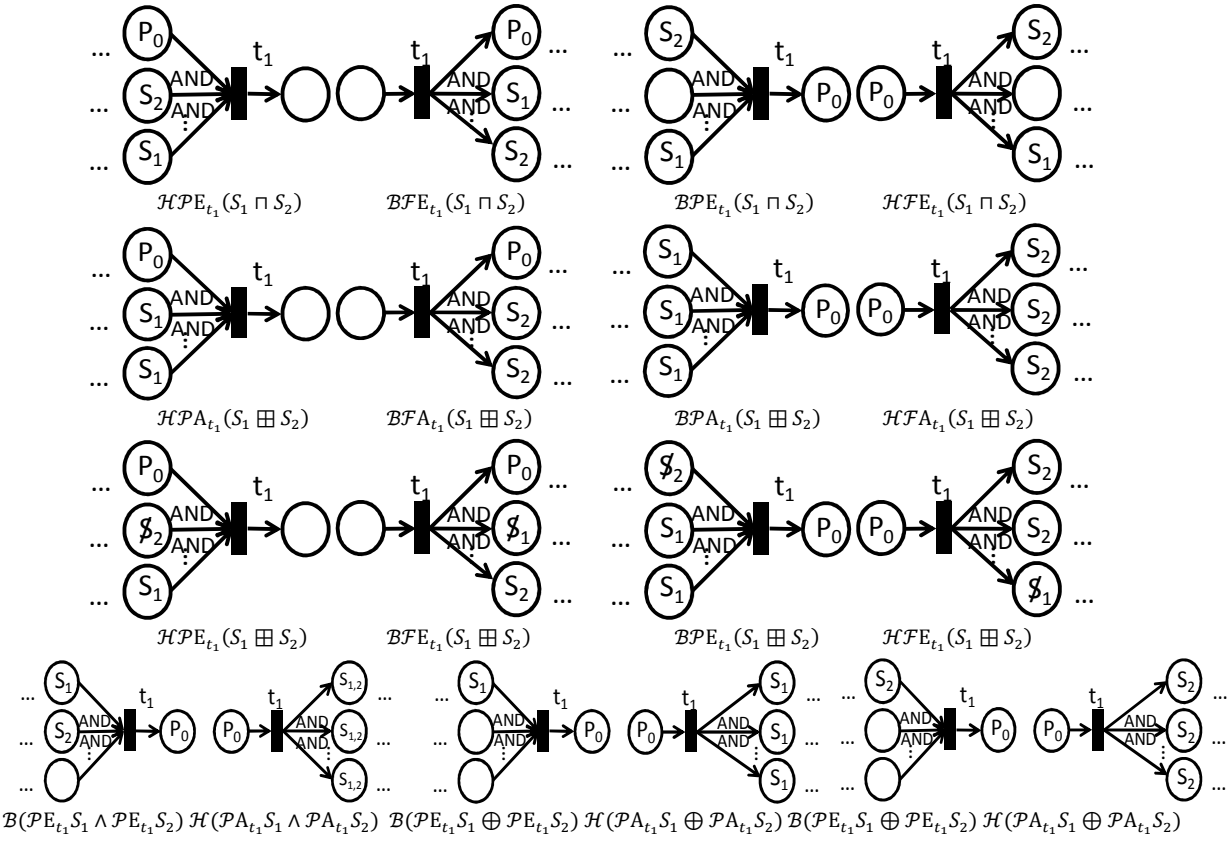
$\mathcal{B}$: *Backward* operator specifies the backward direction along a workflow path. This operator is important to construct BPT.

(W $\wedge$ W, W $\vee$ W, and $\sim$W are similar to the definitions in propositional logic.)

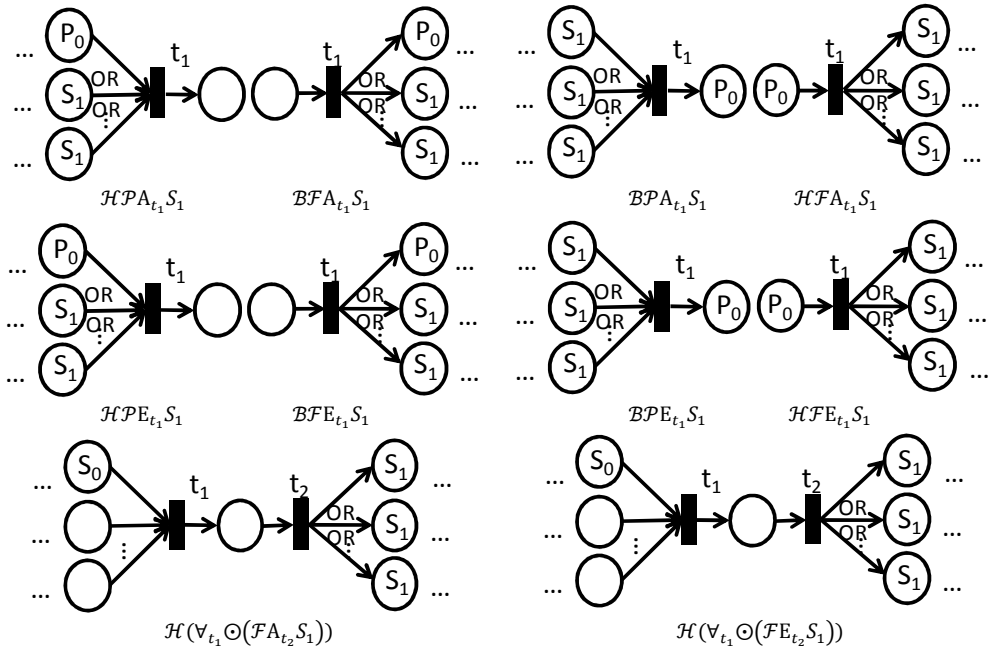
In the case of ET, since it has no condition on an imposing and originating service, this mode is able to incorporate all types of operators. For example, an imposing service X requires that an originating service Y must be immediately connected by the services with property p. This can be expressed as $\mathcal{HF}\forall(p)$, where $E_{TMC}^{Y\rightarrow\{\text{service conntected from Y}\}}$ is the value of X placing on a part of SWN which is Y and the services connected from Y. (The full graphical explanation of all TS operators with each type of workflow connective is given in Figure 3.4)

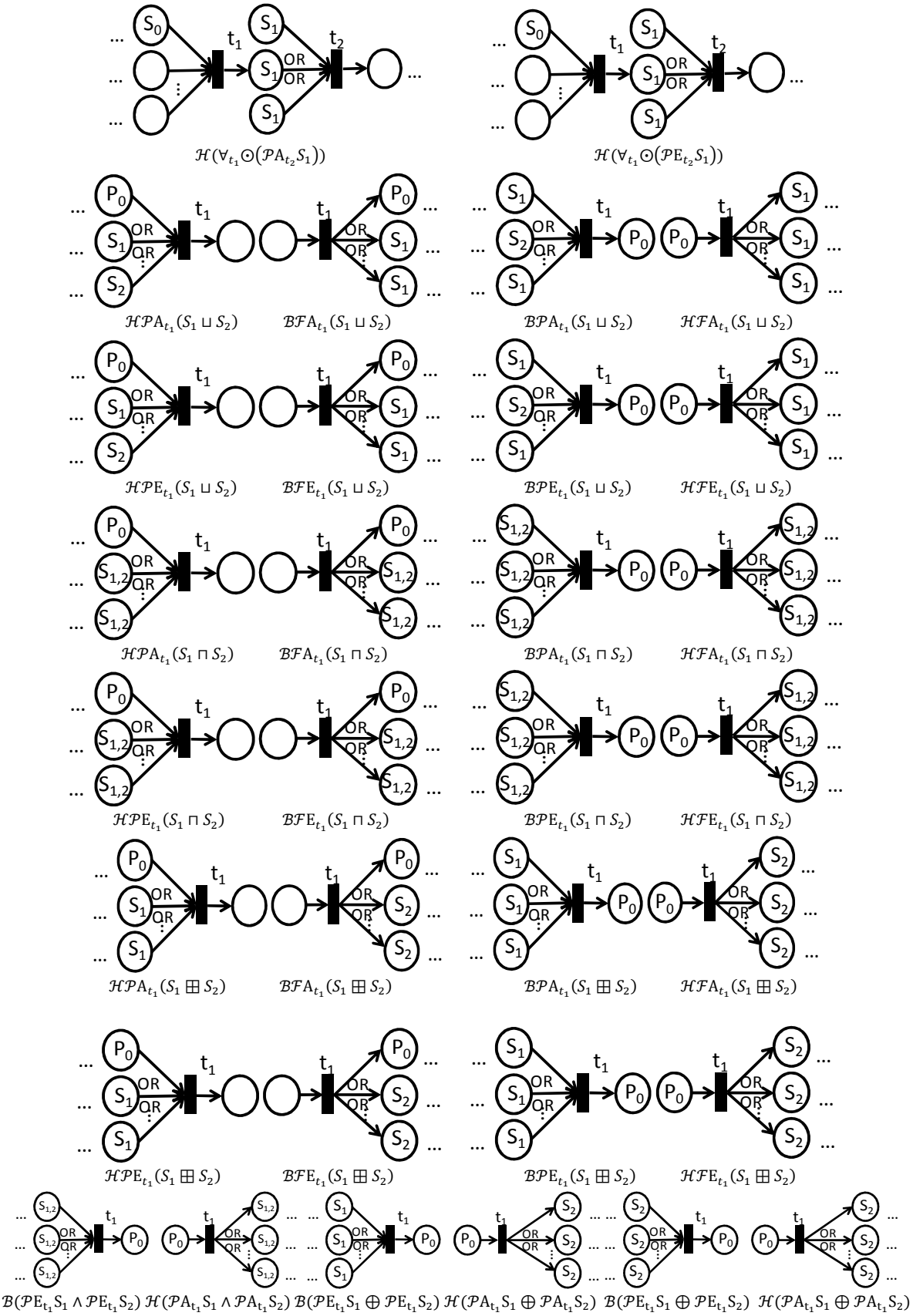
AND Connective



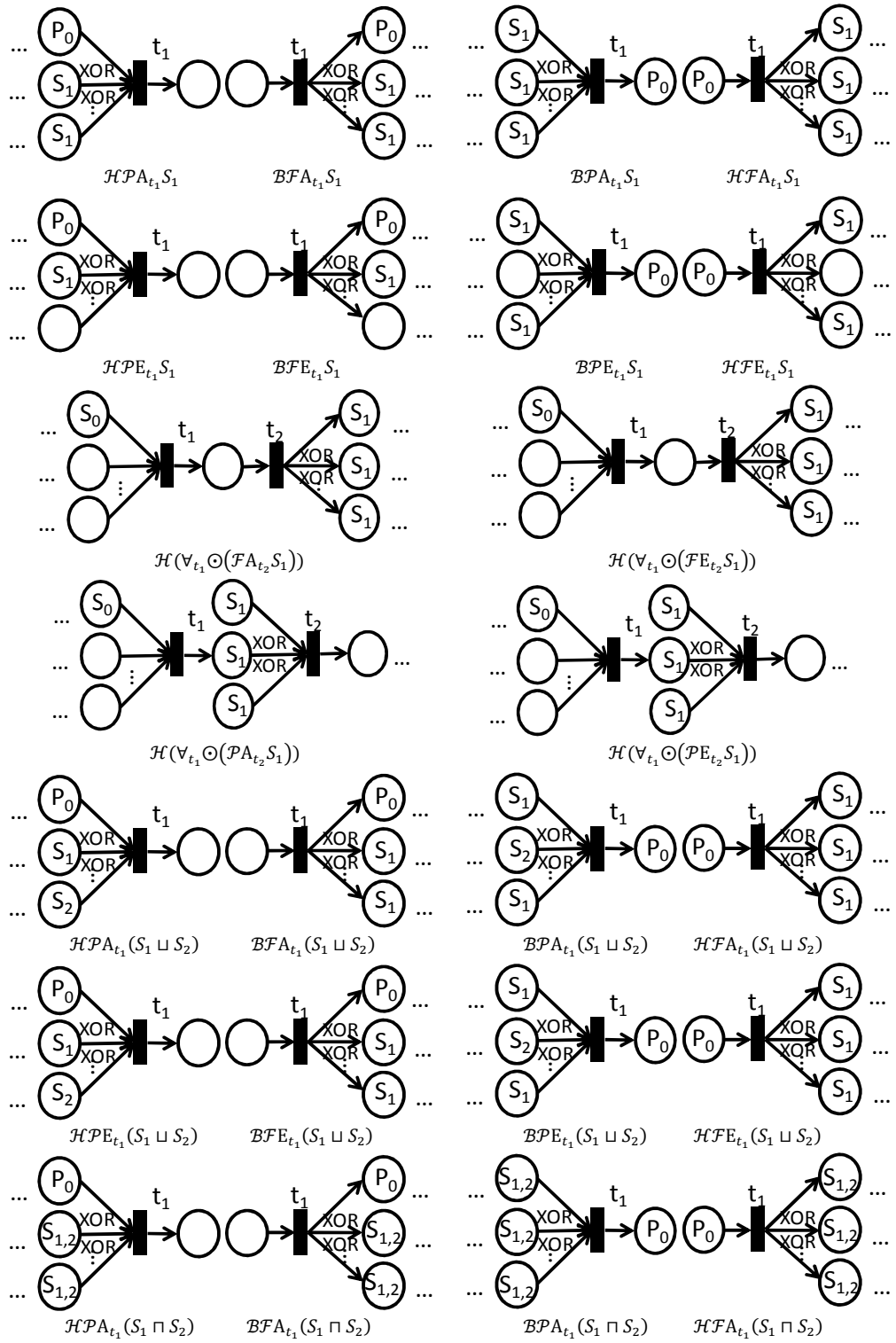


OR Connective




$\mathcal{H}(\forall_{t_1} \odot (\mathcal{PA}_{t_2} S_1))$
$\mathcal{H}(\forall_{t_1} \odot (\mathcal{PE}_{t_2} S_1))$
$\mathcal{HPA}_{t_1}(S_1 \sqcup S_2)$
$\mathcal{BFA}_{t_1}(S_1 \sqcup S_2)$
$\mathcal{BPA}_{t_1}(S_1 \sqcup S_2)$
$\mathcal{HFA}_{t_1}(S_1 \sqcup S_2)$
$\mathcal{HPE}_{t_1}(S_1 \sqcup S_2)$
$\mathcal{BFE}_{t_1}(S_1 \sqcup S_2)$
$\mathcal{BPE}_{t_1}(S_1 \sqcup S_2)$
$\mathcal{HFE}_{t_1}(S_1 \sqcup S_2)$
$\mathcal{HPA}_{t_1}(S_1 \sqcap S_2)$
$\mathcal{BFA}_{t_1}(S_1 \sqcap S_2)$
$\mathcal{BPA}_{t_1}(S_1 \sqcap S_2)$
$\mathcal{HFA}_{t_1}(S_1 \sqcap S_2)$
$\mathcal{HPE}_{t_1}(S_1 \sqcap S_2)$
$\mathcal{BFE}_{t_1}(S_1 \sqcap S_2)$
$\mathcal{BPE}_{t_1}(S_1 \sqcap S_2)$
$\mathcal{HFE}_{t_1}(S_1 \sqcap S_2)$
$\mathcal{HPA}_{t_1}(S_1 \boxplus S_2)$
$\mathcal{BFA}_{t_1}(S_1 \boxplus S_2)$
$\mathcal{BPA}_{t_1}(S_1 \boxplus S_2)$
$\mathcal{HFA}_{t_1}(S_1 \boxplus S_2)$
$\mathcal{HPE}_{t_1}(S_1 \boxplus S_2)$
$\mathcal{BFE}_{t_1}(S_1 \boxplus S_2)$
$\mathcal{BPE}_{t_1}(S_1 \boxplus S_2)$
$\mathcal{HFE}_{t_1}(S_1 \boxplus S_2)$
$\mathcal{B}(\mathcal{PE}_{t_1} S_1 \wedge \mathcal{PE}_{t_1} S_2)$
$\mathcal{H}(\mathcal{PA}_{t_1} S_1 \wedge \mathcal{PA}_{t_1} S_2)$
$\mathcal{B}(\mathcal{PE}_{t_1} S_1 \oplus \mathcal{PE}_{t_1} S_2)$
$\mathcal{H}(\mathcal{PA}_{t_1} S_1 \oplus \mathcal{PA}_{t_1} S_2)$
$\mathcal{B}(\mathcal{PE}_{t_1} S_1 \oplus \mathcal{PE}_{t_1} S_2)$
$\mathcal{H}(\mathcal{PA}_{t_1} S_1 \oplus \mathcal{PA}_{t_1} S_2)$

XOR Connective



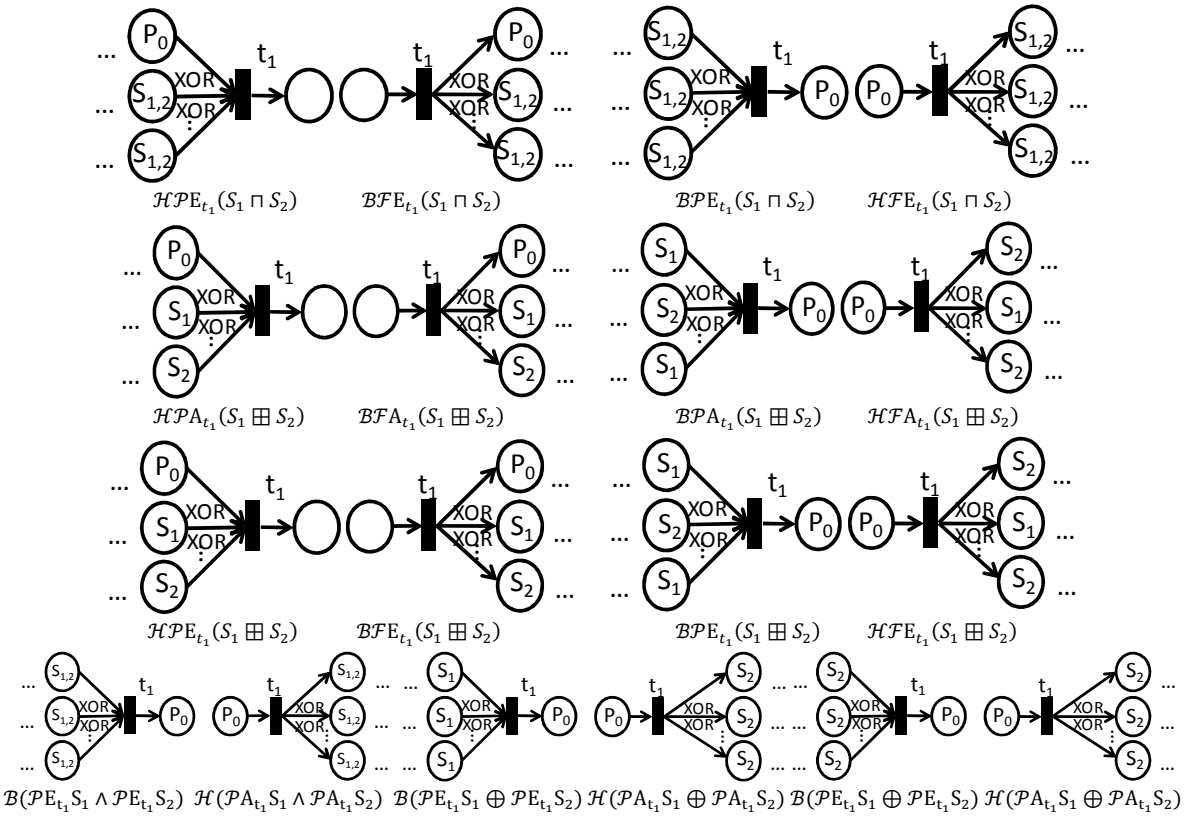


Figure 3.4: Graphical Illustration Examples of Algebraic Operators

3.6.4 Satisfiability Relations

It is essential to extend the specification with a formal analysis of satisfiability that is used to determine whether a TS formula is satisfied by an SWN.

Def. 3.5: Let ω be a TS formula and $\mathcal{M}$ be an SWN. Whether a TS formula satisfies, partially satisfies, or unsatisfies $\mathcal{M}$ is defined by the following relations.

1. $\mathcal{M} \vDash \omega$ is called satisfied, when $\mathcal{M}$ satisfies ω;
2. $\mathcal{M} \vDash' \omega$ is called partially satisfied, when $\mathcal{M}' \vDash \omega$ and $\mathcal{M}' \subseteq \mathcal{M}$; and
3. $\mathcal{M} \nvDash \omega$ is called unsatisfied, when $\mathcal{M}$ does not satisfy ω.

The subset (⊆) of an SWN is defined as all possible sets of services and tasks that can complete the end-to-end process of SWN. In other words, each SWN subset has one starting point and one ending points and there must be a path reachable from one to another.

3.7 The Application Example

In this section, some practical usages of TS formulas are demonstrated in the travel agent workflow scenario (the full scenario will be described later in Chapter 6 and for evaluation in Chapter 7). Here, the satisfiability results from each formula are described narratively. For semantics of reasoning about TS formula and each workflow connective, please find more detail in Chapter 4.

3.7.1 Travel Agent Workflow Scenario

Considering an SWN of a travel agent business workflow, tasks are created and interconnected as follows (see Figure 3.5).

1. *Searching* includes flight, hotel, and car services.
2. *Location* provides a selected set of flight, hotel, and car rental, with the location information of airport, hotel, and car pick-up address, respectively.
3. *HotelRating* provides rating information according to the selected hotels.
4. *Reservation* will be activated after one, or multiple of, flight, hotel, and car selection from *Searching*.
5. *Booking* is activated once the reservation is made. Alternatively, reservation can be cancelled based on a customer decision.
6. *Restaurants* offer information, promotion, and reservation based on the customer's destination, time of travelling, and the current collection of restaurant services.
7. *Payment* is responsible for financial activities which will be activated once *Booking* is successfully executed.
8. *TravellingDocs* produces necessary travelling documents including vouchers, e-tickets, receipts, and acknowledgement messages.

The remaining work to complete the SWN is the process of service acquisition for each task. Services are dynamically selected and the logic of service composition is formulated by workflow formulas. Let us assume that the workflow incorporates with external services including airline, hotel, car rental, location, restaurants, and payment services and the

services executing *TravellingDocs* is provided internally by the workflow owner (see the list in Table 3.5).

3.7.2 Applying TS Formulas

Five example cases of the application of TS formulas are given below and summarised in Table 3.6. For simplicity, there is only one model in vector TMC producing binary result whether a TS formula is complied (1) or not (-1).

CASE 1: *Certified Services.* Consider service S_{XP} at *FlightBooking* which processes transactions from any flight booking request. S_{XP} requires that all transactions will lead to *Payment* and services performing the payment are certified by VISA or Master.

A TS formula $\mathcal{H}\forall \Diamond \mathcal{P}A_{\text{Payment}}((\varepsilon, \varepsilon\, \varepsilon, \{C_{VISA}\}) \sqcup (\varepsilon, \varepsilon\, \varepsilon, \{C_{MASTER}\}))$ demonstrates the certificate requirement either VISA or Master as the properties for the services at *Payment* (which are S_{VS}, S_{MS}, and S_{PP}). It restricts that in *Henceforth* ($\mathcal{H}$) direction, all paths ($\forall \Diamond$) from *FlightBooking* will reach *Payment*, and the properties of the services at *Payment* must be C_{VISA} or C_{MASTER} . Because this formula satisfies only some paths (e.g., (Source) $\mathbf{S_{XP}}$(*FlightBooking*) → S_{WR} (*Restaurants*) →(Target) $\mathbf{S_{VS}}$(*Payment*)), it is *partially satisfied*. This formula falls into the HPT mode where the value is presented by $\alpha_{[1]}^{S_{XP}\to S_{VS}}$, $\alpha_{[1]}^{S_{XP}\to S_{MS}}$, and $\alpha_{[-1]}^{S_{XP}\to S_{PP}}$. The subscript $[1]$ and $[-1]$ indicate the compliance checking result of S_{XP} on S_{VS}, S_{MS}, and S_{PP}. Trust values can be determined from these results affecting participating decision of S_{XP} in this SWN. This case illustrates the concept of system trust defined in [43]. Association between Tasks and Services (R3) and Flow- and Task-oriented Specification (R5) are two criteria illustrated by this case.

CASE 2: *Information Provision.* The hotel service S_{AG} wants to protect secrecy of the information regarding a price promotion offered to its customers. In order to pass such information along workflow paths until reaching *Payment*, it requires that the services involved in the path must be operated by itself. This TS formula can be expressed as $\mathcal{H}\forall[\mathcal{F}A(\varepsilon, S_{AG}, \varepsilon, \emptyset) \sqcup \mathcal{P}A_{\text{Payment}}(\varepsilon, \varepsilon, \varepsilon, \emptyset)]$. Similar to Case 1, since it satisfies only some

paths (e.g. (Source) $\mathbf{S_{AG}}$ (*HotelSearching*) → S_{AG} (*HotelReservation*) → S_{AG} (*HotelBooking*) → $\mathbf{S_{VS}}$ (*Payment*)), it is *partially satisfied*. This formula falls into the HPT mode, and the value is presented by $\alpha_{[1]}^{S_{AG} \to \{\text{Intermediate Services to Payment}\}}$, indicating trust-based participation decision of S_{AG} in this SWN. From this example, TS formula is able to address Flow- and Task-oriented Specification (R4) and Protection of Workflow Data (R7).

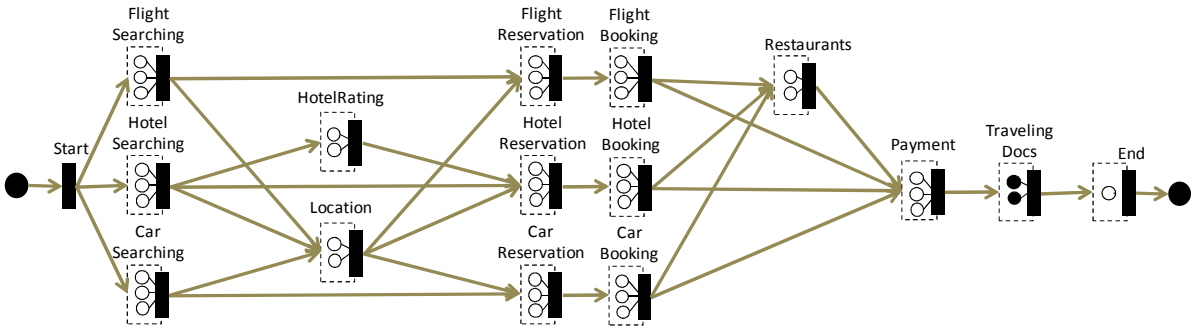


Figure 3.5: The Travel Agent Workflow

Table 3.5: The List of Services and Tasks Associated

Task	Service Associated	Workflow Formulas
FlightSearching, FlightReservation, FlightBooking	Expedia (S_{XP}), Priceline (S_{PL}), EazyJet (S_{EZ})	S_{XP} OR S_{PL}OR (S_{XP}AND S_{EZ})) OR (S_{PL} AND S_{EZ})
HotelSearching, HotelReservation, HotelBooking	Expedia (S_{XP}), Priceline (S_{PL}), Agoda (S_{AG})	S_{XP} OR S_{PL} OR S_{AG}
CarSearching, CarReservation, CarBooking	Expedia (S_{XP}), Priceline (S_{PL}), Avis (S_{AV})	S_{XP} OR S_{PL} OR S_{AV}
HotelRating	Agoda (S_{AG}), Booking (S_{BO})	S_{AG} OR S_{BO}
Location	GoogleMap (S_{GM}), BingMap (S_{BM})	S_{GM} XOR S_{BM}
Restaurants	WordRes (S_{WR}), Res1 (S_{R1})	S_{WR} OR (S_{WR} AND S_{R1})
Payment	Visa (S_{VS}), Master (S_{MS}), Paypal (S_{PP})	S_{VS} XOR S_{MS} XOR S_{PP}
TravellingDocs	$S_{TD1}$and S_{TD2}	S_{TD1} XOR S_{TD2}

Table 3.6: Examples of TS Formulas of Case 1 to 5.

Case	Source	Target	Trust Specification Formulas	Trust Modes (source→target)	Result
1	S_{XP}	S_{VS}, S_{MS}, S_{PP}	$\mathcal{H}\forall \Diamond \mathcal{P}A_{Payment}((\varepsilon, \varepsilon\ \varepsilon, \{C_{VISA}\}) \sqcup (\varepsilon, \varepsilon\ \varepsilon, \{C_{MASTER}\}))$	$\alpha_{[1]}^{S_{XP}\to S_{VS}}$, $\alpha_{[1]}^{S_{XP}\to S_{MS}}$, $\alpha_{[-1]}^{S_{XP}\to S_{PP}}$	Partially Satisfy
2	S_{AG}	SWN	$\forall[\mathcal{F}A(\varepsilon, S_{AG}, \varepsilon, \emptyset)\biguplus]$ $\mathcal{P}A_{Payment}(\varepsilon, \varepsilon, \varepsilon, \emptyset)$	$\alpha_{[1]}^{S_{AG}\to SWN}$	Partially Satisfy
3	S_{PP}	SWN	$\mathcal{B}(\forall_{*Booking}\odot\mathcal{P}A(\varepsilon, \varepsilon, \varepsilon, \emptyset) \vee \forall_{Restaurant}\odot\mathcal{P}A(\varepsilon, \varepsilon, \varepsilon, \emptyset)) \wedge \mathcal{B}\forall \Diamond \mathcal{P}A\big((\varepsilon, S_{XP}, \varepsilon, \emptyset) \sqcup (\varepsilon, S_{PL}, \varepsilon, \emptyset)\big)$	$\beta_{[1]}^{S_{PP}\to SWN}$	Partially Satisfy
4	Owner	SWN	$\mathcal{H}\forall_{Restaurant} \Diamond \mathcal{P}A_{Restaurant}(\varepsilon, \varepsilon, !\,\mathrm{HotChilly}, \emptyset)$	$E_{[1]}^{Owner\to S_{WR}}$, $E_{[1]}^{Owner\to S_{R1}}$	Satisfy
5	S_{PL}	S_{XP}, S_{PL}, S_{EZ}	$\mathcal{B}\exists \Diamond \mathcal{P}A_{FlightSearching}!\,(\varepsilon, \varepsilon, XP, \emptyset)$	$\alpha_{[-1]}^{S_{PL}\to S_{XP}}$, $\alpha_{[1]}^{S_{HI}\to S_{PL}}$, $\alpha_{[1]}^{S_{HI}\to S_{EZ}}$	Partially Satisfy

CASE 3: *Service Provider*. The payment service S_{PP} is willing to provide the service right after **Booking* or *Restaurants* (**Booking* includes *FlightBooking, HotelBooking* and *CarBooking*), and requires that the transactions must be originated only from S_{XP} or S_{PL}. This can be expressed as $\mathcal{B}\left(\forall_{*Booking}\odot\mathcal{P}A(\varepsilon, \varepsilon, \varepsilon, \emptyset) \vee \forall_{Restaurant}\odot\mathcal{P}A(\varepsilon, \varepsilon, \varepsilon, \emptyset)\right) \wedge \mathcal{B}\forall \Diamond \mathcal{P}A\big((\varepsilon, S_{XP}, \varepsilon, \emptyset) \sqcup (\varepsilon, S_{PL}, \varepsilon, \emptyset)\big)$. The first term ensures that the tasks directly connected to S_{PP} must be **Booking* or *Restaurants*, while the latter restricts the flow must be originated from S_{XP} or S_{PL}. Since the formula satisfies only some paths (e.g., (Target) $\mathbf{S_{XP}}$ (*HotelBooking*)→(Source)$\mathbf{S_{PP}}$(*Payment*)), it is *partially satisfied*. This falls in the BPT mode, where the value of the target is $\beta_{[1]}^{S_{PP}\to\{*Bookings, Restaurant, S_{XP}, S_{PL}\}}$ indicating trust-based participation decision of S_{PP} in this SWN. The first term illustrates Enforcement of Sequence (R5) and the second illustrates Flow-oriented Property (R4).

The explanation of the remaining cases is discussed briefly, as the previous three describe how TS formulas can accommodate requirements.

CASE 4: *Existence Trust (ET)*. The workflow owner does not trust restaurant HotChilly, so he requires that this service is not part of the SWN. This requirement can be expressed by

$\mathcal{H}\forall_{Restaurant} \Diamond \mathcal{P}A_{Restaurant}(\varepsilon, \varepsilon, !HotChilly, \emptyset)$. Since there is no service HotChilly at *Restaurant*, this formula is *satisfed*.

CASE 5: *Trust of Partnership*. The flight service S_{PL} at *FlightReservation* and *FlightBooking* does not collaborate with S_{XP} at *FlightSearching* due to the possibility of price comparison. This can be expressed as $\mathcal{B}\exists \Diamond \mathcal{P}A_{FlightSearching}!(\varepsilon, \varepsilon, XP, \emptyset)$. Since there is a situation of both S_{XP} and S_{PL} presented in an execution, this formula is *partially satisfied*.

3.8 The Example of Calculation Method

In this section the example usage of the vector TMC is described for calculation, where each element indicates the result of a trust model. Because some models are not designed to support trust computation, such as yes or no, and the values from some models may not comparable to the others, the Conversion Function in **Def. 3.4** converts and adjusts those values into a uniform range between -1 to 1.

One example for this computation is to use the Euclidian distance calculation. It requires an imposing service to set the base of initial values in the vector, which is used to calculate the similarity between the imposing service and a target.

Def. 3.6: Let X and Y be two vectors, and the similarity value between X and Y is defined by

$$\mathrm{SIM}(\mathrm{X},\mathrm{Y}) = \frac{\mathrm{X}\cdot\mathrm{Y}}{\|\mathrm{X}\|\|\mathrm{Y}\|}, \text{ where}$$

1. $\mathrm{X}\cdot\mathrm{Y} = \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i y_i}$, where $\mathrm{x_i} \in \mathrm{X}$ and $\mathrm{y_i} \in \mathrm{Y}$
2. $\|\mathrm{X}\| = \sqrt{\mathrm{x_1^2} + \mathrm{x_2^2} + \cdots + \mathrm{x_n^2}}$, and $\|\mathrm{Y}\| = \sqrt{\mathrm{y_1^2} + \mathrm{y_2^2} + \cdots + \mathrm{y_n^2}}$.

This method is applied to Case 4 as the example. Suppose that the workflow owner employs three models, i.e., $\mathrm{TMC} = [\mathrm{tmc_1}, \mathrm{tmc_2}, \mathrm{tmc_3}]$ and sets the desired values $[0.2,0.7,0.5]$ that result from the Conversion Function $[\mathrm{f_1}(\mathrm{tmc_1}), \mathrm{f_2}(\mathrm{tmc_2}), \mathrm{f_3}(\mathrm{tmc_3})]$.

$$\mathrm{E}_{[1]}^{\mathrm{Owner}\rightarrow \mathrm{S_{WR}}} = [-0.1,0.3,0.1] \text{ and } \mathrm{E}_{[1]}^{\mathrm{Owner}\rightarrow \mathrm{S_{R1}}} = [0.3,0.5,0.4]$$

The similarity values are computed as

1. $\mathrm{SIM}(\mathrm{Owner}, \mathrm{S_{WR}}) = \frac{0.24}{0.88 \times 0.33} = 0.84$ and $\mathrm{SIM}(\mathrm{Owner}, \mathrm{S_{R1}}) = \frac{0.61}{0.88 \times 0.70} = 0.99$
2. The average value is $\frac{0.84+0.99}{2} = 0.92$

The average values affect the decision of the owner in service acquisition and composition.

3.9 Basic Analysis of TS Formulas

From the five cases above, some analysis is given below.

R1 (ILR): Obviously, TS language is a formal approach enabling ILR since it allows each service to uniformly express its own requirements to others.

R2 (SoD): Although not illustrated by the cases above, TS formulas can address this criterion. For example, in a financial auditing scenario, the annual financial statement must be audited by two different companies. In this situation, it is not necessary to precisely identify the specific companies, but instead have to make sure that they are not the same. The atomic proposition can be extended by introducing a dummy variable $\mathrm{d_1}$ as $\mathcal{BP}\mathrm{A_{audit}}((\mathrm{d_1}, \varepsilon, , \varepsilon, \{\mathrm{audit}\}) \sqcap ((!\,\mathrm{d_1}, \varepsilon, , \varepsilon, \{\mathrm{audit}\}))$. It means that two different services must be present to execute the task. Some of the remaining criteria (R3-R5, and R7) have been explicitly discussed along the way in the cases above.

Mutual Relationship Value: It is important to note that trust relationship is subjective. The value of a service from one might be different from others. In many cases, this relationship exists only in one direction. For example, if X trusts Y, it is not necessary for Y to trust X. However, from the computation point of view, the lack of a relationship established in another direction does not imply that there is no relationship value. According to Conversion Function, if no relationship is established, the function will return 0 as the default value signifying no relationship between two entities.

Direct, Indirect, and Existence Trust: As stated, the interaction along a workflow path is examined in three ways: direct, indirect, and existence. From the above examples, we can

see that TS formulas realises these characteristics through the combination of the Composite, Path, and Direction formulas. The concept of direct trust is illustrated in Case 1 and 5, indirect trust in Case 2, Existence Trust in Case 4, while the remaining Case 3 addresses both direct and indirect.

3.10 Chapter Discussion

Dynamic workflow offers flexibility of interoperation, by allowing previously unknown services to dynamically participate as long as their properties comply with others' requirements. This is very important in open environments because unpredictable emergence or termination of services might result in changes in a workflow. Without the formal TS language and SWN, which are two fundamental elements for automatic compliance checking, it will be very difficult to the success of service workflow interoperation. The development of TS language is used to underline dynamic requirements by allowing a workflow owner and participating services the ability to specify their requirements in the formal and uniformed way. It is developed with three modes and algebraic operators are invented to capture all criteria needed for requirement specification languages in service workflows. With automatic compliance checking, it will lead to the success of secure workflow interoperation. The following subsections discuss some interesting issues in this area.

Architecture Description Languages

The concept of TS language is analogous to the area of software architecture specification, in that it employs a formal language to specify some desired properties of a system. The most recognised Architecture Description Languages (ADLs) for architecture specifications are provided in [162]. These ideas are adopted in our context, but approach a different problem domain. While the purpose of these languages is to specify and control behaviours to decrease the amount of errors in complex software systems at the architectural level (such as deadlock freeness, mutual exclusion and correctness verification), the use of TS language aims at specifying properties desired by a service to evaluate trust of other

services or a workflow. Another work of a formal specification is provided in [78]. However, it is limited to be inside a VOs domain. With the limited types of connectives, this work can only be used to verify properties over a single path.

Autonomy

In distributed environments, autonomy can be preserved as the owner retains full authority within the domain of ownership for managing and selecting services. However, he has less control over the internal activities of participating services as autonomy is also preserved at the service side. Each service can specify arbitrary requirements and make a judgment regarding workflow participation. Within a domain of ownership, a service is made available without being forced to its internal processes from outer domains.

Workflow's Algebra

It can be observed that FOL is employed in the set of workflow connectives. The question arises whether FOL can be substituted by second order logic or others with routing and quantifier operators, which may better fit the tree-like structure of a workflow. This is an attractive choice, providing a workflow with the greater expression. However, more complexity simultaneously arises when one modification of the workflow will transitively affect others to keep updating their structure formulas. Furthermore, it becomes more troublesome when there is a high frequency of changes. From our perspective, the sound solution is to remain using FOL with the integration of the notions to compensate what routing and quantifier operators provide. The relation R in the SWN definition is used to capture this. As a result, employing other types of logic would yield relatively small benefits, but with the increasing unnecessary complexity.

CTL

Another interesting issue is that employing CTL* in Path formulas would provide more expressive power than CTL in specifying requirements in service workflows. However, its computation in reasoning processes is very expensive that may cause undesired

performance and severely impede scalability to large-scale dynamic workflows. Using CTL is sufficient to capture all characteristics of service workflows.

Atomic Proposition

The atomic proposition $(\mathsf{s}, \mathsf{t}, \mathsf{o}, \mathsf{A})$ can be generalised, so that it can express more sophisticated requirements. For example, in a financial audit scenario, an annual financial statement must be audited by two different auditing companies. In this case, it is not necessary to precisely identify the specific companies, but only thing to make sure is that they are not the same company. The atomic proposition can be extended to cover this requirement by introducing a dummy variable, d_1, in which the formula can be expressed as $\mathcal{BP}\mathrm{A}_{\mathrm{audit}}((\mathrm{d}_1, \varepsilon, , \varepsilon, \{\mathrm{audit}\}) \sqcap ((!\,\mathrm{d}_1, \varepsilon, , \varepsilon, \{\mathrm{audit}\})))$. It is important to note that the atomic proposition, as expected for addressing trust-based properties, is large in the sense that further extensions and implementation are possible. In this chapter, rather than thoroughly exploring such matters in comprehensive detail, it has been limited to only the basic type and has left other extension possibilities for future work.

Applications of Petri Net as a Workflow Modelling Language

Workflow is one of the main application domains for Petri Net, by allowing the design of complex workflows using advanced routing constructs and providing an abundance of analysis techniques to verify several properties [150]. There are several products which use Petri Net, such as COSA (Software Ley/COSA Solutions, Pullheim, Germany), INCOME (Promatis, Karlsbad, Germany), LEU (Vebacom, Bochum, Germany), ExSpect (Deloitte & Touche, Netherlands), and Kontinuum (Web and Flo, Australia). Although there are several applications and arguments surrounding Petri Net as a good workflow language, from our perspective, it is used to encapsulate workflow information at the service level by representing how tasks and services are connected, rather than specifically describing workflow operations. However, our approach is not limited to processes defined in Petri Net. It can be modified to be applicable to any graph based process definition language.

Chapter 4

Syntax and Semantics of TS Language

4.1 Chapter Introduction

In this chapter, TS language is formulated by exploring its syntax and semantics that are the core elements for automatic compliance checking to be presented in Chapter 5. Rather than inventing the new one from scratch, the syntax and semantics are extended from FOL and CTL with a new Composite class tailored to the specific structural properties of services associated with tasks in SWN. Because CTL and FOL operators are not meaningful to address all criteria of specification languages in service workflows; for example, they are unable to directly capture Association between Tasks and Services (R3), Composite operators are developed for the treatment of requirements in the association of services and tasks. In this chapter, the syntax and semantics of Path, Composite, and Direction formulas are described. Expressiveness and algebraic properties of TS language are also discussed.

The example here is given to differentiate our logic of the Composite class and the classical propositional logic. Propositional logic is originally used to specify properties of a single component which is best specified properties of a system. For example, a formula $p \wedge q$ states that, in a system to be verified, it must contain properties p and q. This is perfect in determining properties of a single entity, but cannot be applied effectively to composite services. For instance, one statement says that two different services must generate the same result. This can be expressed by using $\sqcap$ operator as $p \sqcap ! p$, where p is a

service property, ! p is another property which is not p. As already stated, this requirement is inexpressible in the propositional logic as the evaluation $p \wedge \sim p$ is always false.

In traditional model verifications such as CTL, the models under consideration are abstracted by transition systems. A transition system simply models a system by a means of states (static structure) and transitions (dynamic structure) [153], which are insufficient to model service workflows that consists of more types of objects, i.e., tasks, services, and their association. The logic of Path class is extended from CTL to be applicable to this characteristic. In short, CTL is used to verify the model abstracted by state-transition system, while Path formulas are used to verify properties of services and tasks in service workflows abstracted by SWN.

The rest of this chapter is organised as follow. We continue with more direct background on the related works of logical languages. Then, the syntax and Semantics of TS language are formulated in Sections 4.3 and 4.4, respectively, which ground for reasoning algorithms in Chapter 5. In Section 4.5, important algebraic properties are described with the discussion of its expressiveness in Section 4.6. Important semantics equivalences are provided in Section 4.7. Based on the example requirements previously addressed in Chapter 1, Section 1.3, Section 4.8 illustrates how they can be expressed by TS formulas. Then, the definition of the subsets of SWN resulted from partially satisfied is introduced in Section 4.9. Finally, the chapter is concluded and discussed in the last section.

4.2 Background

A large variety of logical languages specific to different paradigms have been developed, such as propositional and FOL, e.g., Linear Temporal Logic (LTL), and second order logic, e.g., CTL, and CTL*. The success of these logics is due to their strong mathematical statements, proofs, and practicality. Mostly, in computer science, the main purpose of logical languages is to specify a set of intended properties that a system must have, and to be represented in such a way that they can be formally reasoned. Currently, there is no proposal on logics specific to requirements in the context of service workflows, which makes it difficult for ones to formally specify their requirements efficiently.

In this respect, SWN is a modelling language for service workflows and TS language is developed for requirement representation that can be checked for compliance with SWN. Because the nature of service workflows can be seen as a tree-like structure, some of the CTL operators, combining branching time, linear-time, and quantifier operators, are employed in TS formulas (Path class). Quantifier operators are extended with task annotation that makes CTL operators in our context more specific in quantifying tasks in a workflow. Considering CTL a tree model consisting of a single root and other reachable nodes connected via the finite number of paths, called branches, it can suitably be interpreted over SWN, where branches connected from/to a node form a tree inside a workflow. Two types of CTL operators are employed in Path formulas: temporal operators (U, F, G and X, see [153] for more detail) to specify properties along a path, and quantifier operators ∃ and ∀ to express the for some and the for all paths, respectively. These quantifier operators are enhanced with a task t subscripted in $\exists_t$ and $\forall_t$. As discussed in Chapter 3, we avoids using CTL* because no requirements in our context need the power of CTL* and it is highly computationally expensive.

4.3 Syntax of TS Language

According to the grammar presented in Chapter 3, Section 3.7.3, a TS formula is incrementally built-up from Composite, Path, and Direction operators. When writing a formula, there are three forms of representation depending on types of the operators encountered: prefix (e.g., $\mathcal{H}$, $\mathcal{B}$, $\sim$, $\exists_t\odot$, $\forall_t\Diamond$, $\forall_t\Box$, $\mathcal{F}\mathrm{E}_t$, $\mathcal{P}\mathrm{E}_t$, $\mathcal{F}\mathrm{A}_t$, and $\mathcal{P}\mathrm{A}_t$), infix notations (e.g., ∧, ∨, ⨁, ⋓, ⋃, ⊓, ⊔, ⊞, &, and ||), and the mix of prefix and infix notation (e.g., $\forall_t$[R ⋓ R] and $\exists_t$[R⋃R]). The prefix and infix are used for unary and relational terms, respectively.

Any TS formula is initially built up from the Composite and then by the Path and Direction operators consecutively. Composite and Path have their own Quantifier operators, (E and A, and ∃ and ∀). w is denoted for Direction formulas, r is for Path formulas, and s and z are for two parts of the Quantifier and Property of Composite formulas. Parentheses

are omitted in some cases and will be used to override priorities. Let AP be a set of atomic proposition names $(\mathsf{s}, \mathsf{t}, \mathsf{o}, \mathsf{A})$, the relation of each formula is given by the following rules.

- If z_1 and $\mathrm{z}_2 \in \mathrm{AP}$, z_1, $!\,\mathrm{z}_1$, $\mathrm{z}_1 \sqcap \mathrm{z}_2$, $\mathrm{z}_1 \sqcup \mathrm{z}_2$, and $\mathrm{z}_1 \boxplus \mathrm{z}_2$ form a Composite formula if they are preceded by one of the operators from the Quantifier part, $\mathcal{F}\mathrm{E}_\mathrm{t}$, $\mathcal{P}\mathrm{E}_\mathrm{t}$, $\mathcal{F}\mathrm{A}_\mathrm{t}$, or $\mathcal{P}\mathrm{A}_\mathrm{t}$.
- If s is a Composite formula, s & s or $\mathrm{s} \,||\, \mathrm{s}$ are also Composite formulas.
- If s is a Composite formula, it is also a Path formula.
- If r_1 and r_2 are Path formulas, $\sim\mathrm{r}_1$, $\mathrm{r}_1 \wedge \mathrm{r}_2$, $\mathrm{r}_1 \vee \mathrm{r}_2$, $\mathrm{r}_1 \oplus \mathrm{r}_2$, $\exists_\mathrm{t} \odot \mathrm{r}_1$, $\exists_\mathrm{t} \Diamond \mathrm{r}_1$, $\exists_\mathrm{t} \Box \mathrm{r}_1$ $\exists_\mathrm{t}[\mathrm{r}_1 \Cup \mathrm{r}_2]$, $\exists_\mathrm{t}[\mathrm{r}_1 \cup \mathrm{r}_2]$, $\forall_\mathrm{t} \odot \mathrm{r}_1$, $\forall_\mathrm{t} \Diamond \mathrm{r}_1$, $\forall_\mathrm{t} \Box \mathrm{r}_1$, $\forall_\mathrm{t}[\mathrm{r}_1 \Cup \mathrm{r}_2]$, and $\forall_\mathrm{t}[\mathrm{r}_1 \cup \mathrm{r}_2]$ are also Path formulas.
- If r is a Path formula, $\mathcal{H}\mathrm{r}$ and $\mathcal{B}\mathrm{r}$ are Direction formulas.
- If w_1 and w_2 are Direction formulas, $\sim\mathrm{w}_1$, $\mathrm{w}_1 \wedge \mathrm{w}_2$ and $\mathrm{w}_1 \vee \mathrm{w}_2$ are also Direction formulas and all Direction formulas are well-formed (see **Def. 4.1** below).

The following subsections details how a TS formula is constructed in a stepwise fashion.

Composite Formulas:

1. Any Composite formula must be preceded by one of the Direction operators, and optionally by Path operators.
2. The first part is a set of Composite Quantifier operators, i.e., $\mathcal{F}\mathrm{E}_\mathrm{t}$, $\mathcal{P}\mathrm{E}_\mathrm{t}$, $\mathcal{F}\mathrm{A}_\mathrm{t}$ and $\mathcal{P}\mathrm{A}_\mathrm{t}$. These can be connected by & or $||$ and bind tightly with the Composite Property part.
3. The Composite Property part is used to represent properties of services in two ways: (1) it specifies how the properties are specified through the operators $\sqcap$, $\sqcup$, and $\boxplus$, and (2) the actual property by using atom $(\mathsf{s}, \mathsf{t}, \mathsf{o}, \mathsf{A})$.

A set of Composite operators is adequate if, and only if, it contains at least one of $\{\&, ||\}$, all of $\{\mathcal{F}\mathrm{A}_\mathrm{t}, \mathcal{F}\mathrm{E}_\mathrm{t}, \mathcal{P}\mathrm{A}_\mathrm{t}, \mathcal{P}\mathrm{E}_\mathrm{t}, !\}$, and one of $\{\sqcup, \sqcap\}$. The operators that are not contained in the adequate set can be derived from the operators in that set from algebraic properties to be described in Section 4.5.

Path Formulas:

1. A Path formula is formed after the completion of Composite formulas.
2. The Temporal operators in a Path formula consist of a pair of symbols. The first part is one of ∃ or ∀. ∃ means along at least one path and ∀ means all paths. The second part is one of $\odot$, $\Diamond$, $\Box$, $\Cup$, or $\uplus$. These operators are interpreted in a tree-like structure. It has a single root (an originating service from which a TS formula starts is considered the root node), and a set of paths (other reachable services).
3. Branches of an SWN is determined by the Quantifier operators, ∃ and ∀.
4. The remaining operators ∧ and ∨ use the definitions from propositional logic.

Each of set $\{\exists_t\odot, \exists_t \Cup, \forall_t \Diamond, \sim, \wedge\}$, $\{\exists_t\odot, \exists_t \Cup, \forall_t \Cup, \sim, \vee\}$, and $\{\exists_t\odot, \exists_t \Cup, \exists_t\Box, \sim, \wedge\}$ forms an adequate set which can be described as follows.

- $\exists_t\odot$ is orthogonal to others as it cannot be derived from any combination of others and $\exists_t\odot$ can be written as $\sim\forall_t\odot\sim$.
- $\exists_t \Diamond r_1$ can be defined in term of $\Cup$ as $\exists_t[\top \Cup r_1]$. This is similarly described for $\forall_t \Diamond r_1$.
- $\exists_t\Box r_1$ can be defined in terms of $\Diamond$ as $\sim\exists_t \Diamond \sim r_1$ and in terms of $\Cup$ as $\sim\exists_t[\top \Cup \sim r_1]$. This is similarly described for $\forall_t\Box r_1$.
- $\exists_t[r_1 \uplus r_2]$ can be defined in terms of $\Cup$, as $\exists_t[r_1 \Cup r_2] \vee \exists_t\Box r_1$, where $\exists_t\Box r_1$ can be derived in terms of $\Cup$ as above.
- $\forall_t[r_1 \uplus r_2]$ can be defined in terms of $\Cup$, as $\forall_t[r_1 \Cup r_2] \vee \forall_t\Box r_1$, where $\forall_t\Box r_1$ can be derived in terms of $\Cup$ as above. However, $\exists_t[r_1 \Cup r_2]$ cannot be derived from $\uplus$ since $\exists_t[r_1 \Cup r_2] \not\equiv \exists_t[r_1 \uplus r_2] \wedge \exists_t \Diamond r_2$.
- $\forall_t[r_1 \Cup r_2]$ can be derived from $\sim(\exists_t[\sim r_1 \Cup (\sim r_1 \wedge \sim r_2)] \vee \exists_t\Box\sim r_2)$ (see [163]).
- ∧ can be derived from $\sim(r_1 \wedge r_2) \equiv \sim r_1 \vee \sim r_2$, or in the other way around.

A set of Path operators is adequate if, and only if, it contains at least one of $\{\exists_t\odot, \forall_t\odot\}$, one of $\{\exists_t\Box, \forall_t \Diamond, \forall_t \Cup\}$, $\exists_t \Cup$, one of $\{\wedge, \vee, \oplus\}$, and $\sim$.

Direction Formulas:

1. Every formula must begin with either Henceforth ($\mathcal{H}$) or Backward ($\mathcal{B}$) operators.

2. The remaining operators are defined similarly as in propositional logic.

Each of set $\{\mathcal{H}, \mathcal{B}, \sim, \wedge\}$ and $\{\mathcal{H}, \mathcal{B}, \sim, \vee\}$ is adequate as $\wedge$ and $\vee$ can be substituted from $\sim(w_1 \wedge w_2) \equiv \sim w_1 \vee \sim w_2$. A set of Direction operators in TS formulas is adequate if, and only if, it contains $\mathcal{H}$, $\mathcal{B}$, $\sim$, and one of $\{\wedge, \vee\}$.

Def. 4.1 (Well-formed Formula): The well-formed formulas that are finitely constructed many times from the TS grammar are defined below.

Atom: A propositional atom (s, t, o, A) is a Composite formula if it is preceded by one of the Composite Quantifier operators, $\mathcal{F}E_t$, $\mathcal{P}E_t$, $\mathcal{F}A_t$ or $\mathcal{P}A_t$ and a well-formed formula if further preceded by the Direction operators $\mathcal{H}$ or $\mathcal{B}$, for example, $\mathcal{H}\mathcal{F}E_t(s, t, o, A)$.

Composite formulas: (Let Δ be $\mathcal{F}E_t$, $\mathcal{P}E_t$, $\mathcal{F}A_t$ or $\mathcal{P}A_t$) If Δz_1 and Δz_2 are Composite formulas, then so are $\Delta! z_1$, $\Delta(z_1 \sqcap z_2)$, $\Delta(z_1 \sqcup z_2)$, $\Delta(z_1 \boxplus z_2)$, $\Delta z_1 \,\&\, \Delta z_2$, and $\Delta z_1 \,||\, \Delta z_2$. Every composite formula is also a Path formula.

Path formulas: If r_1 and r_2 are Path formulas, then so are $\sim r_1$, $r_1 \wedge r_2$, $r_1 \vee r_2$, $\exists_t \odot r_1$, $\exists_t \Diamond r_1$, $\exists_t \Box r_1$, $\exists_t [r_1 \Cup r_2]$, $\exists_t [r_1 \Cup r_2]$, $\forall_t \odot r_1$, $\forall_t \Diamond r_1$, $\forall_t \Box r_1$, $\forall_t [r_1 \Cup r_2]$, and $\forall_t [r_1 \Cup r_2]$. These are well-formed if preceded by one of the Direction operators, $\mathcal{H}$ or $\mathcal{B}$.

Direction formulas: If w_1 and w_2 are Direction formulas, then so are $\sim w_1$, $w_1 \wedge w_2$, and $w_1 \vee w_2$. Every Direction formula is well-formed.

The following convention specifies binding priorities on these operators. Although some of the operators have been addressed in CTL and propositional logic, it is important to associate with the new Composite operators.

Convention 4.1 (Binding Priorities): The priorities of some TS operators are similar to what has been defined in propositional logic. The first priority is the Composite operators that come in order of !, $\sqcap$, $\boxplus$, $\sqcup$, $\mathcal{F}E_t$, $\mathcal{P}E_t$, $\mathcal{F}A_t$, $\mathcal{P}A_t$, & and $||$. Then, the unary operators in Path formulas, $\forall_t \odot$, $\forall_t \Diamond$, $\forall_t \Box$, $\exists_t \odot$, $\exists_t \Diamond$, $\exists_t \Box$, bind tightly. Next in the order comes $\wedge$, $\oplus$, $\vee$, $\forall_t \Cup$, $\exists_t \Cup$, $\forall_t \Cup$, and $\exists_t \Cup$. Finally, the Direction operators $\mathcal{H}$ and $\mathcal{B}$ preceding in every clause are considered the lowest priority.

Typically, the parentheses are used for overriding these priorities for ease of understanding. However, many of them can be omitted without ambiguity. In order to understand the syntax, the following examples show some well-formed and not well-formed formulas. For simplicity, let $\mathrm{q_i}$ be atom $(\mathrm{s}, \mathrm{t}, \mathrm{o}, \mathrm{A})$. Some of well-formed formulas are given below.

- $\mathcal{H}(\mathcal{F}\mathrm{A_t q_1}\ \&\ \mathcal{P}\mathrm{E_t q_2})$

  Direction operators: $\mathcal{H}$

  Composite operators: $\mathcal{F}\mathrm{A_t q_1}\ \&\ \mathcal{P}\mathrm{E_t q_2}$
- $\mathcal{H}\mathcal{F}\mathrm{A_t q_1} \wedge\ \mathcal{B}\mathcal{P}\mathrm{E_t}(\mathrm{q_2} \sqcup \mathrm{q_3})$

  Direction operators: $\mathcal{H}$,$\mathcal{B}$, $\wedge$

  Composite operators: $\mathcal{F}\mathrm{A_t q_1}$, $\mathcal{P}\mathrm{E_t}(\mathrm{q_2} \sqcup \mathrm{q_3})$
- $\mathcal{B}(\mathcal{P}\mathrm{A_t q_1} \wedge \mathcal{P}\mathrm{E_t q_2})$,

  Direction operators: $\mathcal{B}$

  Path operators: $\wedge$

  Composite operators: $\mathcal{P}\forall_\mathrm{t}\mathrm{q_1}$, $\mathcal{P}\exists_\mathrm{t}\mathrm{q_2}$
- $\mathcal{B}(\exists_\mathrm{t} \Diamond\, \mathcal{F}\mathrm{A_t q_1} \vee \forall_\mathrm{t}[\mathcal{F}\mathrm{A_t q_1} \Cup \mathcal{P}\mathrm{A_t q_2}])$

  Direction operators: $\mathcal{B}$

  Path operators: $\exists_\mathrm{t} \Diamond$, $\forall_\mathrm{t} \Cup$, $\vee$

  Composite operators: $\mathcal{F}\mathrm{A_t q_1}$, $\mathcal{P}\mathrm{A_t q_2}$
- $\mathcal{B}(\exists_\mathrm{t} \Diamond\, \mathcal{P}\mathrm{A_t}(\mathrm{q_1} \sqcap \mathrm{q_2}) \vee \forall_\mathrm{t}[\mathcal{P}\mathrm{E_t}(\mathrm{q_1} \sqcup \mathrm{q_3}) \Cup \mathcal{F}\mathrm{A_t q_2}])$

  Direction operators: $\mathcal{B}$

  Path operators: $\exists_\mathrm{t} \Diamond$, $\vee$, $\forall_\mathrm{t} \Cup$

  Composite operators: $\mathcal{P}\mathrm{A_t}(\mathrm{q_1} \sqcap \mathrm{q_2})$, $\mathcal{P}\mathrm{E_t}(\mathrm{q_1} \sqcup \mathrm{q_3})$, $\mathcal{F}\mathrm{A_t q_2}$

And the following formulas are not well-formed.

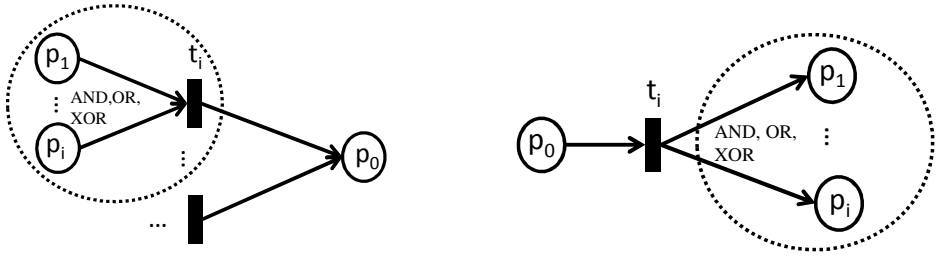
- $\mathcal{B}\mathrm{q_1}$, $\mathcal{H}\mathrm{q_1}$: a Composite Quantifier operator part is missing.
- $\mathcal{B}\mathrm{q_1} \wedge\ \mathrm{q_2}$: a Composite Quantifier and Direction operator are missing.
- $\mathcal{B}{\sim}\forall_\mathrm{t}{\sim}\odot(\mathrm{q_1} \wedge \mathrm{q_2})$: a negation operator cannot be placed between $\forall_\mathrm{t}\odot$.
- $\mathcal{B}(\exists_\mathrm{t} \Diamond \odot(\mathrm{q_1}))$: a quantifier operator is lacking in one of the Path operator.
- $\mathcal{B}(\exists_\mathrm{t} \Diamond\, (\mathrm{q_1} \sqcap \exists_\mathrm{t}\mathrm{q_2})$: a Composite Quantifier operator is misplaced.

It can be observed that our TS operators are more expressive than CTL. The term t is used to augment the expressiveness of Quantifier Operators: A_t, E_t, $\forall_t$ and $\exists_t$, as to precisely specify tasks in consideration. t can also be omitted in order to make the formula less restrictive, meaning that all paths and services are quantified.

4.4 Semantics of TS Language

This section provides comprehensive details on the semantics of TS formulas which is an underlying part for reasoning about SWN.

Def. 4.2 (TS Semantics): Given the Satisfiability relations from **Def. 3.5,** each TS formula ω is interpreted over an SWN $\mathcal{M}$. Let q_i be (s, t, o, A) and p_0 denote a service imposing ω. The semantics of $\mathcal{M} \vDash \omega$, $\mathcal{M} \vDash' \omega$, or $\mathcal{M} \nvDash \omega$, can be understood by each workflow connective in the following subsections.



(a) Service-join (b) Service-split

Figure 4.1: Graphical Illustration of Service-join and Service-split Types

4.4.1 Composite Formulas

The purpose of Composite formulas is to verify properties of services associated with tasks. The properties of services can be considered in two aspects: (1) services composed for a task (*service-join*), and (2) services connected from a task (*service-split*). Figure 4.1 illustrates the graphical representation of these two notions.

The semantics definition of each workflow connective is defined in the following clauses. For simplicity, $\mathcal{F}$ and $\mathcal{P}$ operators are omitted in the Composite term as they are

only used to indicate *service-join* or *service-split*, and in some clauses Δ is used for either E_t or A_t.

1. $\mathcal{M}, p_0 \vDash \top$, and $\mathcal{M}, p_0 \nvDash \bot$.
2. $\mathcal{M}, p_0 \vDash \Delta q_1$ & Δq_2 iff $\mathcal{M}, p_0 \vDash \Delta q_1$ and Δq_2.
3. $\mathcal{M}, p_0 \vDash \Delta q_1 \,||\, \Delta q_2$ iff $\mathcal{M}, p_0 \vDash \Delta q_1$ or Δq_2.

AND Connective (p_1 AND p_2 AND ... AND $p_i, 1 \leq i \leq n$):

1. $\mathcal{M}, p_0 \vDash A_t q_1$, iff all services p_i must satisfy q_1; otherwise $\mathcal{M}, p_0 \nvDash A_t q_1$.
2. $\mathcal{M}, p_0 \vDash E_t q_1$, iff at least one service p_i satisfies q_1; otherwise $\mathcal{M}, p_0 \nvDash E_t q_1$.
3. $\mathcal{M}, p_0 \vDash A_t(q_1 \sqcap q_2)$, iff for all services p_i and p_j, at least one p_i satisfies $q_1$and p_j satisfies q_2. Otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \sqcap q_2)$.
4. $\mathcal{M}, p_0 \vDash E_t(q_1 \sqcap q_2)$, iff for some services p_i and p_j, at least one p_i satisfies q_1 and p_j satisfies q_2. Otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \sqcap q_2)$.
5. $\mathcal{M}, p_0 \vDash A_t(q_1 \sqcup q_2)$, iff all services p_i must satisfy q_1 or q_2; otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \sqcup q_2)$.
6. $\mathcal{M}, p_0 \vDash E_t(q_1 \sqcup q_2)$, iff some services p_i must satisfy q_1 or q_2; otherwise $\mathcal{M}, p_0 \nvDash E_t(q_1 \sqcup q_2)$.
7. $\mathcal{M}, p_0 \vDash A_t(q_1 \boxplus q_2)$, iff all services p_i satisfies only one of q_1 or q_2; otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \boxplus q_2)$.
8. $\mathcal{M}, p_0 \vDash E_t(q_1 \boxplus q_2)$, iff some services p_i satisfies only one of q_1 or q_2, if some p_i satisfy q_1, there will be no other p_i that satisfies q_2; otherwise $\mathcal{M}, p_0 \nvDash E_t(q_1 \boxplus q_2)$.

OR Connectives (p_1 OR p_2 OR ... OR $p_i, 1 \leq i \leq n$):

1. $\mathcal{M}, p_0 \vDash A_t q_1$, iff all services p_i must satisfy q_1; $\mathcal{M}, p_0 \vDash' A_t q_1$, iff some p_i satisfy q_1; otherwise $\mathcal{M}, p_0 \nvDash A_t q_1$.
2. $\mathcal{M}, p_0 \vDash E_t q_1$ is semantically identical to $A_t q_1$.
3. $\mathcal{M}, p_0 \vDash A_t(q_1 \sqcap q_2)$, iff every service p_i must satisfies q_1 and q_2 ; $\mathcal{M}, p_0 \vDash' A_t(q_1 \sqcap q_2)$, iff some of each p_i satisfy q_1 and q_2; otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \sqcap q_2)$.
4. $\mathcal{M}, p_0 \vDash E_t(q_1 \sqcap q_2)$ is semantically identical to $A_t(q_1 \sqcap q_2)$.

5. $\mathcal{M}, p_0 \vDash A_t(q_1 \sqcup q_2)$, iff every service p_i must satisfy q_1 or q_2; $\mathcal{M}, p_0 \vDash' A_t(q_1 \sqcup q_2)$, iff some of each p_i satisfy q_1 or q_2; otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \sqcup q_2)$.
6. $\mathcal{M}, p_0 \vDash E_t(q_1 \sqcup q_2)$ is semantically identical to $A_t(q_1 \sqcup q_2)$.
7. $\mathcal{M}, p_0 \vDash A_t(q_1 \boxplus q_2)$, iff all services p_i satisfy either q_1 or q_2, but not both (there is no p_i and p_j that p_i satisfies q_1 and p_j satisfies q_2); $\mathcal{M}, p_0 \vDash' A_t(q_1 \boxplus q_2)$, iff some of each p_i satisfy only one either q_1 or q_2; otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \boxplus q_2)$.
8. $\mathcal{M}, p_0 \vDash E_t(q_1 \boxplus q_2)$ is semantically identical to $A_t(q_1 \boxplus q_2)$.

It is important to note that some semantics of A_t and E_t in the above clauses are identically interpreted. The reason is that OR connective involves with uncertainties regarding which services are going to be instantiated at runtime. Since the only concern is to ensure that a TS formula still valid, partly valid, or invalid, for all activations, all services connected by OR are evaluated by the same semantics regardless of the effect of A_t or E_t.

XOR Connectives (p_1 XOR p_2 XOR ... XOR $p_i, 1 \leq i \leq n$):

1. $\mathcal{M}, p_0 \vDash A_t q_1$, iff all services p_i must satisfy q_1; $\mathcal{M}, p_0 \vDash' A_t q_1$, iff some p_i satisfy q_1; otherwise $\mathcal{M}, p_0 \nvDash A_t q_1$.
2. $\mathcal{M}, p_0 \vDash E_t q_1$ is semantically identical to $A_t q_1$.
3. $\mathcal{M}, p_0 \vDash A_t(q_1 \sqcap q_2)$, iff every service p_i must satisfy q_1 and q_2; $\mathcal{M}, p_0 \vDash' A_t(q_1 \sqcap q_2)$, iff some of each p_i satisfy q_1 and q_2; otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \sqcap q_2)$.
4. $\mathcal{M}, p_0 \vDash E_t(q_1 \sqcap q_2)$ is semantically identical to $\mathcal{M}, p_0 \vDash A_t(q_1 \sqcap q_2)$.
5. $\mathcal{M}, p_0 \vDash A_t(q_1 \sqcup q_2)$, iff every service p_i must satisfy q_1 or q_2; $\mathcal{M}, p_0 \vDash' A_t(q_1 \sqcup q_2)$, iff some of each p_i satisfy q_1 or q_2; otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \sqcup q_2)$.
6. $\mathcal{M}, p_0 \vDash E_t(q_1 \sqcup q_2)$ is semantically identical to $\mathcal{M}, p_0 \vDash A_t(q_1 \sqcup q_2)$.
7. $\mathcal{M}, p_0 \vDash A_t(q_1 \boxplus q_2)$, iff all services p_i satisfies either q_1 or q_2 (different from Clause 7 in OR connective, there can be p_i and p_j such that p_i satisfies q_1 and p_j satisfies q_2), $p_0 \vDash' A_t(q_1 \boxplus q_2)$, iff for some of p_i, each of them satisfies either q_1 or q_2; otherwise $\mathcal{M}, p_0 \nvDash A_t(q_1 \boxplus q_2)$.
8. $\mathcal{M}, p_0 \vDash E_t(q_1 \boxplus q_2)$ is semantically identical to $\mathcal{M}, p_0 \vDash A_t(q_1 \boxplus q_2)$.

Similarly to the semantics of OR connective, the semantics of A_t and E_t are identical. It can be noticed that the semantics of OR and XOR in many cases are similar (except clauses

7 and 8), despite their different definitions and meanings in the aspect of workflow control. This is because these two connectives involve the uncertainty of the choices of service activations.

Simplifying Workflow Formulas

This section describes a data structure called Exclusive Disjunctive Normal Form (EDNF) a an example to distinguish the different meanings of the semantics of each workflow connective. All services connected by AND connectives are required for every execution, while the remaining OR and XOR provide choices of service activations. For example, A OR B contain three possible executions, i.e., A, B, and A AND B, while A XOR B contains two possible executions, i.e., A and B. It is useful to look at the simplification of these connectives in the form of EDNF. Let *terms* be defined as variables or a group of variables that are connected by AND and each *term* is connected by XOR. In this EDNF form, each term indicates services involved in an execution, and the number of terms indicates the number of possible executions. The example below demonstrates the transformation into EDNF.

$$\text{A AND (B OR C)} = \text{(A AND B) OR (A AND C)}$$
$$\text{(A AND B) OR (A AND C)} = \text{(A AND B) XOR (A AND C) XOR (A AND B AND C)}$$
$$\text{A AND (B XOR C)} = \text{(A AND B) XOR (A AND C)}$$
$$\text{A OR (B XOR C)} = \text{(A OR B) XOR (A OR C)}$$
$$\text{A OR B} = \text{A XOR B XOR (A AND B)}$$

The semantics from the previous subsection can be used to evaluate *satisfiability* relations as follows. Firstly, a TS formula focuses on specifying properties of each *term* that will generate either that *term* is *satisfied* or *unsatisfied*. Then, based on the result of each term, if all *terms* are satisfied, we say that the formula is *satisfied*, if only some terms are satisfied, we say that the formula is *partially satisfied*, otherwise, it is *unsatisfied*. More types of data structures and techniques will be introduced in Chapter 5 with the efficiency comparison in compliance checking.

4.4.2 Path Formulas

Whenever Quantifier and Temporal operators appear in a formula, it is important to consider the structure of how tasks are interconnected. The purpose of Path operators is to specify, for some or for all, the sequences of tasks and their dependency, where properties of a task can be determined from the validity of the Composite part of a TS formula. To deliberately address paths in a workflow, two accessibility relations are defined. $\succ$ and $>^{n}$ are interpreted as the transitive relation (one-step path) and transitive closure (Reachability relation), respectively. $>^{n}$ symbolises $\succ$ n times. The semantics of Path formulas is shown in the following clauses. Let S be a Composite term and ∇ be used for $\mathcal{H}$ or $\mathcal{B}$.

1. $\mathcal{M}, p_0 \vDash \nabla(S_1 \wedge S_2)$ iff $\mathcal{M}, p_0 \vDash \nabla S_1$ and $\mathcal{M}, p_0 \vDash \nabla S_2$.
2. $\mathcal{M}, p_0 \vDash \nabla(S_1 \vee S_2)$ iff $\mathcal{M}, p_0 \vDash \nabla S_1$ or $\mathcal{M}, p_0 \vDash \nabla S_2$.
3. $\mathcal{M}, p_0 \vDash \mathcal{B}\forall_t[S_1 \Cup S_2]$ iff for all i where $p_{n_i} >^{n} p_0$, $\mathcal{M}, p_{n_i} \vDash \mathcal{B}S_2$ and for all p_{m_i} through a task t where $m = 0, \dots, n-1$, $\mathcal{M}, p_{m_i} \vDash \mathcal{B}S_1$.
4. $\mathcal{M}, p_0 \vDash \mathcal{H}\forall_t[S_1 \Cup S_2]$ iff for all i where $p_0 >^{n} p_{n_i}$, $\mathcal{M}, p_{n_i} \vDash \mathcal{H}S_2$ and for all p_{m_i} through a task t where $m = 0, \dots, n-1$, $\mathcal{M}, p_{m_i} \vDash \mathcal{H}S_1$.
5. $\mathcal{M}, p_0 \vDash \mathcal{B}\exists_t[S_1 \Cup S_2]$ iff for some i where $p_{n_i} >^{n} p_0$, such that $\mathcal{M}, p_{n_i} \vDash \mathcal{B}S_2$ and for all p_{m_i} through a task t where $m = 0, \dots, n-1$, $\mathcal{M}, p_{m_i} \vDash \mathcal{B}S_1$.
6. $\mathcal{M}, p_0 \vDash \mathcal{H}\exists_t[S_1 \Cup S_2]$ iff for some i where $p_0 >^{n} p_{n_i}$, such that $\mathcal{M}, p_{n_i} \vDash \mathcal{H}S_2$ and for all p_{m_i} through a task t where $m = 0, \dots, n-1$, $\mathcal{M}, p_{m_i} \vDash \mathcal{H}S_1$.
7. $\mathcal{M}, p_0 \vDash \nabla(\forall_t \Diamond S_1)$ iff $\nabla(\forall_t[\top \Cup S_1])$.
8. $\mathcal{M}, p_0 \vDash \nabla\exists_t \Diamond (S_1)$ iff $\nabla(\exists_t[\top \Cup S_1])$.
9. $\mathcal{M}, p_0 \vDash \nabla(\forall_t \Box S_1)$ iff $\nabla(\sim\exists_t \Diamond \sim S_1)$.
10. $\mathcal{M}, p_0 \vDash \nabla(\exists_t \Box S_1)$ iff $\nabla(\sim\forall_t \Diamond \sim S_1)$.
11. $\mathcal{M}, p_0 \vDash \nabla(\forall_t[S_1 \cup S_2])$ iff $\nabla(\forall_t[S_1 \Cup S_2] \vee \forall_t \Box S_1)$.
12. $\mathcal{M}, p_0 \vDash \nabla(\exists_t[S_1 \cup S_2])$ iff $\nabla(\exists_t[S_1 \Cup S_2] \vee \exists_t \Box S_1)$.
13. $\mathcal{M}, p_0 \vDash \mathcal{B}\forall_t \odot S_1$ iff for all i where $p_i \succ p_0$ through a task t, $\mathcal{M}, p_i \vDash \mathcal{B}S_1$.
14. $\mathcal{M}, p_0 \vDash \mathcal{H}\forall_t \odot S_1$ iff for all i where $p_0 \succ p_i$ through a task t, $\mathcal{M}, p_i \vDash \mathcal{H}S_1$.
15. $\mathcal{M}, p_0 \vDash \mathcal{B}\exists_t \odot S_1$ iff for some i where $p_i \succ p_0$ through a task t, $\mathcal{M}, p_i \vDash \mathcal{B}S_1$.
16. $\mathcal{M}, p_0 \vDash \mathcal{H}\exists_t \odot S_1$ iff for some i where $p_0 \succ p_i$ through a task t, $\mathcal{M}, p_i \vDash \mathcal{H}S_1$.

The above semantics are derived from CTL except the extension of the constrained task t annotated in $\forall_{\mathrm{t}}$ and $\exists_{\mathrm{t}}$, indicating to which task a formula applies. All Path formulas must be preceded by either $\mathcal{H}$ or $\mathcal{B}$ indicating the direction of a path in a workflow. For example, the formula $\mathcal{B}\forall_{\mathrm{t}}[\mathrm{S}_1 \Cup \mathrm{S}_2]$ holds for all paths in the backward direction if it is the case that S_1 holds form a task t continuously until S_2 holds. In Clauses 7-10, $\Box$ and $\Diamond$ include the present position of p_0 as to enable p_0 to specify properties of other services at the task that p_0 is connected.

4.4.3 Direction Formulas

1. $\mathcal{B}$ indicates the application of a formula in the backward direction from the originating service to the preceding tasks. There must be at least one path from those tasks reachable from the originating service.
2. $\mathcal{H}$ indicates the application of a formula in the forward direction from the originating service to the subsequent tasks along workflow paths.
3. The rest of Direction operators including $\wedge$, $\vee$, and $\sim$, are similar to the definitions in propositional logic.

As mentioned, the originating service is a starting service to which a TS formula applies whereas the imposing service is the service that imposes a TS formula. According to the definitions of BTP and HTP, an originating service is restricted to be the same as an imposing service, while this is not necessary in ET, in which an originating service can be arbitrary. Direction operators are meaningful in specifying the direction of the upstream or downstream tasks of a TS formula.

4.5 Algebraic Properties, Soundness, and Completeness

In order to make TS language rigorous, it should be developed in the way that uses logical symbols with some important properties. We write $\omega \equiv \varphi$ to symbolise semantic equivalence for each pair of formulas. Assume that Δ is used for of $\mathcal{F}\mathrm{E}_{\mathrm{t}}, \mathcal{P}\mathrm{E}_{\mathrm{t}}, \mathcal{F}\mathrm{A}_{\mathrm{t}}$, or $\mathcal{P}\mathrm{A}_{\mathrm{t}}$, Δ' is for $\mathcal{F}$ or $\mathcal{P}$ and ∇ is for $\mathcal{H}$ or $\mathcal{B}$.

Rules	**Direction Formulas**	**Path Formulas**	**Composite Formulas**
Identity Property	$W \vee \perp \equiv W$ $W \wedge \top \equiv W$	$R \vee \perp \equiv R$ $R \wedge \top \equiv R$	$Z \sqcup \varepsilon \equiv Z$ $Z \sqcap \varepsilon \equiv Z$
Commutative Property	$W_1 \vee W_2 \equiv W_2 \vee W_1$ $W_1 \wedge W_2 \equiv W_2 \wedge W_1$ $W_1 \oplus W_2 \equiv W_2 \oplus W_1$	$R_1 \vee R_2 \equiv R_2 \vee R_1$ $R_1 \wedge R_2 \equiv R_2 \wedge R_1$	$\Delta(Z_1 \sqcup Z_2) \equiv \Delta(Z_2 \sqcup Z_1)$ $\Delta(Z_1 \sqcap Z_2) \equiv \Delta(Z_2 \sqcap Z_1)$ $\Delta(Z_1 \boxplus Z_2) \equiv \Delta(Z_2 \boxplus Z_1)$
Associative Property	$(W_1 \vee W_2) \vee W_3 \equiv$ $W_1 \vee (W_2 \vee W_3)$ $(W_1 \wedge W_2) \wedge W_3 \equiv$ $W_1 \wedge (W_2 \wedge W_3)$	$(R_1 \vee R_2) \vee R_3 \equiv$ $R_1 \vee (R_2 \vee R_3)$ $(R_1 \wedge R_2) \wedge R_3 \equiv$ $R_1 \wedge (R_2 \wedge R_3)$ $(R_1 \oplus R_2) \oplus R_3 \equiv$ $R_1 \oplus (R_2 \oplus R_3)$	$\Delta((Z_1 \sqcup Z_2) \sqcup Z_3) \equiv$ $\Delta(Z_1 \sqcup (Z_2 \sqcup Z_3))$ $\Delta((Z_1 \sqcap Z_2) \sqcap Z_3) \equiv$ $\Delta\big(Z_1 \sqcap (Z_2 \sqcap Z_3)\big)$ $\Delta((Z_1 \boxplus Z_2) \boxplus Z_3) \equiv$ $\Delta(Z_1 \boxplus (Z_2 \boxplus Z_3))$
Distributive Property	$W_1 \vee (W_2 \wedge W_3) \equiv$ $(W_1 \vee W_2) \wedge (W_1 \vee W_3)$ $W_2 \wedge (W_2 \vee W_3) \equiv$ $(W_1 \wedge W_2) \vee (W_1 \wedge W_3)$	$R_1 \vee (R_2 \wedge R_3) \equiv$ $(R_1 \vee R_2) \wedge (R_1 \vee R_3)$ $R_2 \wedge (R_2 \vee R_3) \equiv$ $(R_1 \wedge R_2) \vee (R_1 \wedge R_3)$	$*\Delta(Z_1 \sqcap (Z_2 \sqcup Z_3)) \equiv$ $\Delta((Z_1 \sqcap Z_2) \sqcup (Z_1 \sqcap Z_3))$ $*\Delta(Z_1 \sqcup (Z_2 \sqcap Z_3)) \equiv$ $\Delta((Z_1 \sqcup Z_2) \sqcap (Z_1 \sqcup Z_3))$ $*\Delta(Z_1 \sqcap (Z_2 \boxplus Z_3)) \equiv$ $\Delta((Z_1 \sqcap Z_2) \boxplus (Z_1 \sqcap Z_3)))$ $*\Delta Z_1 \,\|\| (\Delta Z_2 \,\&\, Z_3) \equiv$ $(\Delta Z_1 \,\|\| \Delta Z_2) \,\&\, (\Delta Z_1 \|\| \Delta Z_3)$ $*\Delta Z_1 \,\&\, (\Delta Z_2 \,\|\| Z_3) \equiv$ $(\Delta Z_1 \,\&\, \Delta Z_2) \,\|\| (\Delta Z_1 \|\| \Delta Z_3)$
De Morgan's Rule	$\sim(W_1 \vee W_2) \equiv \sim W_1 \wedge \sim W_2$ $\sim(W_1 \wedge W_2) \equiv \sim W_1 \vee \sim W_2$	$\sim(R_1 \vee R_2) \equiv \sim R_1 \wedge \sim R_2$ $\sim(R_1 \wedge R_2) \equiv \sim R_1 \vee \sim R_2$ $*\sim \exists_t \odot R \equiv \forall_t \odot \sim R$ $*\sim \exists_t \Diamond R \equiv \forall_t \Box \sim R$ $*\sim \forall_t \odot R \equiv \exists_t \odot \sim R$ $*\sim \forall_t \Diamond R \equiv \exists_t \Box \sim R$	$*!\Delta' E_t Z \equiv \Delta' A_t ! Z$ $*!\Delta' A_t Z \equiv \Delta' E_t ! Z$ $*! (\Delta Z_1 \,\&\, \Delta Z_2) \equiv$ $!\Delta Z_1 \,\|\| \,!\Delta Z_2$ $*! (\Delta Z_1 \,\|\| \Delta Z_2) \equiv$ $!\Delta Z_1 \,\&\, !\Delta Z_2$

There are additional rules of inference in which they can be derived from the initial set of the given rules, so in some sense they are redundant. Nevertheless, they are pretty intuitive and very useful. Some other essential equivalences of Composite transformation is shown below.

$\sqcap$ *Elimination*	$*\Delta' E_t(Z_1 \sqcap Z_2) \equiv \Delta' E_t Z_1 \,\&\, \Delta' E_t Z_2$
	$*\Delta' A_t(Z_1 \sqcap Z_2) \equiv \Delta' A_t(Z_1 \sqcup Z_2) \,\&\, \Delta' E_t Z_1 \,\&\, \Delta' E_t Z_2$
$\sqcup$ *Elimination*	$*\Delta' E_t(Z_1 \sqcup Z_2) \equiv \Delta' E_t Z_1 \,\|\|\, \Delta' E_t Z_2$
	$*\Delta' A_t(Z_1 \sqcup Z_2) \equiv \Delta' A_t(Z_1 \sqcap Z_2) \,\|\|\, \Delta' A_t Z_1 \,\|\|\, \Delta' A_t Z_2$

⊞ Transformation	$*\Delta' E_t(Z_1 \boxplus Z_2) \equiv (\Delta' E_t Z_1 \,\&\, !\Delta' E_t Z_2) \,||\, (!\Delta' E_t Z_1 \,\&\, \Delta' E_t Z_2)$
	$*\Delta' A_t(Z_1 \boxplus Z_2) \equiv (\Delta' A_t Z_1 \,\&\, !\Delta' E_t Z_2) \,||\, (!\Delta' E_t Z_1 \,\&\, \Delta' A_t Z_2)$
Breaking Association ⊔ *and* ⊞	$*\nabla\Delta' E_t(Z_1 \sqcup (Z_2 \boxplus Z_3)) \equiv \Delta' E_t Z_1 \,||\, \Delta' E_t(Z_2 \boxplus Z_3)$
	$\equiv \Delta' E_t Z_1 \,||\, (\Delta' E_t Z_2 \,\&\, !\Delta' E_t Z_3) \,||\, (!\Delta' E_t Z_2 \,\&\, \Delta' E_t Z_3)$
	$*\nabla\Delta' A_t(\phi \sqcup (\omega \boxplus \psi)) \equiv \nabla((\Delta' A_t(\phi \sqcup \omega) \,\&\, !\Delta' E_t\psi) \,||$
	$(\Delta' A_t(\phi \sqcup \psi) \,\&\, !\Delta' E_t\omega) \,||\, \Delta' A_t\phi)$

* indicates the new properties of TS formulas. Below, some important properties are provided and the proofs are given in Appendix A.

Theorem 4.1: The equivalence $\Delta' A_t(\phi \sqcup \omega) \equiv \Delta' A_t(\phi \sqcap \omega) \,||\, \Delta' A_t\phi \,||\, \Delta' A_t\omega$ holds for all Composite formulas ϕ and ω.

Theorem 4.2: The equivalence $\Delta' A_t(\phi \sqcap \omega) \equiv \Delta' A_t(\phi \sqcup \omega) \,\&\, \Delta' E_t\phi \,\&\, \Delta' E_t\omega$ holds for all Composite formulas ϕ and ω.

Theorem 4.3: The equivalence $\Delta' E_t(Z_1 \boxplus Z_2) \equiv (\Delta' E_t\phi \,\&\, !\Delta' E_t\omega) \,||\, (!\Delta' E_t\phi \,\&\, \Delta' E_t\omega)$ holds for all Composite formulas ϕ and ω.

Theorem 4.4: The equivalence $\Delta' A_t(Z_1 \boxplus Z_2) \equiv (\Delta' A_t\phi \,\&\, !\Delta' E_t\omega) \,||\, (!\Delta' E_t\phi \,\&\, \Delta' A_t\omega)$ holds for all Composite formulas ϕ and ω.

Theorem 4.5: The equivalence $\nabla\Delta' E_t(\phi \sqcup (\omega \boxplus \psi)) \equiv \nabla(\Delta' E_t\phi \,||\, (\Delta' E_t\omega \,\&\, !\Delta' E_t\psi) \,||\, (!\Delta' E_t\omega \,\&\, \Delta' E_t\psi))$ holds for all Composite formulas ϕ, ω and ψ.

Theorem 4.6: The equivalence
$\nabla\Delta' A_t(\phi \sqcup (\omega \boxplus \psi)) \equiv \nabla((\Delta' A_t(\phi \sqcup \omega) \,\&\, !\Delta' E_t\psi) \,||\, (\Delta' A_t(\phi \sqcup \psi) \,\&\, !\Delta' E_t\omega) \,||\, \Delta' A_t\phi)$
holds for all Composite formulas ϕ, ω and ψ.

Several TS operators used the same definitions from propositional logic and CTL with the extension of Composite formulas tailored to service composition. In this respect, there is clear evidence in proving soundness, completeness, and consistency properties of propositional logic and CTL. This section shows how these properties can be recognised in the Composite part which includes semantics extension of propositional logic, i.e., ⊓, and ⊔.

Theorem 4.7 (Soundness): Let $S = \{s_1, \dots, s_n\}$ are services for a task t, with a sequent $\phi_1, \phi_2, \dots, \phi_n$. The meaning of each premise ϕ_i refers to $s_i(\alpha)$ where the service s_i satisfies α. The soundness says that if $\phi_1, \phi_2, \dots, \phi_n \vdash \omega$ holds, then $\phi_1, \phi_2, \dots, \phi_n \vDash \omega$ also holds.

Theorem 4.8 (Completeness): Let $S = \{s_1, \dots, s_n\}$ are services for a task t, with a sequent $\phi_1, \phi_2, \dots, \phi_n$. The meaning of each premise ϕ_i refers to $s_i(\alpha)$ where the service s_i satisfies α. The completeness says that If $\phi_1, \phi_2, \dots, \phi_n \vDash \omega$ is valid, then $\phi_1, \phi_2, \dots, \phi_n \vdash \omega$ also holds.

The main idea for the proofs of soundness and completeness is that all Composite formulas employ all operators in FOL and the semantics extension which includes $\sqcap$ and $\sqcup$ can be translated into FOL. The reason is that since FOL is sound and complete, we will automatically obtain that TS language is also sound and complete.

To achieve this, let set up the environment as follows. Suppose $S = \{s_1, \dots, s_n\}$ are services for a task t, which are considered a sequent $\phi_1, \phi_2, \dots, \phi_n$. The assignment of $s_i(\alpha)$ to T or F if service s_i satisfies, or does not satisfies, α, respectively. In this way, truth tables can be constructed based on atom $s_i(\alpha)$. There can be a number of truth tables that correspond to the number of formula presented. For instance, $A(\phi \sqcap \omega)$ has two truth tables, one for $s_i(\phi)$ and another for $s_i(\omega)$. The following case analysis describes how Composite formula can be translated to FOL.

- $A(\phi)$: Truth table is constructed with atoms $s_1(\phi)$, $s_2(\phi)$, … $s_n(\phi)$. By its definition, every atom must be true which is $s_1(\phi) \wedge s_2(\phi) \wedge \dots \wedge s_n(\phi)$, or $\bigwedge s_i(\phi)$.
- $E(\phi)$: Truth table is constructed with atoms $s_1(\phi)$, $s_2(\phi)$, … $s_n(\phi)$. By its definition, at least one $s_i(\phi)$ is true which is $s_1(\phi) \vee s_2(\phi) \vee \dots \vee s_n(\phi)$, or $\bigvee s_i(\phi)$.
- $A(\phi \sqcup \omega)$: Two truth tables are constructed with atoms $s_1(\phi), s_2(\phi), \dots s_n(\phi)$ and $s_1(\omega), s_2(\omega), \dots s_n(\omega)$. By its definition, all atoms must satisfy ϕ or ω which is $\bigwedge(s_i(\phi) \vee s_i(\omega))$.
- $E(\phi \sqcup \omega)$: Two truth tables are constructed with atoms $s_1(\phi), s_2(\phi), \dots s_n(\phi)$ and $s_1(\omega), s_2(\omega), \dots s_n(\omega)$. By its definition, at least one $s_i(\phi)$ or $s_i(\omega)$ is true which is $\bigvee(s_i(\phi) \vee s_i(\omega))$.

- $A(\phi \sqcap \omega)$: Two truth tables are constructed with atoms $s_1(\phi)$, $s_2(\phi)$, … $s_n(\phi)$ and $s_1(\omega)$, $s_2(\omega)$, … $s_n(\omega)$. From the example above, we have the equivalence $A(\phi \sqcap \omega) \dashv\vdash A(\phi \sqcup \omega)$ & $E\phi$ & $E\omega$. From the case analysis above, we obtain

    1. $A(\phi \sqcup \omega)$: $\bigwedge(s_i(\phi) \vee s_i(\omega))$
    2. $E\phi$: $\bigvee s_i(\phi)$, and
    3. $E\omega$: $\bigvee s_i(\omega)$, such that
    4. $A(\phi \sqcap \omega)$ can be translated to $\bigwedge(s_i(\phi) \vee s_i(\omega)) \wedge \bigvee s_i(\phi) \wedge \bigvee s_i(\omega)$

- $E(\phi \sqcap \omega)$: Two truth tables are constructed with atoms $s_1(\phi)$, $s_2(\phi)$, … $s_n(\phi)$ and $s_1(\omega)$, $s_2(\omega)$, … $s_n(\omega)$. By its definition, $s_i(\phi)$ and $s_i(\omega)$ are true which is $\bigvee s_i(\phi) \wedge \bigvee s_i(\omega)$.

- $A(\phi \boxplus \omega)$ and $E(\phi \boxplus \omega)$ can be translated into $\sqcap$ or $\sqcup$.

- & and || operators are identical to $\wedge$ and $\vee$ in propositional logic.

The above clauses show that all Composite operators can be translated into FOL which has the evidence for soundness and completeness proofs. Consistency is automatically gained from the completeness property. For the rest of operators, the soundness and completeness can be explicitly recognised since they are from FOL and CTL which have the proof evidences. From this knowledge, it can be concluded that TS language is sound and complete.

4.6 Expressiveness of TS Language in Workflow Systems

For many TS formulas and their expressivity, it is possible to specify the following properties of workflow systems.

Self-complementary: We say that a service composition for a particular task is Self-complementary if one service possesses a certain set of properties and there must be another service possesses a complement set of properties. This is expressible by using $\Delta(z_1 \sqcap !\, z_1)$. This property is essential in specifying SoD.

Property-exclusive: If one property is present, there will not be other properties present in a workflow task. This property can be recognised by using $\boxplus$. For example, $\Delta(z_1 \boxplus ... \boxplus z_n)$.

Property Composition: There must be a certain set of properties present in executing a workflow task. This is expressible by using $\Delta(z_1 \sqcap ... \sqcap z_n)$.

Property Collaboration: This property is a special type of **Property Composition** describing there must be a certain set of properties for a task. A service that imposes this requirement to its partner is also part of the same task execution.

The expressiveness in the domain of CTL including Safety, Liveness, Non-blocking, and Strict and No Strict Sequence can also be recognised in service workflow systems.

Safety: This property guarantees that a bad thing will not happen. One notable safety property is mutual exclusion; that is, only one process is in its critical section at any time. This can also be explained in TS formulas by specifying that for all paths starting from an originating service, there are no services with properties s_1 and s_2 at the same time to be part of an execution flow. This is expressible as $\forall_t \Box \sim (s_1 \wedge s_2)$.

Liveness: This property guarantees that a good thing will eventually happen. In CTL one can easily express properties such as "for any state, if a process requests to enter the critical section, it will eventually be granted [180]". This property can also be realised in TS formulas. For example, one might specify that if a hotel booking is made $(\mathcal{P}A_{\text{HotelBooking}}(\varepsilon, \varepsilon, \varepsilon, \emptyset))$, it will eventually reach a payment service $(\mathcal{P}A_{\text{Payment}}(\varepsilon, S_P, \varepsilon, \emptyset))$. This can be expressible by using $\mathcal{H}(\mathcal{P}A_{\text{HotelBooking}}(\varepsilon, \varepsilon, \varepsilon, \emptyset) \rightarrow \forall_{\text{HotelBooking}} \Diamond \mathcal{P}A_{\text{Payment}}(\varepsilon, S_P, \varepsilon, \emptyset))$.

Non-blocking: This property states that for every state satisfying property s_1, there is a successor satisfying s_2. In service workflow systems, one might require that if the hotel request is made, there is a successor to go to booking services. This can be expressed by $\mathcal{H}(\mathcal{F}As_1 \rightarrow \forall \Diamond \mathcal{F}Es_2)$.

Strict and No Strict Sequence: This property is related to Enforcement of Sequences (R5). In some situations, security of workflow systems depends on a sequence of tasks and services. In either case, it can be accommodated by TS formulas.

4.7 Important Equivalences of TS Formulas

Two formulas φ and ω are equivalent for any workflow $\mathcal{M}$ (1) if $\mathcal{M} \vDash \varphi$ then $\mathcal{M} \vDash \omega$ and (2) if $\mathcal{M} \vDash \omega$ then $\mathcal{M} \vDash \varphi$. Some examples of equivalence are given below (Let Δ be used for $\mathcal{F}\mathrm{E}_\mathrm{t}, \mathcal{P}\mathrm{E}_\mathrm{t}, \mathcal{F}\mathrm{A}_\mathrm{t}$, or $\mathcal{P}\mathrm{A}_\mathrm{t}$, and ∇ for $\mathcal{H}$ or $\mathcal{B}$).

4.7.1 Direction Formulas

Below, the semantics equivalences of Direction operators are provided.

- $\sim\mathcal{H}\mathrm{r} \equiv \mathcal{H}\sim\mathrm{r}$
- $\sim\mathcal{B}\mathrm{r} \equiv \mathcal{B}\sim\mathrm{r}$

The negation has no effect on $\mathcal{H}$ and $\mathcal{B}$, which can be propagated to the inner components.

4.7.2 Path Formulas

These equivalences are the same as presented in CTL.

- $\nabla(\sim(\mathrm{r}_1 \wedge \mathrm{r}_2)) \equiv \nabla(\sim\mathrm{r}_1 \vee \sim\mathrm{r}_2)$
- $\nabla(\sim\exists_\mathrm{t}\odot\mathrm{r}) \equiv \nabla(\forall_\mathrm{t}\odot\sim\mathrm{r})$
- $\nabla(\sim\exists_\mathrm{t} \Diamond \mathrm{r}) \equiv \nabla(\forall_\mathrm{t}\Box\sim\mathrm{r})$
- $\nabla(\forall_\mathrm{t}\Box\mathrm{r}) \equiv \nabla(\sim\exists_\mathrm{t} \Diamond \sim\mathrm{r})$
- $\nabla(\forall_\mathrm{t}\Box\mathrm{r}) \equiv \nabla(\mathrm{r} \wedge \forall_\mathrm{t}\odot\forall_\mathrm{t}\Box\mathrm{r})$

4.7.3 Composite Formulas

According to the algebraic properties described earlier, important equivalences are given as follows.

- $\nabla(\Delta(\mathrm{z}_1 \sqcup !\,\mathrm{z}_1)) \equiv \mathrm{TRUE}$
- $\nabla(\Delta(\mathrm{z}_1 \sqcup \mathrm{z}_1)) \equiv \nabla(\Delta(\mathrm{z}_1))$
- $\nabla(\Delta(\mathrm{z}_1 \sqcap (\mathrm{z}_2 \sqcup \mathrm{z}_3))) \equiv \nabla(\Delta((\mathrm{z}_1 \sqcap \mathrm{z}_2) \sqcup (\mathrm{z}_1 \sqcap \mathrm{z}_3)))$
- $\nabla(\Delta(\mathrm{z}_1 \sqcup (\mathrm{z}_2 \sqcap \mathrm{z}_3))) \equiv \nabla(\Delta((\mathrm{z}_1 \sqcup \mathrm{z}_2) \sqcap (\mathrm{z}_1 \sqcup \mathrm{z}_3)))$

- $\nabla(\Delta(z_1 \sqcap (z_2 \boxplus z_3))) \equiv \nabla(\Delta((z_1 \sqcap z_2) \boxplus (z_1 \sqcap z_3)))$
- $\nabla \mathcal{P} A_t(z_1 \sqcap !\, z_1) \equiv \nabla \mathcal{P} E_t(z_1 \sqcap !\, z_1)$

It is worth noting that some of the none-equivalent formulas seem to be counter intuitive from those who experience in propositional logic. Some examples are given below.

- $\nabla(\Delta(z_1 \sqcap !\, z_1)) \not\equiv \text{FALSE}$. This is not evaluated to false as a part of a workflow example in Figure 4.2(a) shows that it satisfies the formula $\mathcal{BP} A_t(z_1 \sqcap !\, z_1)$.
- $\nabla\Delta(z_1 \sqcup (z_2 \boxplus z_3)) \not\equiv \nabla\Delta((z_1 \sqcup z_2) \boxplus (z_1 \sqcup z_3))$. This seems to be equivalent at the first glance; however, in the case when only Z_1 is present, the formula on the left hand side is satisfied, but not for the right (see Figure 4.2 (b)).
- $\nabla\Delta!\,(z_1 \sqcap z_2) \not\equiv \nabla\Delta(!\, z_1 \sqcup !\, z_2)$. De Morgan's law is not valid for $\sqcap$ and $\sqcup$ operators. The counter example is given in Figure 4.2 (c), where it satisfies $\mathcal{BP} A_t(!\, z_1 \sqcup !\, z_2)$, but not $\mathcal{BP} A_t!\,(z_1 \sqcap z_2)$.
- $\nabla\Delta(z_1 \boxplus z_2) \not\equiv \nabla\Delta((!\, z_1 \sqcap z_2) \sqcup (z_1 \sqcap !\, z_2))$ - Figure 4.2 (c) exemplifies this non-equivalence as it satisfies $\mathcal{BP} E_t((!\, z_1 \sqcap z_2) \sqcup (z_1 \sqcap !\, z_2))$, but not $\mathcal{BP} E_t(z_1 \boxplus z_2)$.

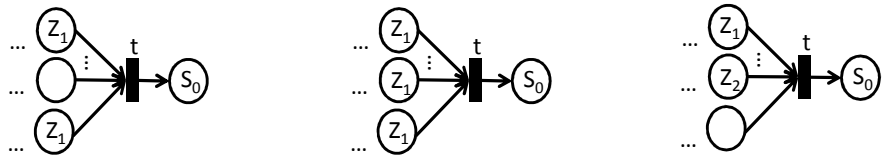


Figure 4.2: Examples of Non-equivalence Formulas

4.7.4 Formula Transformation

The property part of Composite formulas is represented by the combination of $\sqcup$, $\sqcap$ and $\boxplus$ operators. This section provides an example of $\boxplus$ and $\sqcap$ elimination. This would yield a great benefit for compliance checking algorithms to be presented in Chapter 5, so that a formula is composed of only $\sqcup$ operator. A set of algebraic properties described above are used for transformation given below.

$$\sim\mathcal{H}\exists_t \Diamond \left(\mathcal{P} E_t(Z_1 \sqcup (Z_2 \boxplus Z_3)\right)$$
$$\equiv \mathcal{H}\sim\exists_t \Diamond \left(\mathcal{P} E_t(Z_1 \sqcup (Z_2 \boxplus Z_3)\right)$$

$$\equiv \mathcal{H}\forall_t\Box\sim\left(\mathcal{P}\mathrm{E}_t(\mathrm{Z}_1 \sqcup (\mathrm{Z}_2 \boxplus \mathrm{Z}_3)\right)$$

- *Apply Breaking Association* $\sqcup$ *and* $\boxplus$ *property*

$$\equiv \mathcal{H}\forall_t\Box\sim(\mathcal{P}\mathrm{E}_t\mathrm{Z}_1 \;||\; (\mathcal{P}\mathrm{E}_t\mathrm{Z}_2 \;\&\; !\,\mathcal{P}\mathrm{E}_t\mathrm{Z}_3) \;||\; (!\,\mathcal{P}\mathrm{E}_t\mathrm{Z}_2 \;\&\; \mathcal{P}\mathrm{E}_t\mathrm{Z}_3))$$

Other examples of transformation, which is used for compliance checking, will be given with discussions in the next chapter.

4.8 Practical Patterns of TS Language

It is useful to look at TS formulas that might arise in verifying service properties in the real-world service workflows. The requirements from Chapter 1, Section 1.3, now can be expressed by TS formulas below.

- $\mathcal{H}\mathcal{F}\mathrm{A}_{\mathrm{weatherforecast}}(\mathrm{BBC}, \varepsilon, \varepsilon, \emptyset)$: Only services that belong to BBC are trusted before providing weather forecast information.
- $\mathcal{H}\mathcal{F}\mathrm{A}(\varepsilon, \varepsilon, \varepsilon, \{\mathrm{rep} \geq 0.8\})$: In order to share information, a node in a P2P network imposes a formula that all subsequent nodes must possess a reputation score not less than 0.8 (out of a possible 1).
- $\mathcal{H}\forall \Diamond \mathcal{P}\mathrm{A}_{\mathrm{payment}}(\varepsilon, \varepsilon, \varepsilon, \emptyset)$: In e-commerce, it is possible to get to a payment-type service. One example is that any booking service might impose this formula to ensure that once the booking is made; all paths must lead to the payment service eventually.
- $\mathcal{H}\mathcal{P}\mathrm{A}_{\mathrm{payment}}((\varepsilon, \varepsilon, \varepsilon, \{\mathrm{VISA}\}) \sqcup (\varepsilon, \varepsilon, \varepsilon, \{\mathrm{MASTER}\}))$: At payment, services must be either VISA or MASTER.
- $\mathcal{B}\forall \Diamond \mathcal{P}\mathrm{A}_{\mathrm{booking}}(\varepsilon, \varepsilon, \varepsilon, \emptyset)$: Booking services must be previously executed. In addtion to the above example, any payment service might impose this formula to ensure that booking has been made before executing payment.
- $\mathcal{H}\forall\left(\left[\mathcal{F}\mathrm{A}(\varepsilon, \varepsilon, \varepsilon, \{\mathrm{SSL}\}) \sqcup \mathcal{P}\mathrm{A}_{\mathrm{payment}}\ (\varepsilon, \varepsilon, \varepsilon, \varepsilon)\right]\right)$: A service might impose this formula to ensure that all subsequent services must operate on SSL until the payment task is reached.
- $\mathcal{H}\forall\Box\ \mathcal{F}\mathrm{A}(\varepsilon, \varepsilon, \varepsilon, \{\mathrm{Microsoft}\})$: In e-research, one might requires that all subsequent services are trusted if they belong to Microsoft. One example is that the University of

Oxford's Computer Science signs an agreement to provide research results only to Microsoft.

4.9 Subsets of SWN and Partially Satisfied

There is a high frequency of conflict occurrences in dynamic workflow interoperation, as TS formulas can be autonomously imposed and dynamically changed overtime. Typically, a single conflict could halt workflow executions and possibly leads to interoperation failures or changes to specification requirements. However, in some situations one conflict in a particular position does not affect some other parts of a workflow. Specifically, a conflict found in a workflow does not imply that all parts of a workflow encounter the same conflict, such that some executions may remain valid. In this situation, *partially satisfy* is a key element to address this situation, by indicating that there are some possible valid executions in the workflow. The definitions of *partially satisfy* and subset of a workflow are given in Appendix B.

4.10 Chapter Discussion

In this chapter, TS formulas have been comprehensively formulated by exploring its syntax and semantics. They are incrementally built up from Composite, Path, and Direction operators. The binding convention is described for the operator priorities and important semantics equivalences are provided to address the connections between each formula that have the same meaning. To be able to reason about a service workflow, *satisfiability* relations of each workflow connective are defined where the results of reasoning can be generated from the algorithms to be explained in Chapter 5. Several properties of TS formulas with the proofs of soundness, completeness, and consistency are presented, providing a strong foundation for reasoning.

At this stage, it can be observed that the semantics of Path formulas has no relation to the workflow connectives, which makes them unable to capture the difference between paths and instances quantification. Since instance quantification, or in other words, choices

of executing paths based on the receipt of both internal and external inputs, is dynamically determined during runtime, this implies that each execution may involve only some parts of SWN. This is due to the effect of OR and XOR connectives that allow the choices of paths to be executed. Modal logic operators, i.e., *possibility* and *necessity*, can be supplemented in augmenting the expressiveness of Path operators. The incorporation of modal logic to deal with the difference between path and instance quantification will be left for the future investigation.

Chapter 5

Compliance Checking Algorithms

5.1 Chapter Introduction

The core of the framework is the compliance checking algorithms between TS formulas and SWN. It takes an SWN and TS formulas as two inputs and returns the results of *satisfied*, *partially satisfied*, and *unsatisfied*, depending on the properties of services and tasks which constitute evidences that the TS formulas comply, partly comply, or do not comply with the SWN. Rather than to detect and decrease the amount of errors in complex systems as the original model checker aims for, the algorithms in this chapter are developed to verify TS formulas to ensure that services in a given SWN, or the SWN itself, possess required properties. Since the number of services in a workflow is comparatively much smaller than states in complex software systems, we do not suffer from the state explosion problem at this moment. Based on this characteristic, the reasoning algorithms are decidable and deterministic; offering guaranteed results.

This chapter is divided into three major parts. The first part describes the compliance checking algorithms for Composite formulas, where the result will be used in the algorithms for Path and then Direction formulas presented in the second and third parts, respectively. Before detailing the specific information about the algorithms, essential compliance checking criteria and related works are provided.

In large-scale open environments where service interoperation is highly dynamic, automating compliance checking is needed to support scalability, dynamicity and security. To achieve this, three basic elements are required: (1) a modelling language in which service workflows can be logically abstracted to represent structure of services, tasks, and their dependency, (2) a specification language as a formal and uniformed representation of requirements, and (3) compliance checking algorithms to ascertain that services, or workflows, satisfy requirements [164]. The first two elements have been comprehensively addressed in Chapter 3 and 4 where the workflow to be verified is modelled by SWN, and requirements are expressed by TS formulas. This chapter underlies the third part focusing on compliance checking algorithms. Since TS formulas are rooted in logic, several existing techniques and analysis can be exploited to automatically verify properties of an SWN. This is important to support scalability in the way to eliminate manually checking for the compliance of requirements. According to [163], these algorithms are positioned among the following criteria.

Proof- vs. Model-based: There are two different aspects in which a given formula can be verified. First, the proof-based approach incorporates two sets of formulas; one is used to model a system and another is used for the specification of required properties. However, this technique requires a large amount of human interventions, which consequently introducing scalability problems. In the model-based approach, on the other hand, the system is represented by a model and the specification is formulated in the form of logical languages, such that we have a model that can be automatically checked whether a formula is true, or not, for that model. Our algorithms are classified into the model-based approach.

High- vs Low Degree of automation: High-degree automation is desired to deal with dynamicity of service workflows exhibited in large-scale open environments.

Full- vs. Property-verification: Our algorithms do not intend to verify a single property or the whole behaviour of a system. Instead, they focus on property-verification of trust-related compliance checking whether a given TS formula can satisfy an SWN.

Intended Domain of Application: Our TS language and reasoning algorithms are applied to the service workflow domain.

Pre- and Post-development: Despite yielding a great performance advantage if reasoning processes can be fully done during workflow formation (pre-development), this is impractical when changes can be made dynamically during workflow executions. Our algorithms focus on both pre- and post-development which are repeatedly run to check for further compliance whenever changes are detected.

One factor to the success of service workflow interoperation in large-scale environments relies on the degree of automation. It is needed to assist and replace several iterative time-consuming activities, usually related to human interventions; and to accommodate dynamic changes of requirements. Although some of our reasoning processes can never be automatic as some actions requires human's intervention, for example, customising (relaxing or restricting) existing TS formulas to match current needs, automatic compliance checking is still a key component to escalate efficiency and scalability, and simultaneously improve security overall.

The remaining of this chapter continues with background materials and related work including model checking and logical languages, compliance checking, and reasoning algorithms. In Section 5.3, Existing data structures are reviewed and a new technique is invented which can be used for Composite compliance checking, which include (1) Constrained Truth Table (CTT) [165] (2) Exclusive Disjunctive Normal Form (EDNF) [166], and (3) Reduced Constrained Order Binary Decision Diagrams (RCOBDDs). In this section, we propose a special data type, RCOBDDs, which in theory makes the performance of Composite compliance checking more efficient than CTT and EDNF. Based on this result, the algorithms for Path and Direction parts are proposed in Section 5.4 and Section 5.5, respectively. We extend the existing labelling algorithm [163] to deal with constrained tasks (annotated in Path Quantifiers $\forall_t$ and $\exists_t$). How the algorithms run to obtain *satisfied*, *partially satisfied*, or *unsatisfied* is presented in Section 5.6. Finally, the chapter is concluded in section 5.7 with directions for future research.

5.2 Background

This chapter is related to two research areas: (1) model checking and logical languages, and (2) reasoning algorithms.

5.2.1 Model Checking and Specification Languages

Model checking is first introduced by Clarke and Emerson ([167] and [168]). Its original applications are on sequential circuit and communication protocols to verify specific properties of a system. Specifications are expressed by logical formulas and systems to be verified are modelled by state-transition diagrams. The early advantage of the model checking is to verify the correctness of hardware or software, or combination. Currently, this becomes a standard in designing systems and its usability is visualised in industrial and commercial sectors as a key component in routinely testing products before releasing. Much of the success of model checking is due to the fact that it is a fully automatic verification method.

It is desirable to have a model checking to traverse all reachable states. However, at a present time, the number of states that can be represented explicitly is approximately 10^6 [164] and some systems can range over 10^{100} states. This has become a well-known state explosion problem to model checkers that they are unable to guarantee the result of large systems. Because of this incapability, several techniques and proof theories ([169] and [170]) have been improved. For example, a symbolic technique by using Binary Decision Diagrams (BDDs), or Ordered Binary Decision Diagrams (OBDDs) in their canonical forms, bounded model checking, partial order reduction technique, probabilistic model checking, and heuristic search, are developed to tackle and alleviate this problem. Especially, the use of OBDDs in symbolic model checking is a breakthrough that has dramatically changed the view in applying formal verification in practices. Many techniques ([171] and [172]) are introduced in conjunction with this approach to increase the size of the verified systems and improve efficiency. As stated, reasoning in model checking cannot be possible without a model abstraction. Mostly in computer science, a model is presented by a semantical (finite Kripke structure) model known as state-transition systems.

Recently, model checking has been extended to cover wider domains beyond the specific systems modelled by state-transition diagrams. It has inspired several variants of compliance checking that spreads across many domains ranging from verification of business processes ([140], [173], and [174]), policy-based compliance checking for trust [175], logic-based verification [194] to hardware and software component testing at very low level. In this essence, the concept of model checking is applied to the compliance checking in the service workflow domain.

5.2.2 Reasoning Algorithms

In computer science, one of the earliest works of reasoning involves FOL to find possible combinations in which a formula is evaluated to true (or false). One popular technique is to translate a propositional formula into a parse tree that yields three possible outcomes: (1) validity, a formula is evaluated to true in all possible assignments, (2) satisfiability, a formula is evaluated to true in some assignments, and (3) invalidity, a formula is evaluated to false in all assignments. A significant amount of works motivated by this technique has been proposed. For example, a liner SAT solver [176], one of the earliest implementations using this technique, converts a formula into Conjunctive Normal Form (CNF) to construct a parse tree and employs a formal set of rules to check for satisfiability. Cubic solver (section 1.6.2 in [163]) improves the classic SAT solver by introducing the temporary marking technique, and afterwards checking for marking consistency when all atoms are finally marked. Alloy [177] exploits state machines to specify and explores a model pertaining to properties written in FOL.

One major limitation of FOL is that it cannot specify a certain property on every, or some, future state of execution sequences. Therefore, FOL is enhanced to be more expressive with quantifiers over paths of executions, known as second-order logic. CTL is a special type of the second-order logics that is able to formulise paths and events in time. For checking validity of CTL, the labelling algorithm [163] and NuSMV [178] are prominent. The labelling algorithm is developed on the basis of sequences of points where each point determines validity of a propositional part of a formula whereas NuSMV is developed based on the symbolic model checking method based on OBDDs.

Representing a workflow system in a way to reason requires a specific representation. SWN is used to abstract the information of tasks and associated services. Since service workflows usually exhibit in open environments, changes can be periodically and unpredictably made. Finding efficient algorithms for compliance checking in this context is unique and yields a great contribution. Chapter 4 has pointed out several theoretical concepts and semantics between TS operators and workflow connectives. This chapter focuses on the algorithms of how an SWN can be verified by TS formulas.

5.3 Part I: Composite Formulas, Boolean Representations, Compliance Checking Algorithms, and Comparisons

This section describes the compliance checking algorithms for Composite formulas. We first review the different forms of Boolean representation of workflow formulas including CTT [165] and EDNF [166]. Then, the RCOBDDs data structure is introduced. Performance evaluation of these forms is also compared at the end of this section.

5.3.1 Pre-processing TS Formulas

It is useful to translate any TS formula into a simpler form which benefits the process of the compliance checking. The algebraic properties presented in Chapter 4, Section 4.5, are essential ingredients such that the Property part of a Composite formula can be pre-processed until it contains only $\sqcup$ and $\sqcap$, and ultimately contains only $\sqcup$. A tree structure for the transformed formula is illustrated in a strict order of $\sqcup$ and $\sqcap$. The notions of clove and clove tree of a formula are defined below.

Def. 5.1 (Clove): Given a Composite formula and let q_i be (s, t, o, A), a clove is defined as a set of atomic propositions q_i linked by $\sqcap$, or a single atomic proposition if no such operator is presented.

Def. 5.2 (Clove Tree): A clove tree is the representation of a transformed formula with the Quantifier part as a root, the second level is $\sqcup$, and the leaves are comprised of cloves.

Below, the example of the transformation is presented. The formula $\mathcal{P}A_t(q_1 \sqcup (q_2 \boxplus (q_3 \sqcap q_4))$ can be interpreted into four sub-trees (see Figure 5.1). For the ease of reading,

the Direction and Composite operators, $\mathcal{B}$ and $\mathcal{P}$, in this formula are not shown. The transformation example is described below.

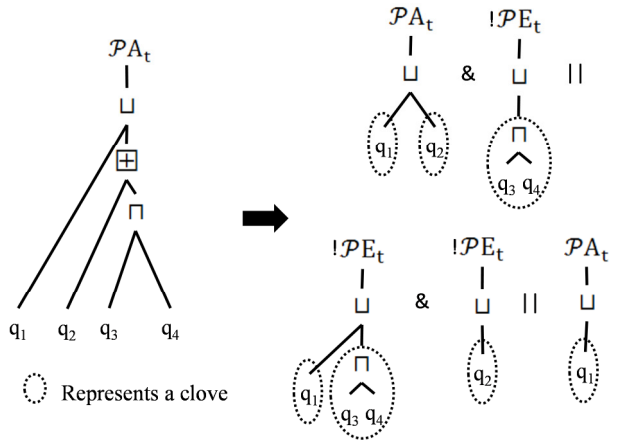
- *Initial Form*

$$\mathrm{A_t}(\mathrm{q_1} \sqcup (\mathrm{q_2} \boxplus (\mathrm{q_3} \sqcap \mathrm{q_4}))$$

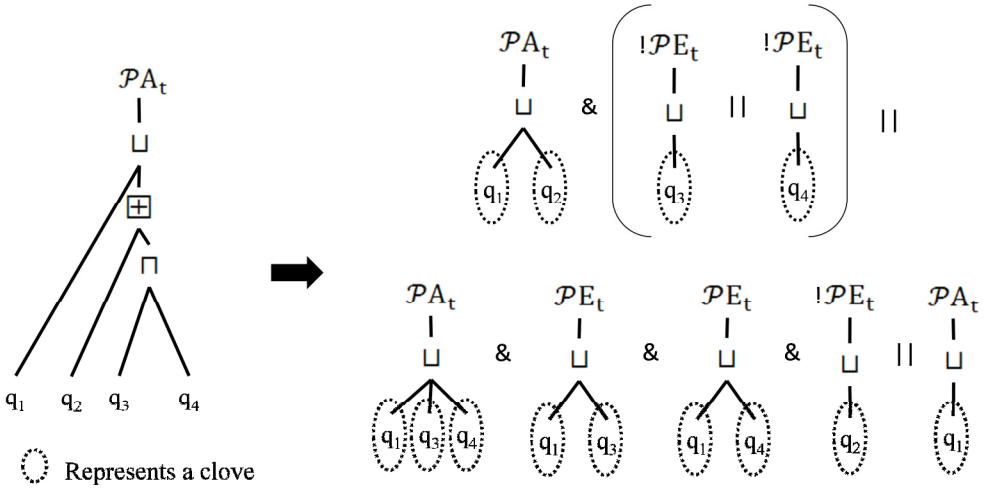
- *Applying Breaking Association* $\sqcup$ *and* $\boxplus$

$$\mathrm{A_t}(\mathrm{q_1} \sqcup \mathrm{q_2}) \,\&\, !\,\mathrm{E_t}(\mathrm{q_3} \sqcap \mathrm{q_4}) \,||\, \mathrm{A_t}\big(\mathrm{q_1} \sqcup (\mathrm{q_3} \sqcap \mathrm{q_4})\big) \,\&\, !\,\mathrm{E_t}\mathrm{q_2} \,||\, \mathrm{A_t}\mathrm{q_1}$$

At this step, there are five clove trees where the smallest components are cloves. In the following steps, further transformation is applied such that the final result will contain only atomic proposition cloves (see Figure 5.1 (b)).



(a) Transformation into Cloves



(b) Full Transformation that Consists of only Atomic Cloves

Figure 5.1: The Tree Structure of $\mathcal{B}\mathcal{P}\mathrm{A_t}(q_1 \sqcup (q_2 \boxplus (q_3 \sqcap q_4))$

- *Applying Distributive Property*

$$A_t(q_1 \sqcup q_2)\ \&\ !\,E_t(q_3 \sqcap q_4)\ ||\ A_t((q_1 \sqcup q_3) \sqcap (q_1 \sqcup q_4))\ \&\ !\,E_t q_2\ ||\ A_t q_1$$

- *Applying ⊓ Elimination*

$$A_t(q_1 \sqcup q_2)\ \&\ !\,(E_t q_3\ \&\ E_t q_4)\ ||\ A_t((q_1 \sqcup q_3) \sqcup (q_1 \sqcup q_4))\ \&\ E_t(q_1 \sqcup q_3)\ \&\ E_t(q_1 \sqcup q_4)\ \&\ !\,E_t q_2\ ||\ A_t q_1$$

- *Regrouping*

$$A_t(q_1 \sqcup q_2)\ \&\ !\,(E_t q_3\ \&\ E_t q_4)\ ||\ A_t(q_1 \sqcup q_3 \sqcup q_4)\ \&\ E_t(q_1 \sqcup q_3)\ \&\ E_t(q_1 \sqcup q_4)\ \&\ !\,E_t q_2\ ||\ A_t q_1$$

- *Applying De Morgan's Rule*

$$A_t(q_1 \sqcup q_2)\ \&\ (!\,E_t q_3\ ||\ !\,E_t q_4)\ ||\ A_t(q_1 \sqcup q_3 \sqcup q_4)\ \&\ E_t(q_1 \sqcup q_3)\ \&\ E_t(q_1 \sqcup q_4)\ \&\ !\,E_t q_2\ ||\ A_t q_1$$

This transformation makes the algorithms for compliance checking simpler. This is because the absence of ⊞ allows us to circumvent the check between ⊞ and workflow connectives. Based on this transformation, Boolean representation and reasoning algorithms are presented below.

5.3.2 CTT

The CTT technique in [165] employs a basic truth table to represent possible truth assignments of a Boolean function with the application of the reduction rules, described in Section 3.5.2, Chapter 3, to express a workflow formula. As a result, a CTT contains only the possible ways of workflow executions. The example of a CTT representation for a formula A XOR (B OR (C AND D)) is shown in Tables 5.1 and the algorithm is presented in Figure 5.2. For the full explanation, please refer to [165].

The benefit of using CTT is that the compliance checking is very simple. However, it suffers from the space inefficiency as the maximum number of rows is exponential to the number of variables (services for a task in an SWN). This happens when only OR connectives are involved in the formula which creates $2^n - 1$ many lines (-1 is the line that all variables contain 0 marking). In terms of performance, assume that the checking

operation between a clove and a variable occupies one time unit, the efficiency is evaluated to $O(|C||T||V|2^{|V|})$ in terms of time and $O(|C| + |V|2^{|V|} + |T|)$ in terms of space, where $|C|$, $|T|$ and $|V|$ are the number of cloves, clove trees, and workflow variables, respectively.

Table 5.1: Constrained Truth Table for A XOR (B OR (C AND D))

No	A	B	C	D	ω =(C AND D)	α =(B OR ω)	ψ =(A XOR α)
1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0
3	0	0	1	0	0	0	0
4	0	0	1	1	1	1	1
5	0	1	0	0	0	1	1
6	0	1	0	1	0	1	1
7	0	1	1	0	0	1	1
8	0	1	1	1	1	1	1
9	1	0	0	0	0	0	1
10	1	0	0	1	0	0	1
11	1	0	1	0	0	0	1
12	1	0	1	1	1	1	0
13	1	1	0	0	0	1	0
14	1	1	0	1	0	1	0
15	1	1	1	0	0	1	0
16	1	1	1	1	1	1	0

```
1.  Function CTTSat(CTT, V)  // CTT.R = a set of rows and V= a set of workflow variables
2.  Begin
3.      For each r_i ∈ CTT. R
4.          For each v ∈ V
5.              If A_t is presented,
6.                  If for all v ∈ V that are marked with 1 ⊨ any clove
7.                      r_i = satisfied;
8.                  End if
9.              End if
10.             If E_t is presented,
11.                 If for some v ∈ V that are marked with 1 ⊨ any clove
12.                     r_i = satisfied;
13.                 End if
14.             End if
15.         End for
16.     End for
17.     If for all r_i = satisfied
18.         return satisfied;
19.     Else if some r_i = satisfied
20.         return partially satisfied;
21.     Else
22.         return unsatisfied;
23.     End if
24. End
```

Figure. 5.2: Pseudo Code for CTTSat Algorithm

5.3.3 EDNF

Normal form is an alternative choice for Boolean representation in a more concise form than CTT. A formula with the same number of variables is much more compact. In this section, the algorithms for EDNF from [166] provide better performance than CTTSat. Furthermore, when the answer is *partially satisfied,* subsets of an SWN, described in Chapter 4, Section 4.9, (see **Def. B.2** in Appendix B) can be easily determined by selecting only the terms that are evaluated to true. The *terms* in EDNF are the variables connected by AND connectives, and XOR connectives are used to connect between such *terms*. If all *terms* are true, the result is *satisfied*. If only some *terms* are true, the result is *partially satisfied*; otherwise, it is *unsatisfied*.

The complexity of EDNF algorithms depends on (1) the number of the occurrence of workflow variables (services), (2) the number of connectives, and (3) reasoning algorithms. In [166], a binary tree is employed to represent a workflow formula. Leaf nodes are workflow variables while the upper nodes are workflow connectives. The graphical explanation is illustrated in Figure 5.3. Pseudo code for EDNFSat is presented in Figure 5.4. For full detailed operations, please refer to [166]).

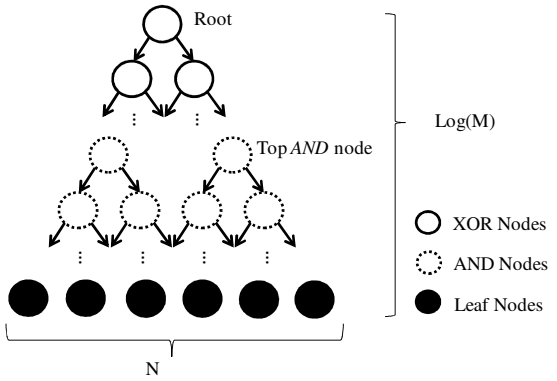


Figure 5.3: Graphical Representation of Efficiency Complexity Calculation

1. **Function** EDNFSat(SWND) // SWND is a EDNF tree representing workflow formulas
2. Q = a set of properties of unknown status to satisfy any clove;
3. V = SWND.V // a set of workflow variables;
4. M = all presented connectives;
5. **Begin**
6. **For each** $v_i \in V$,
7. **If** $v_i \vDash$ any clove

```
8.              v_i = satisfied;
9.          Else if v_i partly complies with any clove
10.             v_i = unknown_Q;
11.         Else  v_i = unstisfied;
12.         End if
13.     End for
14.     For each m_i ∈ M and m_i = AND  //Assume m_i is chosen in order from low-to-high of the  SWND
15.         If A_t is presented
16.             If (m_i.left = satisfied AND m_i.right = satisfied) OR (m_i.left = unknown_Q AND
17.             (m_i.right = satisfied) OR (m_i.left = satisfied ANDm_i.right = unknown_Q)
18.                 m_i = satisfied;
19.             Else If (m_i.left = unknown_Q AND m_i.right = unknown_Q)
20.                 m_i = satisfied;
21.             Else  m_i = unsatisfied;
22.             End if
23.             If m_i is TOP AND NODE
24.                 If m_i = unknown_Q
25.                     m_i = unsatisfied;
26.                 End if
27.             End if
28.         End if
29.         If E_t is presented
30.             If  m_i.left = satisfied OR m_i.right = satisfied
31.                 m_i = satisfied;
32.             Else If (m_i.left = unknown_Q AND m_i.right = unknown_Q)
33.                 m_i = satisfied;
34.             Else If (m_i.left = unknown_Q OR m_i.right = unknown_Q)
35.                 m_i = unknown_Q;
36.             Else  m_i = unsatisfied;
37.             End if
38.             If m_i is TOP AND NODE
39.                 If m_i = unknown_Q
40.                     m_i = unsatisfied;
41.                 End if
42.             End if
43.         End if
44.     End for
45.     For each m_i ∈ M and m_i = XOR  ////Assume m_i is chosen in order from low-to-high layer of SWND
46.         If  m_i.left = satisfied AND m_i.right = satisfied
47.             m_i = satisfied;
48.         Else If  (m_i.left = unsatisfied & m_i.right = unsatisfied)
49.             m_i = unsatisfied;
50.         Else  m_i = partially satisfied;
51.         End if
52.     End for
53. End
```

Figure 5.4: Pseudo Code for Algorithm EDNFSat

Efficiency of EDNFSat can be determined as follows. Assume again the checking operation between a clove and a leaf node occupies one time unit, the efficiency evaluation of this form is $O(|C||T||V||M|)$ where $|C|$, $|T|$, $|V|$, and $|M|$ are the number of cloves, clove tress, workflow variable, and workflow connectives respectively. In the worst case, the maximum number of occurrence M is $2^{|V|}$, such that, the computational complexity of this form is $O(|C||T||V|2^{|V|})$. However, it is important to look at the average occurrence of V that can be computed as the following equation. The proof is provided in Appendix C.

$$\text{average} = E(n \times \binom{n}{n} + (n-1) \times \binom{n}{n-1} + \cdots + 0 \times \binom{n}{0})$$

$$\text{average} = \frac{n}{2}$$

Therefore, the average time complexity of EDNFSat is $O(|C||T||V|^2)$.

5.3.4 RCOBDDs

Inspired by OBDDs, we propose RCOBDDs, the reduced from of COBDDs, that are used for more efficient compliance checking than CTTSat and EDNFSat. OBDDs are notable for the substantially more compact and canonical form of representing a Boolean formula. We exploit this advantage by converting a workflow formula into an RCOBDD from which we can obtain a relatively more concise representation of such formulas than CTT and EDNF. Before describing RCOBDDs in detail, the definitions of COBDDs are first given.

5.3.4.1 COBDDs Definitions and Theorems

COBDDs are similar to OBDDs except that impossible executions are ignored as to avoid invalid executions. According to the reduction rules in Chapter 3, Section 3.6.2, the impossible executions, any assignment that evaluates a workflow formula to false, are excluded from the traditional OBDDs. The reduction process is simple by eliminating the false node and the links connected to that node. This section explains how a workflow formula can be encoded by COBDD.

Def. 5.3 (COBDDs): Boolean variables x and y representing services in an SWN are variables that can be evaluated into true or false. According to the reduction rules, we define the following functions on a set $\{\text{true}, \text{false}\}$ restricted by.

1. $x = \text{true}$. Rule 3 restricts that there will be no false node occurred in COBDDs.
2. $x \text{ AND } y = \text{true}$ if x and y are true, and $x \text{ AND } y = \text{false}$ if x and y are both false. Rule 1 restricts that x and y must both be true or false.
3. $x \text{ XOR } y = \text{true}$ if exactly one of x or y is true, and $x \text{ XOR } y = \text{false}$ if x and y are both false. Rule 2 restricts that x and y cannot both be true.
4. $x \text{ OR } y = \text{true}$ if one or both of x and y is true, and $x \text{ OR } y = \text{false}$ if x and y are both false.

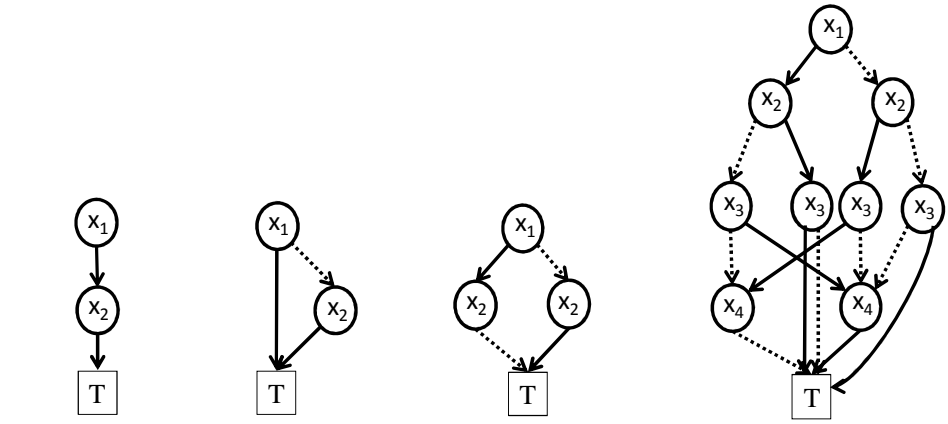
The false node in COBDDs is excluded as impossible executions are not part of this concern. As a result, the tree contains only the true node where the way from root to this node represents the possible ways of execution. This is done by the same reason in CTT. The definition of COBDD tree is described below.

Def. 5.4 (COBDD Tree): A COBDD Tree is a tuple (N, RL, RH, x_1^1) where N is a set of nodes, RL is a set of relations with false assignment, RH is a set of relations with true assignment, and x_1^1 (or x_1 in Figure 5.5) is a root node.

1. $C = (N, RL, RH, x_1^1)$.
2. $N = \{x_1^1, \dots, x_n^m\} \cup \{T\}$. N is a set of nodes. x_i^m represents the variable x_i where m indicates the number of occurrence of x_i. The node T is the true node.
3. x_1^1 is a root node. There is only one root node in a COBDD tree.
4. RL is a set of relation between two nodes connected by the dash line indicating false assignment.
5. RH is a set of relation between two nodes connected by the solid line indicating true assignment.
6. $lo: N \times N$. $lo(x)$ is a function that its result is a node pointed via a dash line from x where $(x, lo(x)) \in RL$.
7. $hi: N \times N$. $hi(x)$ is a function that its result is a node pointed via a solid line from x where $(x, hi(x)) \in RH$.

8. If x is a terminal node, either $\mathrm{lo}(x)$ or $\mathrm{hi}(x)$, or both, is connected to a T node.
9. The lowest terminal nodes are located at the lowest level in a tree. For example, for any $\mathrm{hi}(x_i) = \mathrm{true}$ and $\mathrm{hi}(x_j) = \mathrm{true}$, x_i is the lowest terminal node if $i > j$.

A Boolean function in the COBDD form with n arguments is a function from $\{\mathrm{true}, \mathrm{false}\}^n$ to {T}. Some examples of COBDD trees are illustrated in Figure 5.5(a)-(d) where x_1 is a root node and x_2 and x_4 are the lowest terminal nodes in Figure 5.5(a)-(c) and 5.5(d) respectively.



(a) $x_1 \mathrm{AND} x_2$ (b) $x_1 \mathrm{OR} x_2$ (c) $x_1 \mathrm{XOR}\ x_2$ (d) Combinations

Figure 5.5: Examples of COBDD Trees

It is important to note that the second level node of x_3 in Figure 5.5(d) has both dash and solid lines pointing to T. Normally, this type of node is not appeared in the original OBDDs as the presence of x_3 does not affect evaluation. However, instead of concerning the evaluation of truth values of a formula, workflow formulas are the way to represent every possible way of executions. As a result, this node must be kept in the tree; otherwise the information that x_3 might be part of some executions is missing. For instance, one execution involves services x_1, x_2, and x_3 and another involves services x_1 and x_2. There are a few theorems of COBDDs where the proofs can be found in Appendix D.

In order to develop efficient algorithms, we present a reduced form of COBDDs, namely RCOBDDs. This data structure yields an important benefit since the number of node is linear to the number of workflow variables (see the proof in Appendix D).

Def. 5.5 (RCOBDD Tree): An RCOBDD tree is a topology sorted type of DAG with only one starting and ending nodes. It represents all possible executions regarding services involved for each workflow task. There are two types of nodes: non-variable nodes including one root (starting) node x_0 and one labelled (ending) node T (the true node); and variable nodes representing SWN services occurred along the way from x_0 to T. In this reduced form, an original COBDD tree is compressed. All variables are ordered without duplication where each appears only once in a tree. Any link that connects two variable nodes indicates that they are connected by AND, and many links leaving from a node represents XOR characteristics. Figure 5.6 depicts some examples of RCOBDD trees. An RCOBDD tree is two-tuple $RC = (N, R)$.

1. $V =$ a set of workflow variables $\{x_1, \dots, x_n\}$.
2. $N = \{x_0\} \cup \{T\} \cup V$ where N is a set of nodes including the starting node x_0, ending node T, and variable node V.
3. $R =$ a set of relation between two variables. For example, (x_1, x_2) means that there is a directed link from x_1 to x_2.
4. $r: N \times N$. $r(x_i)$ is a function that its result is a set of nodes pointed via a link from x_i.

Def. 5.6 (RCOBDD Ordering): Let $[x_1, \dots, x_n]$ be an ordered list of variables without duplications. An RCOBDD tree has the ordering $[x_1, \dots, x_n]$ if all variables occur in order according to the list. For every occurrence of x_i followed by x_j along any path in a tree, we have $i < j$.

A COBDD tree that represents the interpretation of the possible valuations of SWN variables into {T} can be compressed into RCOBDDs. The advantage is that the number of nodes is linear to the number of variables which is much smaller than COBDDs. Efficiency is gained by sharing variable nodes and by elimination unnecessary nodes. However, as a node can have many links to other subsequent nodes, there can be as many as $\frac{|V|(|V|-1)}{2}$ links in one RCOBDD tree or $O(|V|^2)$ in terms of efficiency where V is the number of SWN variables (The transformation algorithm from COBDDs to RCOBDDs can also be found in Appendix D).

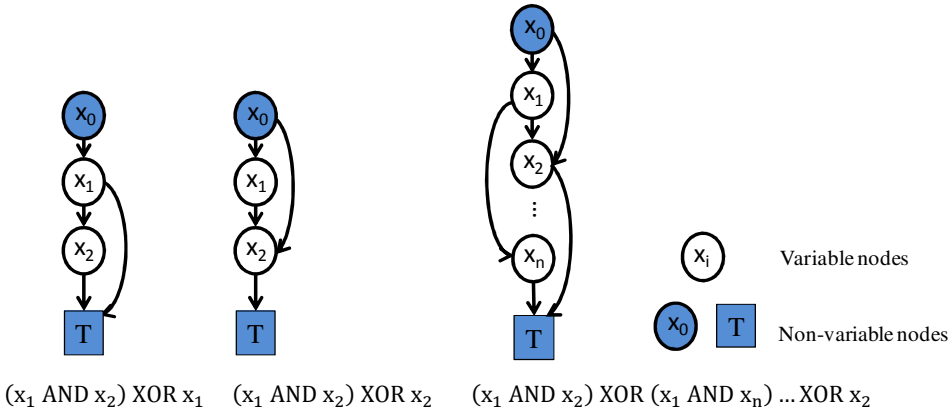


Figure 5.6: Some Examples of Reduced COBDD Trees

5.3.4.2 Algorithms for RCOBDDs

With the benefit of an RCOBDD tree where the number of node is linear to the number of SWN variables, efficient algorithms can be devised. In this section, we propose two classes of algorithms to deal with Composite formulas: (1) the first class is the algorithms when logical operators (& and ||) are not presented, which implies that a Composite formula has only one clove tree, and (2) the second class is developed to deal with all types of operators. The reason is the first class can exploit efficiency from RCOBDDs, such that time complexity is reduced into polynomial time. In these algorithms, TS formulas must be fully transformed, such that it contains only the atomic type of cloves.

5.3.4.2.1 1st Class Algorithms

In the 1st class algorithms, there are two parts of algorithms to specifically deal with E_t and A_t. The core function in these algorithms is the reachability query in DAG denoted as $u < v$. When we say that v is reachable from u, there must be at least one path from u to v. The following paragraph summarises the works in this area.

Much research has been done to tell whether a vertex u can reach another vertex v in a DAG ([179] and [180]). Here, we assume that n is the number of vertices (n = |V|) and m is the number of edges (m = |E|). Many existing reachability algorithms require pre-processing. The very first implementation by Warren used matrix multiplication (MM for short) [181] to obtain a transitive closure set. The reachability query is done in a constant time $O(1)$. However, the time for constructing indices for transitive closure is $O(nm)$ or

$O(n^3)$ where the number of edges m can be maximally n^2. There are other important techniques including Graph Labelling, DAG Decomposition [182], and Tree Encoding [183]. For the latter case, resolving a single reachability query is linear to the number of edges. Table 5.2 summarises the computational complexity of these algorithms [180].

Table 5.2: Complexity Comparison of Reachability Query Approaches

Models	Query time	Labelling time	Space Overhead	
Graph-traversal	$O(e)$	0	0	
DAG Decomposition	$O(k)$	$O(mn)$	$O(kn)$	k is the number of chains
Tree Encoding	$O(n)$	$O(mn)$	$O(n^2)$	
2-hop	$O(m^{1/2})$	$O(n^4)$	$O(nm^{1/2})$	
Dual Labelling	$O(1)$	$O(m + n + t^3)$	$O(n + t^2)$	t is the number of non-tree edges
Matrix-multiplication	$O(1)$	$O(n^3)$	$O(n^2)$	

In our respect, time spending for constructing indices becomes significant in compliance checking. This is because the dynamic changes of a workflow lead to the consequent effect on structuring the index that requires to be rebuilt according to each change. To avoid this high overhead, the following algorithms are devised without pre-processing. In what follows, algorithm ForSomeSat1 and ForAllSat1 are developed according to E_t and A_t.

1. **Function** ForSomeSat1(V, C) // V = a set of workflow variables and C = a set of cloves
2. **Begin**
3. $V' = \{\}$ // a set of nodes with satisfied marking
4. **For each** $v_i \in V$,
5. **If** $v_i \vDash$ any clove in C
6. v_i = satisfied;
7. Add(v_i, V'); // Add he node v_i to the set V'
8. **End if**
9. **End for**
10. $V'' = V - V'$ // V'' = Nodes that are marked with unsatisfied
11. **If** ($V' == V$) or (reachable(x_0, x_T, V'') == false) // Nodes marked with satisfied
12. **return** satisfied;
13. **Else if** $V' \neq \emptyset$
14. **return** partially satisfied;
15. **Else**
16. **return** unsatisfied;
17. **End if**
18. **End**

Figure 5.7: Pseudo Code for Algorithm ForSomeSat1

Algorithm $ForSomeSat1$

This algorithm is developed to deal with E_t. Assume that an RCOBDD tree has an ordering $[x_1, x_2, \dots, x_n]$. The operations are described as follows (see Figure 5.7).

- Suppose we have an RCOBDD tree of a formula $E_t(\phi_1 \sqcup \dots \sqcup \phi_n)$: Label x_i with *satisfied* if $\phi_i(x_i)$ is true.
    - If all x_i are marked, this formula is *satisfied*.
    - If some x_i are marked,
        a. If T is reachable from x_0 and each x_i is not marked, this formula is *partially satisfied*.
        b. Otherwise, the formula is marked with *satisfied* because for all executions (paths) there will be some x_i such that $\phi_i(x_i)$ is true.
    - Otherwise, the formula is marked with *unsatisfied*.

```
1.  Function ForAllSat1(V, C)  // V = a set of workflow variables and C = a set of cloves
2.  Begin
3.      V′ = { }        // a set of nodes with satisfied marking
4.      For each v_i ∈ V,
5.          If v_i ⊨ any clove in C
6.              v_i = satisfied;
7.              Add(v_i, V′);          // Insert node v_i to the set V′
8.          End if
9.      End for
10.     If (V′ == V)        // Nodes that are not marked
11.         return satisfied;
12.     Else if reachable(x_0, x_T, V′) == true
13.         return partially satisfied;
14.     Else
15.         return unsatisfied;
16.     End if
17. End
```

Figure 5.8: Pseudo Code for Algorithm ForAllSat1

Algorithm $ForAllSat1$

This algorithm is developed for A_t. The ordering of an RCOBDD tree is the same as above. The operations are described as follows (see Figure 5.8).

- Suppose we have an RCOBDD tree of a formula $A_t(\phi_1 \sqcup \dots \sqcup \phi_n)$: Label x_i with *satisfied* if $\phi_i(x_i)$ is true.

- If all x_i are marked, this formula is *satisfied*.
- If some x_i are marked,
    a. If T is reachable from x_0 and each x_i is marked with *satisfied*, this formula is *partially satisfied*.
    b. Otherwise, the formula is marked with *unsatisfied* because for all executions (paths) there will not be all x_i where $\phi_i(x_i)$ is true.
- Otherwise, the formula is marked with *unsatisfied*.

```
1.  Function reachable(x0, T, V)
2.  Begin
3.      V' = { x0}        // a set of currently reachable nodes, a starting node
4.      V = V − V'
5.      rn.R = x0.R  //Obtain a set of immediately reachable nodes from x0
6.      While V ≠ ∅
7.          If rn.R − V' = ∅,  return false;   // This means that there is no other reachable nodes
8.          V' = rn.R ∪ V';   // Include all immediately reachable nodes in the set V'
9.          If  T ∈ V',  return true;
10.         For each xj in V'
11.             R = R ∪ (xj.R − V') ;    //Include all adjacent nodes into the set R
12.         End for
13.         rn.R = R;
14.         V = V − V';        // Eliminating the already reachable nodes from V
15.     End while
16.     return false;
17. End
```

Figure 5.9: Pseudo Code for Algorithm reachable

Algorithm Reachable

Here, the $O(n^2)$ time algorithm for reachability query in RCOBDDs is proposed. The operations are described by the following steps (see Figure 5.9).

1. Let $V = \{x_1, x_2, \ldots, x_n\} \cup \{x_n, T\}$, V' be a set of reachable nodes starting from x_0, $x_i.R$ be the set of nodes that is directly connected by x_i, and T be an ending node.
2. V' is initialised as $V' = \{x_0\}$ and $V = V - V'$.
3. V' is then converted to be a representative node rn and at the beginning $rn = x_0$.
4. While $V \neq \emptyset$,
    a. If $rn.R - V' = \emptyset$, go to step 5. This implies that there is no link connecting to the nodes outside V'.

b. $V' = rn.R \cup V'$. The connected nodes from rn outside V' are now included in V'.

c. If T is reached ($T \in V'$), go to step 5.

d. Converting V' to be a representative node rn as for each $x_j \in V'$

   i. $R = R \cup x_j$ (Including the nodes connected by V').

e. $rn.R = R$.

f. $V = V - V'$ (Whittling down V for the remaining non-reachable nodes).

5. If $T \in V$, T is not reachable from x_0; otherwise, we say that T is reachable from x_0.

The representative node rn is presented as a collection of the current reachable nodes during the algorithm run and determines the links that connect to the nodes outside its collection (unreachable nodes at that moment). Efficiency is gained as the reachable nodes are memorised for the next iteration and will no longer be revisited. The line $V = V - V'$ is an essential part to reduce the set V by the set of already reached nodes V'. The iteration at line 6 and 11 run linearly to the workflow variables. As such, the time used in the while loop is reduced. This makes the reachability algorithm in several cases less than $O(n^2)$.

Figure 5.10 shows the example of how the reachability algorithm runs. An RCOBDD tree represents a workflow formula $(x_1 \text{ AND } x_2) \text{ XOR } (x_1 \text{ AND } x_2 \text{ AND } x_3 \text{ AND } x_6) \text{ XOR } (x_1 \text{ AND } x_4) \text{ XOR } (x_1 \text{ AND } x_4 \text{ AND } x_5 \text{ AND } x_6) \text{ XOR } x_2 \text{ XOR } (x_2 \text{ AND } x_3 \text{ AND } x_6)$ and a Composite formula is $\mathcal{P}E_t(q_1 \sqcup q_2)$. Suppose that the highlighted nodes x_2 and x_5 are two services marked with *satisfied* with the property q_1 and q_2, respectively. The final result from algorithm ForSomeSat1 depends on whether there is a path reachable from x_0 to T that traverses only *unsatisfied* nodes. This algorithm runs in the following steps. First, it eliminates the highlighted nodes as shown in Figure 5.10(a) and (b). Then, the algorithm reachable takes place that is graphically described in Figure 5.10(c)-(e). Since there exists a path $< x_0, x_{1,}, x_4, T >$ where all variable nodes (x_1 and x_4) are marked with *unsatisfied*, the algorithm returns *partially satisfied* as a final result.

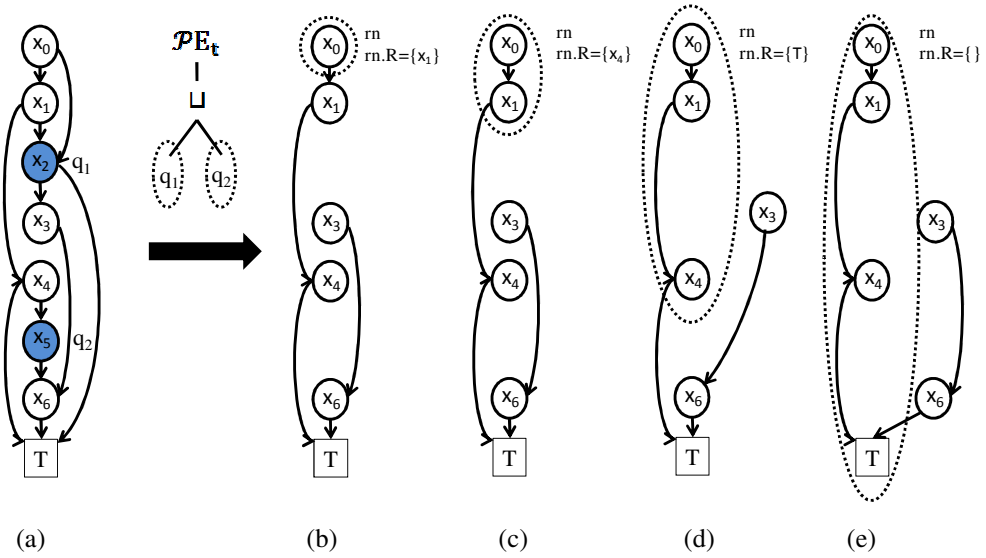


Figure 5.10: An Example Execution for Algorithm Reachable

5.3.4.2.2 2nd Class Algorithms

Whenever there are several cloves in which logical operators (& and ||) of Composite formula are presented, the reachability technique is no longer applicable. The reason is that both quantifier operators (E_t and A_t) can be presented at the same time. The way reachability query used in ForSomeSat1 and ForAllSat1 can no longer be employed. This forces the algorithms, to be presented below, to extract every execution path (each set of activated services) and then check for its satisfiability. The algorithms are much simpler since checking for an execution implies that services involved in that execution are connected by AND. From each result, the final one can be obtained. These algorithms are described below (see Figure 5.11):

```
1. Function ForSomeSat2(V, C)
2. Begin
3.     For x_i ∈ V
4.         If  x_i satisfies any clove in C
5.             return satisfied;
6.         End if
7.     End for
8.     return unsatisfied;
9. End
```

Figure 5.11: Pseudo Code for Algorithm ForSomeSat2

Algorithm ForSomeSat2

Assume that an RCOBDD tree has an ordering $[x_1, x_2, \ldots, x_n]$, the operations are described as follows (see Figure 5.11).

- For each clove tree $E_t(\phi_1 \sqcup \ldots \sqcup \phi_n)$, if $\phi_i(x_i)$ is true, the formula is marked with *satisfied*; otherwise it returns *unsatisfied*.

```
1. Function ForAllSat2(V, C)
2. Begin
3.     For x_i ∈ V
4.         If  x_i does not satisfy any clove in C
5.             return unsatisfied;
6.         End if
7.     End for
8.     return satisfied;
9. End
```

Figure 5.12: Pseudo Code for Algorithm ForAllSat2

Algorithm ForAllSat2

The ordering of an RCOBDD tree is the same as above. The operations of this check are described as follows (see Figure 5.12).

- For each clove tree $A_t(\phi_1 \sqcup \ldots \sqcup \phi_n)$ if some $\phi_i(x_i)$ is false, the formula is marked with *unsatisfied*; otherwise it returns *satisfied*.

```
1. Function LogicalCheck(Cformula, Path)
2. Begin
3.     If Cformula == 'E'
4.         return ForSomeSat2(Cformula, Path);
5.     Else
6.         return ForAllSat2(Cformula, Path);
7.     End if
8.     result1 = LogicalCheck(Cformula. Tree1, Path);
9.     result2 = LogicalCheck(Cformula. Tree2, Path)
10.    Switch
11.        Case '&'
12.            If result1== satisfied and result2 == satisfied
13.                return satisfied;
14.            Else return unsatisfied;
15.            End if
16.        Case '||'
17.            If result1== unsatisfied and result2 == unsatisfied
18.                return satisfied;
19.            Else return satisfied;
```

```
20.          End if
21.     End switch
22. End
```

Figure 5.13: Pseudo Code for Algorithm LogicalCheck

Algorithm LogicalCheck

From the result obtained from $ForSomeSat2$ and $ForAllSat2$, $LogicalCheck$ takes place to determine the result of all clove trees that are connected by & or || (see Figure 5.13).

- ϕ_1 & ϕ_2 : If ϕ_1 and ϕ_2 are satisfied, the algorithm returns *satisfied*, or *unsatisfied* otherwise.
- ϕ_1 || ϕ_2 : If ϕ_1 or ϕ_2 are satisfied, the algorithm returns *satisfied*, or *unsatisfied* otherwise.

```
1.  Function ExtractPathCheck(Cformula, V)
2.  Begin
3.      For each Path of a set V
4.          result[] = LogicalCheck(Cformula, Path);
5.      End for
6.      If all(result[]) == satisfied
7.          return satisfied;
8.      Else if some(result[]) == satisfied
9.          return  partially satisfied;
10.     Else
11.         return  unsatisfied;
12.     End if
13. End
```

Figure 5.14: Pseudo Code for Algorithm EtractPathCheck

Algorithm ExtractPathCheck

Algorithm ExtractPathCheck extracts all possible sets of services for each execution and then recursively sends each set to algorithm LogicalCheck. Based on each result, it determines the final result of a Composite formula as follows (see Figure 5.14).

- Extract all possible executions which are inputted to algorithm LogicalCheck
    a. If all executions are satisfied, the result is *satisfied*,
    b. if some executions are satisfied, the result is *partially satisfied*, and
    c. if no execution is satisfied, the result is *unsatisfied*.

The whole algorithms run as follows (see Figure 5.15). Firstly, an RCOBDD tree is extracted by algorithm ExtractPathCheck to generate all possible executions and each execution is inputted into algorithm ForSomeSat2 or ForAllSat2 according to E_t or A_t via algorithm LogicalCheck. Each returned result is used by algorithm LogicalCheck to check *satifiability* for & or ||. Finally, algorithm ExtractPathCheck takes place again to generate the final result.

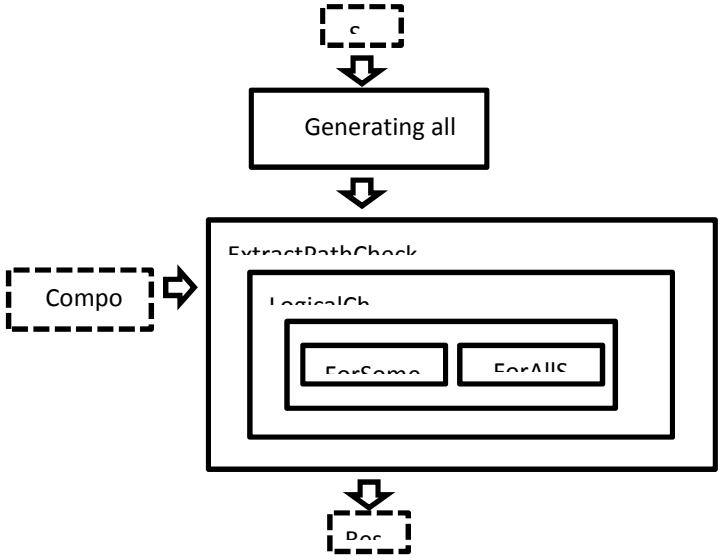


Figure 5.15: A High Level of How Algorithms for Composite Formula Run

5.3.4.3 Analysis of Algorithms for RCOBDDs

1st Class Algorithms

Algorithms ForSomeSat1 and ForAllSat1 contains two parts: (1) to check for each node whether it satisfies any clove, and (2) to find the answer from reachability query. In the latter part, the check is incrementally processed: starting with an initial node, and iteratively encompass the next connected node. The efficient compliance checking of both algorithms is $O(|C||V| + |V|^2)$. The term $|C||V|$ is from the check between cloves C and workflow variables V in the first part. The second term $|V|^2$ is the worst case time complexity for algorithm reachable when all links are saturated. The number of iterations from line 6 to 13 in Figure 5.9 is reduced as the elements in the set of variables V are removed by the set

of already reached nodes (V') in each iteration. This make the time complexity in average cases less than $O(|V|^2)$. As a result, the efficiency for algorithms ForSomeSat1 and ForAllSat1 is $O(|C||V| + |V|^2)$ using $O(|C| + |V|^2)$ storage space. In the average case, the time complexity can be computed as follows (see the calculation in Appendix C).

$$\text{average (the number of links)} = E(n \times \binom{n}{n} + (n-1) \times \binom{n}{n-1} + \cdots + 0 \times \binom{n}{0})$$

$$\text{average} = \frac{n}{2}$$

From this result, the average time complexity is $O\left(|C||V| + \frac{|V|}{2}\right) = O(|C||V|)$.

2nd Class Algorithms

Algorithms ForSomeSat2, ForAllSat2, LogicalCheck, and ExtractPathCheck take all variable ($|V|$) in each execution to check for the cloves ($|C|$) and clove trees ($|T|$), and the algorithm ExtractPathCheck generates all possible executions which is maximally $2^{|V|}$. Therefore, the time complexity is $O(|C||T||V|2^{|V|})$. Similar to EDNF approach, the average time complexity is $O(|C||T||V|^2)$.

Table 5.3. Comparing Efficiency of Three Representations and the Algorithms

Models	Checking time	Average	Space Overhead
CTT	$O(\|C\|\|T\|\|V\|2^{\|V\|})$	$O(\|C\|\|T\|\|V\|2^{\|V\|})$	$O(\|C\| + \|V\|2^{\|V\|} + \|T\|)$
EDNF	$O(\|C\|\|T\|\|V\|2^{\|V\|})$	$O(\|C\|\|T\|\|V\|^2)$	$O(\|C\| + 2^{\|V\|} + \|T\|)$
RCOBDD with Non-atomic Cloves (2nd class algorithms)	$O(\|C\|\|T\|\|V\|2^{\|V\|})$	$O(\|C\|\|T\|\|V\|^2)$	$O(\|C\| + \|V\|^2 + \|T\|)$
RCOBDD with Atomic Cloves (1st class algorithms)	$O(\|C\|\|V\| + \|V\|^2)$	$O(\|C\|\|V\|)$	$O(\|C\| + \|V\|^2)$

5.3.5 Comparison of CTT, EDNF, and RCOBDD Algorithms

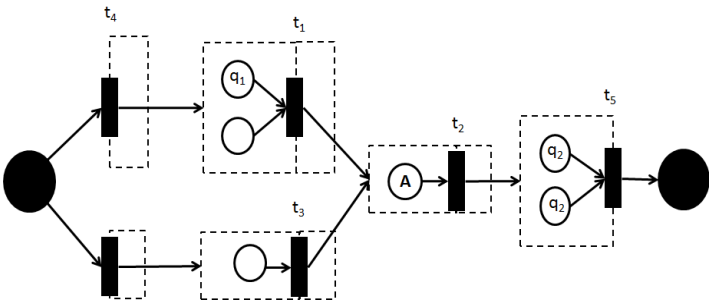
Table 5.3 presents the efficiency of Composite compliance checking algorithms of different approaches. From the comparison, the algorithms used for RCOBDDs are the most appealing. This is because the representation of RCOBDDs which makes the number of nodes linear to workflow variables.

5.4 Part II: Compliance Checking for Path Formulas

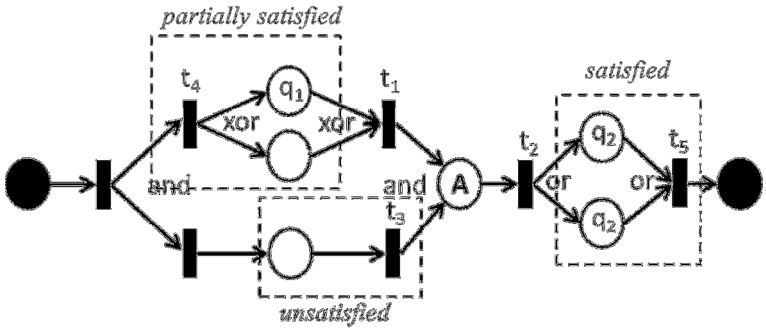
From the results of Composite compliance checking, the algorithms for Path formulas are the next step to check *satisfiability* regarding paths and sequence of tasks. This section describes in details the compliance checking algorithms for Path formulas with the introduction of constrained tasks which makes the algorithms slightly more complicated than existing CTL model checkers. Here, any SWN is simplified which are very similar to FSM and then Path algorithms is devised based on the original labelling algorithm (Chapter 3, Section 6, in [163] and Chapter 8, Section1, [164]) with the enhancement to deal with constrained tasks.

The reason to avoid using NuSMV [178] for these algorithms is that constrained tasks make the complexity overwhelming in building OBDD trees, which are used as a basic data structure in NuSMV. Precisely, FSM as an initial input for NuSMV is changed depending on each occurrence of constrained tasks which implies that there will be several OBDD trees to be created depending on the presence of constrained tasks. As a result, the algorithm must be executed several times in order to obtain the final answer. For instance, SWN shown in Figure 5.16 has to be converted into two FSMs with two starting states t_1 and t_3 in order to check a TS formula $\mathcal{B}(\forall_{\mathrm{t}_1} \Diamond \mathcal{F}\mathrm{E}_{\mathrm{t}_4}\mathrm{q}_1 \wedge \mathcal{P}\mathrm{A}_{\mathrm{t}_3}\mathrm{q}_3)$.

In the following, we present the algorithms with the extension from the original labelling algorithm to deal with constrained tasks for compliance checking of Path formulas.



(a) High-level Workflow Representation



(b) Detailed-level Workflow Representation

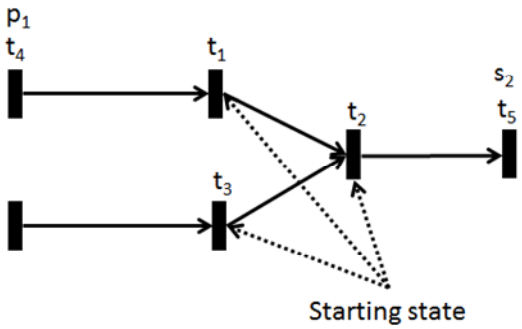
Figure 5.16: An Example of Simplifying SWN

5.4.1 A Quick Glance on Simplifying SWNs

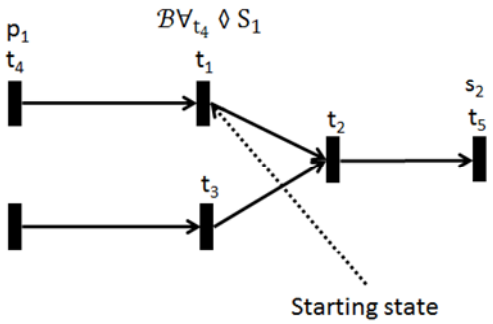
In this section, we describe informally how an SWN is simplified. There are two aspects of composite property from a task view point. (1) $\mathcal{F}$ indicates properties of services split (*service-split*) from a task, while $\mathcal{P}$ indicates properties of services responsible for a task (*service-join*). These two types are considered properties of a particular task. Figure 5.17 and 5.18 depict a simplified SWN. For better understanding, please see the example below.

Suppose that a service A is an imposing and also originating service with three formulas: (1) $\mathcal{B}\forall_{t_1} \Diamond \mathcal{F}A_{t_4}q_1$ (all paths in future from t_1 must reach t_4 and all recipient services at t_4 must satisfy property q_1), (2) $\mathcal{H}\forall\odot\mathcal{P}A_{t_5}q_2$ (for all tasks connected to t_2, all services composed for t_2 must satisfy property q_2), and (3) $\mathcal{B}\mathcal{P}A_{t_3}q_3$ (all services responsible for t_3 must satisfy property q_3). Focusing only on the Composite part of these formulas which are $\mathcal{F}A_{t_4}q_1$, $\mathcal{P}A_{t_5}q_2$, and $\mathcal{P}A_{t_3}q_3$ according to the first, second, and the third formulas respectively, the workflow example in Figure 5.16(b) yields three results from Composite compliance checking: *partially satisfied, satisfied*, and *unsatisfied* for the first, second and

third formulas (see the dash squares in Figure 5.16(b) indicating where the Composite formulas apply). Atomic propositions s_i, p_i and u_i are used for *satisfied*, *partially satisfied*, and *unsatisfied*. The enumeration S_i represents the parts of Composite formulas in the given TS formulas. In this example, we use S_1 for $\mathcal{F}A_{t_4}q_1$, S_2 for $\mathcal{P}A_{t_2}q_2$, and S_3 for $\mathcal{P}A_{t_3}q_3$, such that the three previous formulas are shorten as $\mathcal{B}\forall_{t_1} \Diamond S_1$, $\mathcal{H}\forall\odot S_2$, and $\mathcal{B}\mathcal{P}S_3$. Now, we can see that the formulas are very much similar to CTL.



(a)



(b)

Figure 5.17: The Example of the Labelled SWN

Based on the results from Composite compliance checking, the steps of simplifying SWN are described. Each task (from now on, we use tasks and states interchangeably) is labelled in the SWN with properties, i.e. s_i and p_i where i indicate the formulas 1, 2 and 3. u_i is not labelled meaning that there is no such property in that task. Figure 5.17(a) shows the result of this labelling at tasks t_4 and t_5 where t_3 is not labelled since its result is

unsatisfied. Finally, services in the SWN are no longer concern which will be removed and the links connected to the removed services will be redirected as shown in Figure 5.17(a).

In order to check for satisfiability for Path formulas, starting states must be located. In this example (see Figure 5.17(b)), any task connected to/from A is a starting state. t_1 is the starting state for the formula $\mathcal{B}\forall_{t_1} \Diamond \mathcal{F}A_{t_4}q_1, t_2$ for $\mathcal{H}\forall\odot\mathcal{P}A_{t_2}q_2$, and t_3 for $\mathcal{B}\mathcal{P}A_{t_3}q_3$. This is because A is an originating service in which each formula applies.

Here, the example of the labelling process is described. The first formula $\mathcal{B}\forall_{t_1} \Diamond S_1$ indicates that there must be at least one service in the *backward* ($\mathcal{B}$) direction from A with property S_1, ($S_1 = \mathcal{F}A_{t_4}q_1$). Since services split of the task t_1 partially satisfies $\mathcal{F}A_{t_4}q_1$, we label t_1 with $\mathcal{B}\forall_{t_1} \Diamond S_1$ (see Figure 5.17(b)). The presence of t_1 and t_4 in the formula implies that we concern with only tasks t_1 and t_4 connected from A. In this formula, t_1 is only one staring state. The compliance checking will return *partially satisfied* since t_1 is labelled by $\mathcal{B}\forall_{t_4} \Diamond S_1$ and t_4 is labelled by p_1. The process of simplifying SWN and its algorithms are referred to Appendix D.

5.4.2 Labelling Algorithm with Constrained Tasks

In any TS formula, constrained tasks are specified in a left-to-right sequence and will be applied according to the direction operators $\mathcal{B}$ or $\mathcal{H}$. For instance, a formula $\mathcal{B}(\forall_{t_1} \Diamond \exists_{t_1}[\forall_{t_2} \Diamond S_1 \Cup \exists_{t_5} \Diamond S_2])$ for the SWN presented in Figure 5.18 has the sequence $< t_1, t_2, t_5 >$. This sequence does not restrict t_i and t_j in $< t_i, t_j >$ to be adjacent. However, it restricts that there must be at least a path connecting from t_i to t_j. When the propositional type of Path operators is presented, the constrained task at the terms connected by a propositional operator must be identical; otherwise the formula is always invalid. For example, a formula $\mathcal{H}(\forall_{t_1} \Diamond \exists_{t_2}[(\forall_{t_2} \Diamond S_1 \wedge \exists_{t_2} \Diamond S_3) \Cup \exists_{t_5} S_2])$ indicates that two terms connected by $\wedge$ have the same constrained task t_2 ($\forall_{t_2} \Diamond$ and $\exists_{t_2} \Diamond$). If two of these constrained tasks are not the same, for instance, $\forall_{t_1} \Diamond S_1 \wedge \exists_{t_2} \Diamond S_2$, a formula is always false because t_1 and t_2 are considered different states, and there will not be a single state that satisfy both $\forall_{t_1} \Diamond S_1$ and $\exists_{t_2} \Diamond S_2$ (see Figure 5.18). Constrained tasks play an essential

factor in our labelling algorithm to determine the scope of labelling process. The labelling operation is done recursively to produce a final set of states that satisfy TS formulas.

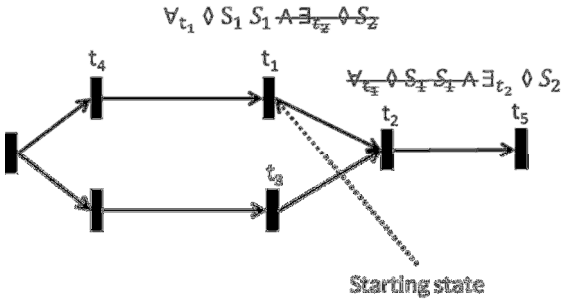


Figure 5.18: The Example of Labelling States of $\forall_{t_1} \Diamond S_1 \wedge \exists_{t_2} \Diamond S_2$

In this section, we present Path algorithms extended from the original labelling algorithms with the introduction of constrained tasks. Restricted by these tasks, we proceed to mark each point with the set of the smallest sub-formulas which are valid for that particular point. Then, the algorithm works recursively outwards towards the complete formula. This algorithm takes a TS formula and SWN as two inputs, where the output is a set of states that satisfy the formulas. The final result is obtained by simply checking if the starting states are members of the output states.

We develop an algorithm PathSat that takes two inputs: (1) a TS formula ϕ and (2) an SWN. Any TS formula is transformed such that no Composite parts are presented. Now, the operators to be considered consists of $\forall_t\odot$, $\exists_t\odot$, $\forall_t \Cup$, $\exists_t \Cup$, $\exists_t \Diamond$, $\forall_t \Diamond$, $\exists_t\Box$, and $\forall_t\Box$. The equivalences given in Chapter 4, Section 4.4.2, are used to translate the formula into the operators $\sim$, $\wedge$, $\vee$, $\exists_t\odot$, $\forall_t\odot$, $\exists_t \Cup$, and $\forall_t \Diamond$. Based on the labelling algorithm [163], we describe by using a case analysis below (t represents a task (state) in an SWN):

- p: label t with p if p is a property resulted from Composite compliance checking.
- $\phi_1 \wedge \phi_2$: label t with $\phi_1 \wedge \phi_2$ if t is already labelled with ϕ_1 and ϕ_2.
- $\sim\phi_1$: label t with $\sim\phi_1$ if it is not labelled with ϕ_1.
- $\exists_t\odot\phi_1$: (see Figure 5.19)
    - If t is not specified,
        i. label any state with $\exists\odot\phi_1$ if one of its successors is labelled with ϕ_1.

- If t is specified,
  i. Label the state t with $\exists_t \odot \phi_1$ if one of its successors is labelled with ϕ_1.

- $\forall_t \odot \phi_1$:
  - If t is not specified,
    i. label any state with $\forall \odot \phi_1$ if all of its successors is labelled with ϕ_1.
  - If t is specified,
    i. label the state t with $\forall_t \odot \phi_1$ if all of its successors is labelled with ϕ_1.

- $\exists_t [\phi_1 \Cup \phi_2]$: (see Figure 5.20)
  - If t is not specified,
    i. for any state is labelled with ϕ_2, we label it with $\exists [\phi_1 \Cup \phi_2]$.
    ii. Repeat until there is no change: label any state with $\exists [\phi_1 \Cup \phi_2]$ if it is labeled with ϕ_1 and one of it successor is already labeled with $\exists [\phi_1 \Cup \phi_2]$.
  - If t is specified, continue from above.
    i. If the state t is labelled with $\exists [\phi_1 \Cup \phi_2]$, we label it with $\exists_t [\phi_1 \Cup \phi_2]$.

- $\forall_t \Diamond \phi_1$: (see Figure 5.21)
  - If t is not specified,
    i. if any state is labelled with ϕ_1, we label it with $\forall \Diamond \phi_1$.
    ii. Repeat until there is no change: label any state with $\forall \Diamond \phi_1$ if all of its successors are is labelled with $\forall \Diamond \phi_1$.
  - If t is specified, continue from above.
    i. If the state t is labelled with $\forall \Diamond \phi_1$, we label it with $\forall_t \Diamond \phi_1$.

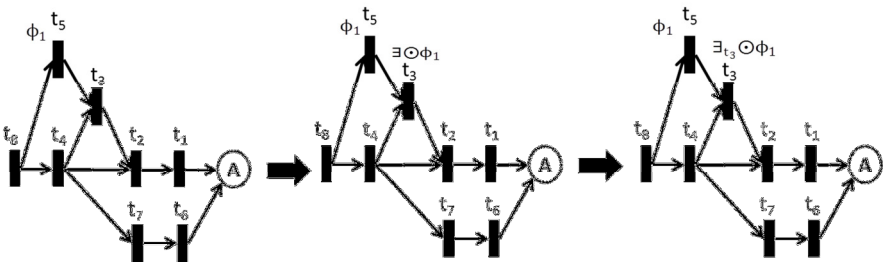


Figure 5.19: The Iteration Step for Labelling States with Formula $\exists_{t_3} \odot \phi_1$

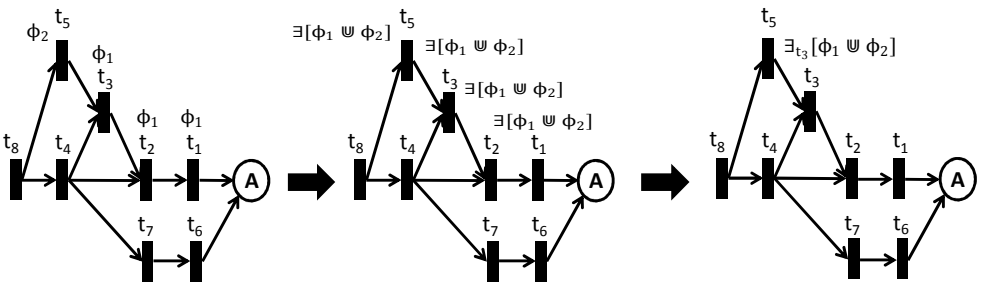


Figure 5.20: The Iteration Step for Labelling States with Formula $\exists_{t_3}[\phi_1 \Cup \phi_2]$

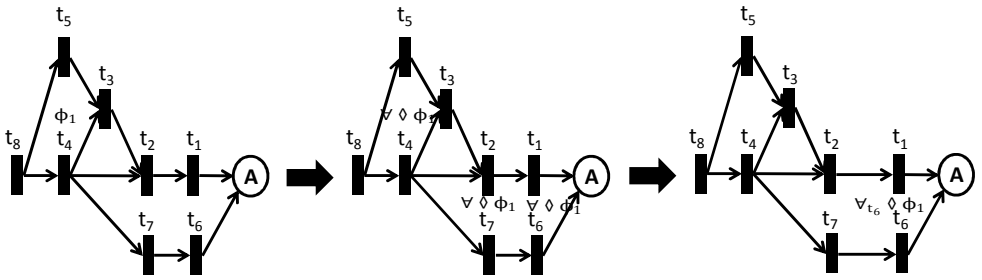


Figure 5.21: The Iteration Step for Labelling States with Formula $\forall_{t_6} \Diamond \phi_1$

Some functions are not explained as they can be written in the form of others. In the following algorithms (see pseudo codes in Figure 5.22-5.27), we use an adequate set of operators consists of $\exists_t \odot \phi_1$, $\forall_t \odot \phi_1$, $\exists_t[\phi_1 \Cup \phi_2]$, and $\forall_t \Diamond \phi_1$ and the rest of operators can be derived by using the following rules.

- $\forall_t[\phi_1 \Cup \phi_2] \equiv \sim(\exists_t[\sim\phi_2 \Cup (\sim\phi_1 \wedge \sim\phi_2)] \vee \exists_t \Box \sim\phi_2)$
- $\exists_t \Diamond \phi_1 \equiv \exists_t[\top \Cup \phi_1]$
- $\exists_t \Box \phi_1 \equiv \sim\forall_t \Diamond \sim\phi_1$
- $\forall_t \Box \phi_1 \equiv \sim\exists_t \Diamond \sim\phi_1$

The labelling algorithms are based on sub-functions SATEX for $\exists_t \odot \phi_1$, SATAX for $\forall_t \odot \phi_1$, SATEU for $\exists_t[\phi_1 \Cup \phi_2]$, and SATAF for $\forall_t \Diamond \phi_1$. They rely on the implementation of the sub-functions $\mathrm{pre}_\exists(X, d)$ and $\mathrm{pre}_\forall(X, d)$ which are defined below.

$$\mathrm{pre}_\exists(X, d) = \begin{cases} \{s \in S | \text{there exists that } s \to s' \text{where } s' \in X \text{ and } d = \mathcal{H}\} \\ \{s \in S | \text{there exists that } s' \to s \text{ where } s' \in X \text{ and } d = \mathcal{B}\} \end{cases}$$

$$\mathrm{pre}_\forall(X, d) = \begin{cases} \{s \in S | \text{for all } s \to s' \text{where } s' \in X \text{ and } d = \mathcal{H}\} \\ \{s \in S | \text{for all } s' \to s \text{ where } s' \in X \text{ and } d = \mathcal{B}\} \end{cases}$$

The function $\mathrm{pre}_\exists$ (see Figure 5.26) used in SATEX, SATEU, and SATAF takes an initial set Y. It returns the set of states that can make a transition to Y in the case of $d = \mathcal{H}$ and return the set of states that is immediately reachable from Y in the case of $d = \mathcal{B}$. The function $\mathrm{pre}_\forall$ (see Figure 5.27) used in SATAX and SATAF takes an initial set Y. It returns the set of states that only makes a transition to Y in the case of $d = \mathcal{H}$ and returns the set of states to which only Y can make a transition in the case of $d = \mathcal{B}$. $\mathrm{pre}_\forall$ can be expressed in terms of complementation of $\mathrm{pre}_\exists$, as follows:

$$\mathrm{pre}_\forall(X, d) = S - \{\mathrm{pre}_\exists(S - X, d)\}$$

```
1.  Function PathSat(ϕ, d)
2.  Begin
3.      Case
4.          ϕ = ⊤: return S;
5.          ϕ = ⊥: return ∅;
6.          ϕ = atomic: return {s ∈ S|ϕ is the property from Composite compliance checking};
7.          ϕ = ~ϕ1: return S − PathSat(ϕ1, d);
8.          ϕ = ϕ1 ∧ ϕ2: return PathSat(ϕ1, d) ∩ PathSat(ϕ2, d);
9.          ϕ = ϕ1 ∨ ϕ2: return PathSat(ϕ1, d) ∪ PathSat(ϕ2, d);
10.         ϕ = ϕ1 → ϕ2: return PathSat(~ϕ1 ∨ ϕ2, d);
11.         ϕ = ∀t⊙ϕ1: return SATAX(ϕ1, d, t);
12.         ϕ = ∃t⊙ϕ1: return SATEX(ϕ1, d, t);
13.         ϕ = ∀t[ϕ1 ⩊ ϕ2]: return PathSat(~(∃t[~ϕ2 ⩊ (~ϕ1 ∧ ~ϕ2)] ∨ ∃t□~ϕ2), d);
14.         ϕ = ∃t[ϕ1 ⩊ ϕ2]: return SATEU(ϕ1, ϕ2, d, t);
15.         ϕ = ∃t ◊ ϕ1: return PathSat(∃t[⊤ ⩊ ϕ1], d);
16.         ϕ = ∃t□ϕ1: return PathSat(~∀t ◊ ~ϕ1, d);
17.         ϕ = ∀t ◊ ϕ1: return SATAF(ϕ1, d, t);
18.         ϕ = ∀t□ϕ1: return PathSat(~∃t ◊ ~ϕ1, d);
19.     End case
20. End
```

Figure 5.22: Pseudo Code for TSSat *that Calls* $Sat_{\exists\odot}$, $Sat_{\forall\Diamond}$, *and* $Sat_{\exists\Cup}$

```
1.  Function SATEX(ϕ, d, t);
2.  Begin
3.      X, Y = { };    // sets of state
4.      X = CompositeSat(ϕ, d);
5.      Y = Pre∃(X, d);
6.      Case
7.          t = null: return Y;   // if in the case that t is not specified in ∃t⊙
8.          t ∈ Y: return t;
```

```
9.          default: return ∅;
10.     End case
11. End
```

Figure 5.23: Pseudo Code for SATEX

```
1.  Function SATEU(ϕ1, ϕ2, d, t)
2.  Begin
3.      W, X, Y = {};     // sets of states
4.      W = CompositeSat(ϕ1, d);
5.      X = S;
6.      Y = CompositeSat(ϕ2, d);
7.      While  X~ = Y
8.          X = Y;
9.          Y = Y ∪ (W ∩ Pre∃(Y, d));
10.     End while
11.     Case
12.         t == null: return Y;
13.         t ∈ Y: return t;
14.         default: return ∅;
15.     End case
16. End
```

Figure 5.24: Pseudo Code for SATEU

```
1.  Function SATAF(ϕ, d, t)
2.  Begin
3.      X, Y, Z = {};     // sets of states
4.      X = S;
5.      Y =  CompositeSat(ϕ, d);
6.      Repeat until  X == Y
8.          X = Y;
9.          Y = Y ∪ Pre∀(Y, d);
10.     End repeat
11.     Case
12.         t == null: return Y;
13.         t ∈ Y: return t;
14.         default: return ∅;
15.     End case
16. End
```

Figure 5.25: Pseudo Code for SATAF

```
1.  Function pre∃(X, d)
2.  Begin
3.      Y = { };     // sets of states
4.      If d == ℋ
5.          For each (s, s′) ∈ F. R
6.              If s′ ∈ X
7.                  Y = Y ∪ s;
8.              End if
9.          End for
10.     End if
11.     If d == ℬ
12.         For each   (s, s′) ∈ F. R
```

```
13.            If s ∈ X
14.               Y = Y ∪ s′;
15.            End if
16.         End for
17.      End if
18.      return Y;
19. End
```

Figure 5.26: Pseudo Code for $\mathrm{pre}_{\exists}$

```
1. Function pre∀(X, d)
2. Begin
3.      return F.S − pre∃(F.S − X, d);
4. End
```

Figure 5.27: Pseudo Code for $\mathrm{pre}_{\forall}$

5.4.3 Analysis of Labelling Algorithm

The complexity depends on the number of operators presented (m) that will be labeled on the depth first search fashion. The sub-functions SATEX, SATAX, SATEU and SATAF contain a loop that runs on searching possibly every tasks (states) T and also rely on the implementation of $\mathrm{pre}_{\exists}$. The function $\mathrm{pre}_{\exists}$ determines all possible transitions in the set R. As a result, the complexity evaluation of this algorithm is $O(|m||T||R|)$. In the worst case where the transitions are saturated, this function will take $O(|m||T|^3)$ using $O(|m|\log|m||T|)$ space.

5.5 Part 3: Compliance Checking for Direction Formulas

The result obtained from the compliance checking for Path formulas determines which tasks (states) are satisfied. In this way, truth value can be evaluated as follows. If an originating service is part of the satisfied states, the formula is marked with true; otherwise it will be marked with false. Since some Direction operators are similarly defined as in propositional logic, we apply marking algorithm based on SAT solver [176] to get the final answer. The algorithm works from each sub-formula outward toward the complete formula that is represented by a parse tree. In Figure 5.28, the pseudo code for compliance checking for Direction formulas (DirectionSat) is described.

```
1.  Function DirectionSat(ϕ, d);
2.  Begin
3.      If ϕ is Path formula
4.          states = PathSat(ϕ, d);
5.      End if
6.      If originating service is part of states
7.          return true;
8.      Else
9.          return false;
10.     End if
11.     ϕ1 = DirectionSat(ϕ.LeftNode, d)
12.     ϕ2 = DirectionSat(ϕ.RightNode, d)
13.     Case ϕ.Element
14.         ~: If ϕ1 == true
15.                return false;
16.            Else
17.                return true;
18.            End if
19.         ∧: If ϕ1 == true and ϕ2 == true
20.                return true;
21.            Else
22.                return false;
23.            End if
24.         ∨: If ϕ1 == true or ϕ2 == true
25.                return true;
26.            Else
27.                return false;
28.            End if
29.         →: If ϕ1 == false
30.                return true;
31.            Else
32.                return ϕ2;
33.            End if
34.         ↔: If (ϕ1 == false and ϕ1 == false) or (ϕ1 == true and ϕ1 == true)
35.                return true;
36.            Else
37.                return false;
38.            End if
39.     End case
40. End
```

Figure 5.28: Pseudo Code for The DirectionSat *Algorithm*

5.6 Handling the Results between Satisfied and Partially Satisfied

It turns out that the algorithms for compliance checking generate a binary result (satisfy or not), indicating whether a given SWN is satisfied by a TS formula. In order to determine the yes answer which can be *satisfied* or *partially satisfied*, the algorithm has to be executed twice. In the first run, the *partially satisfied* result is labelled as a valid property. If this

checking produces the no as a result, *unsatisfied* is immediately obtained. The second run is necessary whenever the yes answer is produced from the first run. Now, *partially satisfied* is no longer a valid property which will be taken out from the first labelling. If the algorithm in the second run still produces the yes answer, it can be concluded that the workflow is *satisfied*; or *partially satisfied* otherwise (see Figure 5.29).

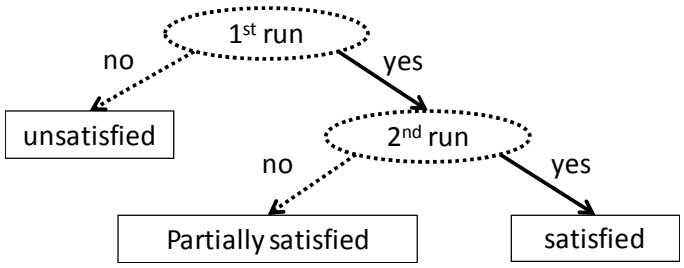


Figure 5.29: Running Sequences of Unsatisfied, Partially Satisfied, and Satisfied

5.7 Chapter Discussion

This chapter presents the algorithms for compliance checking between SWNs and TS formulas. In the first part, the existing works on CTT and EDNF for Composite compliance checking are reviewed and then the efficient algorithms are devised that run on the special type of data structure, RCOBDDs. In the 1st class algorithm, reachability query is a core component that provides better running time compared to CTT and EDNF. The 2nd class is developed to deal with multiple clove trees, or in other words, when logical operators & and || are presented. In this case, the running time is similar to EDNF. The efficiency is evaluated with comparison with these existing approaches. Based on the result from Composite compliance checking, it will be inputted to the compliance checking algorithm for Path formulas presented in the second part. This algorithm is extended from the existing CTL labelling algorithm to deal with constrained tasks. Finally, in the third part, the algorithm for Direction formula is presented on the basis of SAT solver. Since it produces a binary answer, it must be run twice to obtain the final result. The algorithms have several advantages for the verification of SWNs. For instance, they can be made automatic by the implementation presented in Chapter 6 and 7. Below, we discuss some interesting issues and future directions.

SWN Formulas

Instead of evaluating truth values of a formula as a primary focus in several classes of logics, SWN formulas aim at representing workflow structure and providing all possible executions based on service activation in each task. As a result, some rules in classical logic are not applicable since some essential structural information can be lost. For example, it will be impossible to reduce the formula (A AND B) OR A to A because the information that B can also be part of a workflow execution is missing. Workflow formulas must capture this information in order to correctly check for compliance of requirements.

Algorithms for Composite Formulas

The goal of Composite compliance checking is to check the properties of services that are connected from/to (*split-* or *join-type*) a task. These services for each task are presented in the RCOBDD format, a special type of DAG with topological sort. The benefit is gained from the 1st class algorithms (ForSomeSat1 and ForAllSat2) that run on top of the reachability query. It is implemented based on depth first search that provides $O(|V|^2)$ running time. However, other reachability query approaches presented in Table 5.2 can be used with the possibility to increase performance in some cases such as when a workflow is not changed overtime. This requires further investigation in terms of characteristics of services composition in different domains that would then decide the most appropriate approach. At this stage, since the number of services in an SWN is relatively small compared to software systems, the practicality of the algorithm reachable with $O(|V|^2)$ is promising.

Algorithm for Path and Direction Formulas

As mentioned, the algorithms for Path formula is extended from the existing labelling algorithm to deal with constrained tasks. In any TS formula, tasks come in order which depends upon user discretion to ensure that the order is valid. For any two tasks t_i and t_j in the ordering $< t_i, \dots, t_j >$, there always be at least one connecting path from t_i to t_j. These tasks are coordinated and controlled by workflow connectives which describe how an SWN is executed.

Our labelling algorithm relies on algorithm SWN2FSM in Appendix D that transforms a given SWN into the FSM format. Clearly, FSM does not retain the information of workflow connectives. For example, it cannot address the difference between AND and OR of the outgoing paths at each state which in the case of AND more than one path is required to be enabled. This incapability affects the meaning of $\exists_t$ which only describes that it has the possibility to reach the target state from a current state, but it cannot guarantee that the target is always activated in each workflow instance. $\forall_t$, on the other hand, describes that it will get to the target states from a current one, with guarantee that the states are always executed in any execution. To capture the different effect of workflow connectives, more operators have to be employed. For example, the TS grammar can be extended with the operators *possibly* and *necessarily* borrowed from modal logic to address this effect.

Conflict Discovery

To locate conflicts in SWN from TS formulas, the algorithms can be augmented to produce the actual services and paths that falsify TS formulas as well as identify the cloves that cause such conflicts. That way, it can reduce time of decision makings when this information is ready at the same time when the algorithms finish. This information facilitates a workflow owner to be able to identify and also replace conflict services. The participating services on the other hand are provided with the identification of the conflict TS formulas.

Composite and Workflow Trust Calculation

Even though traces and composite notions are addressed in TS formulas, their purpose is to specify properties of parts of services and tasks of an SWN which may have a significant effect on trust-based participation decision. In future research, based on the result of compliance checking, trust values of a trace and composite service based on the value of a single service trust value can be derived. As a result, the overall numeric trust value of a workflow can be evaluated.

SWN Subsets

One interesting characteristic in our approach is that it is not statically true or false in a model. Since a TS formula can be true in some positions, or structures, and can be false in some parts of a workflow, some workflow instantiations can still be valid if the false parts are not involved. The algorithms can be improved to extract the SWN components for these valid executions.

Chapter 6

A Scenario

6.1 Chapter Introduction

This chapter describes the full scenario of the travel agent workflow from Chapter 3 which will be used to evaluate the performance of our framework in Chapter 7. It is important to note that this workflow solely is not complete for the entire functionalities of travel agent business. It focuses on the core business that facilitates how a customer can plan his/her trip, where other related workflows such as cancellation, refunding, managing account, etc. which may interoperate in this workflow are not presented. Many types of TS formulas are provided in this scenario to demonstrate the applicability of capturing possible real world requirements.

6.2 Scenario Overview

The travel agent workflow is part of the travelling application that facilitates customers in searching, reserving and booking for flights, hotels, and transportations including car, train, and cruise. Each task is dynamically composed of services where service acquisition is determined from the compliance of requirements. All services are registered in the central database that serves as a pool for service discovery and acquisition.

Consider the workflow example in Figure 3.5 in Chapter 3, which depicts the partial view of the travel agent business. This paragraph describes deliberately each step of workflow creation from the beginning towards the complete one (to reduce complexity, we assume that the workflow and services are operated under the same ontology enabling the seamless interoperation from difference of service interfaces). Firstly, a workflow owner initialises a preliminary workflow by creating and coordinating workflow tasks shown in Figure 6.1. In this figure, the left dash rectangular represents *service-join* type (service composition) waiting to be filled later by the service acquisition phase, while the right one represents *service-split* type (service separation) which will be derived after the process of service composition is done. Below, the set of workflow tasks are described.

- **Searching* includes five tasks, i.e., *FlightSearching*, *HotelSearching*, *CruiseSearching*, *CarSearching*, and *TrainSearching*.
- *Location* provides a selected flight, hotel, car rental, and train with the location information of airport, hotel, car pick-up address, and train station.
- *WeatherForecasts* provides weather information according to the specified travelling period and location.
- *HotelRating* provides rating information according to selected hotels.
- **Reservation* will be activated after one, or multiple of, flight, hotel, cruise, car, or train, is selected from **Searching*. It consists of five tasks, *FlightReservation*, *HotelReservation*, *CruiseReservation*, *CarReservation*, and *TrainReservation.*
- **Booking* is activated once the reservation is made. It consists of five tasks according to **Reservation* which are *FlightBooking*, *HotelBooking*, *CruiseBooking*, *CarBooking*, and *TrainBooking.*
- *Restaurants* offer information, promotion, and reservation based on the destinations, the time of travelling, and the current collections of restaurant services.
- *Events* offer customers for special events at destinations.
- *TouristInfo* provides the relevant tourist information of selected destinations according to the time of travelling. It will be activated only after a payment is done.
- *Statistics* is a task with analysis and report capabilities based on incurred transactions.
- *Insurance* and *GuideandLocalTour* are two additional purchasing options for insurance, and guide and local tour.

- *Payment* is responsible for financial transactions which will be activated after one, or multiple, *Booking* is done with the purchasing options of *Events*, *Restaurants*, *GuideandLocalTour*, and *Insurance*.
- *TravellingDocs* produces electronic documents, such as vouchers, e-tickets, receipts, and acknowledgement messages.
- *PromotionPack* provides a more convenient and economy way by offering several items in bundle, such as flight tickets and hotel nights.

Mainly, the travel agent workflow provides two options to its customers. To serve accurate demands, the first option allows the customers better flexibility to select flight tickets, hotel nights, cars, cruises, or trains. Alternatively, the second option provided by *PromotionPack* is an economy choice, offering promotion packages in bundle which may include, for example, flight tickets, hotel nights, and restaurant banquets.

In the first option, whenever a customer request is received, the workflow will interact with service providers for providing searching results for one, or more, of flight, car, cruise, train tickets and hotel nights together with schedules and price quotations. **Searching* is the first task to be invoked, followed by **Reservation* executing the items selected from the results of **Searching*. If possible, the workflow provides optional *Location*s information of the selected items before **Reservation* starts. Weather information is provided by *WeatherForecast* according to the traveling period and location. The customer can also find reviews and rating information of the selected hotels provided by *HotelRating*. After **Reservation* is confirmed, the workflow proceeds to **Booking*. There are a number of additional offers including *Insurance*, *GuideandLocalTour*, *Restaurant* and other current available *Events* (for the ease of reading, these optional tasks are grouped as shown the round box in Figure 6.1). Upon the selected items, the customer is required to make a payment in two methods either credit card or Paypal. If the payment is approved, *TravellingDocs* will generate and send electronic documents such as hotel vouchers, e-ticket, and acknowledgement messages back to the customer. Optionally, *TouristInfo* may be activated to provide tourist information of the destinations. The transaction history may be stored for analysis at *Statistics*. Alternatively, the second option provides a more convenient and economy way by offering *PromotionPack*. Once a package is chosen, the workflow will consecutively proceed to *Payment* and *TravellingDocs*.

In reality, the workflow may require to interoperate with other workflows. For instance, to deal with cancellation that may involves refunding, fee, and surcharge, another cancellation workflow is required. However, as we focus on the compliance checking of the requirements within the single workflow domain, it does not cover the higher relationship among interoperating workflows (please refer to the architecture HEN+ [118] for inter-domain trust relationship to deal with this issue).

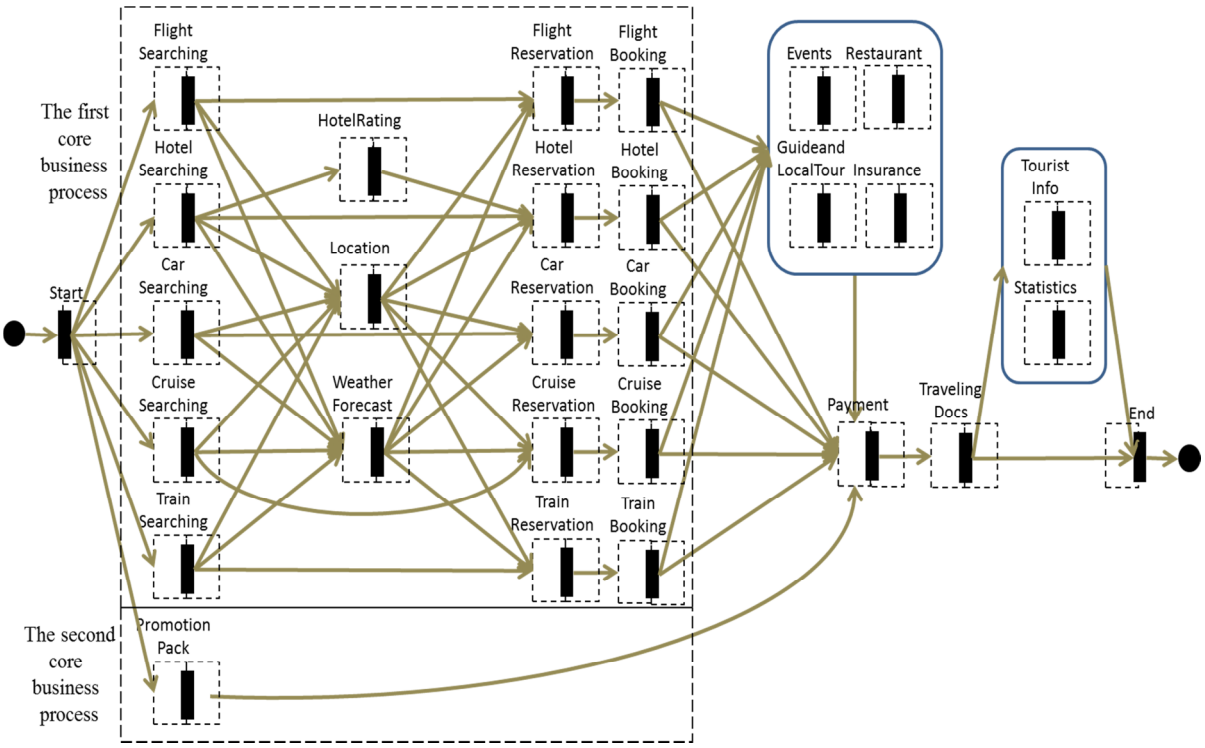


Figure 6.1: Task Coordination of the Travel Agent Workflow

From the detail above, there are two core processes in the workflow. The first process involves the tasks, or group of tasks, i.e. **Searching*, **Reservation, **Booking, Payment* and *TravellingDocs* that are activated in sequence. The second process includes the tasks in the sequence of *PromotionPack*, *Payment,* and *TravlellingDocs*. These tasks are considered *necessary* in the workflow operations. The other remaining tasks are considered *unnecessary*, offering auxiliary services along with the cores to facilitate travelling plans.

6.3 Task Necessity, Dependencies, and Specification

In this section, task necessity and dependencies are two main factors that regulate how tasks are coordinated and inter-connected. As mentioned, the first factor classifies tasks into *necessary* and *unnecessary* (see Table E.1 in Appendix E for the full list of task necessity). The second factor regulates how tasks are interconnected, either in sequence or parallel, controlled by the set of workflow connectives AND, OR, and XOR. For example, to start an execution, one option of the tasks in **Searching* or *PromotionPack* is invoked. This can be addressed by using XOR connective shown in the formula below. The mathematical expression is resided in *Start* to specify the invocation of the next tasks (please refer to **Def. 3.1** for the notations used here).

$$\text{Start}.\text{f} = (\psi, \text{split}), \text{ where}$$
$$\psi = (\langle \textit{FlightSearching} \rangle \text{ OR } \langle \textit{HotelSearching} \rangle \text{ OR } \langle \textit{CarSearching} \rangle \text{ OR } \langle \textit{CruiseSearching} \rangle \text{ OR } \langle \textit{TrainSearching} \rangle) \text{ XOR } \langle \textit{PromotionPack} \rangle$$

Since services for each task are not yet acquired, the temporary notation <*t*> is used to denote a set of services associated with a task. For instance, <*FlightSearching*>, as part of the formula ψ in $Start.\text{f} = (\psi, \text{split})$ above, is a set of services composed for *FlightSearching* which are reachable from *Start*. In other words, <*FlightSearching*> in the formula ψ contains a set of intermediate services between *Start* and *FlightSearching*.

WeatherForecast and *Location* are considered *unnecessary*. The incoming links are controlled by using OR connective as to make these tasks optional. For example, they are always possible to be avoided if *FlightReservation* is activated right after *FlightSearching* is completed. The task coordination specification formulas between *FlightSearching* and *FlightReservation* can be expressed by

$$\text{FlightSearching}.\text{f} = (\psi, \text{split}), \text{ where}$$
$$\psi = \langle \textit{FlightReservation} \rangle \text{ XOR } (\langle \textit{Location} \rangle \text{ OR } \langle \textit{WeatherForecast} \rangle)$$

$$\text{FlightReservation}.\text{f} = (\psi, \text{join}), \text{ where}$$

$$\psi = \langle FlightSearching \rangle \text{ XOR } (\langle Location \rangle \text{ OR } \langle WeatherForecast \rangle)$$

We note that the same <*t*> in different formulas at different tasks may not be identical. For example, <*FlightReservation*> appeared in the formulas (ψ, split) at FlightSearching.f, and Location.f may contain different sets of services. This is because there can be a service at *FlightReservation* that required to be connected only from *FlightSearching,* but not from *Location* (please also refer to Table E.1 in Appendix E for the full list of task specification).

It is important to note that task coordination is fully based on the workflow owner discretion which is done during the beginning of the workflow formation stage. At this point, the workflow is half-way complete; only services acquired for each task are missing. The next section will describe services acquisition according to the predefined tasks.

6.4 Service Composition, Separation, and Specification

The remaining work to complete is the acquisition of services. In open environments, services corresponding to each task can be both statically and dynamically selected and then composed together controlled by workflow connectives. In many cases, services are utilised in a composite manner. To execute a task that goes beyond one's capabilities, or for some other reasons, incorporating with multiple services is required.

In the travel agent workflow, assume that it incorporates with external services including airline, hotel, car rental, train, cruise, location, weather, insurance, guide and local tour, tourist information, restaurants, events, payment, and statistics. The workflow owner provides its own services for *TravellingDocs* (see the full list of services for each task in Table E.2 in Appendix E). Each of these services possesses different properties and functionalities resulting in the competent of executing a single, or multiple, task(s). For example, a VISA service is able to execute only *Payment*, while some airline services are able to execute *FlightSearching*, *FlightReservation*, and *FlightBooking*. Some of them may also provide *Insurance* along with flight tickets.

The following subsections describe service composition (*service-join*) which is a combination of services composed for a task, service separation (*service-split*) which involves services that are connected from a task, and finally, the specification of requirements from these services and the workflow owner in the form of TS formulas.

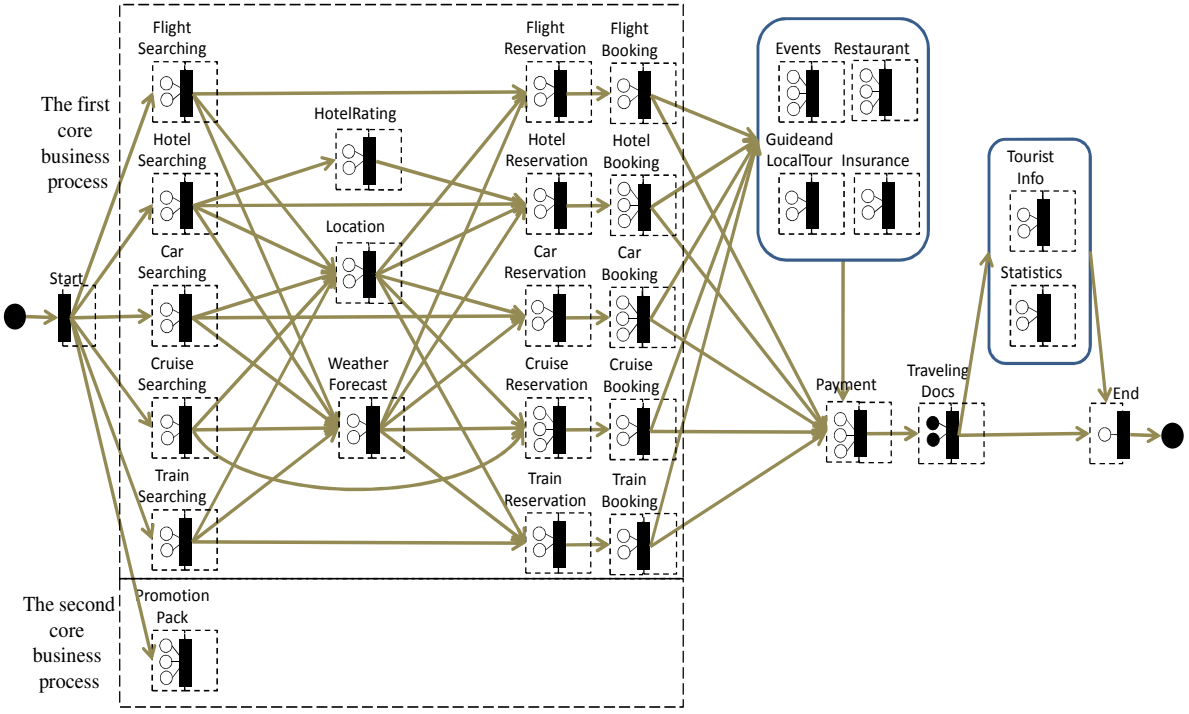


Figure 6.2: Task Coordination with Acquired Services

6.4.1 Service Composition

Suppose that we are at the point where all tasks are plugged with services (see Figure 6.2). *FlightSearching* gathers the integral search results from services *Expedia* (S_{XP}), *Priceline* (S_{PL}), *Ebookers* (S_{EB}), *EasyJet* (S_{EZ}), and *RyanAir* (S_{RA}). The logic in this composition is that S_{XP} and S_{PL} are two main services; one of them must be activated in every execution, while others are considered optional. This can be expressed by

$$FlightSearching.\mathrm{f} = (\psi, \mathrm{join}), \text{ where}$$
$$\psi = ((S_{XP}\ \mathrm{OR}\ S_{PL})\ \mathrm{AND}\ (S_{EB}\ \mathrm{OR}\ S_{EZ}\ \mathrm{OR}\ S_{RA}))\ \mathrm{OR}\ S_{XP}\ \mathrm{OR}\ S_{PL}$$

The graphical illustration in the form of RCOBDDs of this formula is shown in Figure 6.3 in the canonical order of S_{XP}, S_{PL}, S_{EB}, S_{EZ}, and S_{RA}. The rest of other services and their logical relations are also encoded by RCOBDDs (please refer the full list of formulas in Table E.3, the full service glossary in Table E.4, and graphical illustration of *service-join* and *service-split* in Section E.2 and E.3 in Appendix E, respectively).

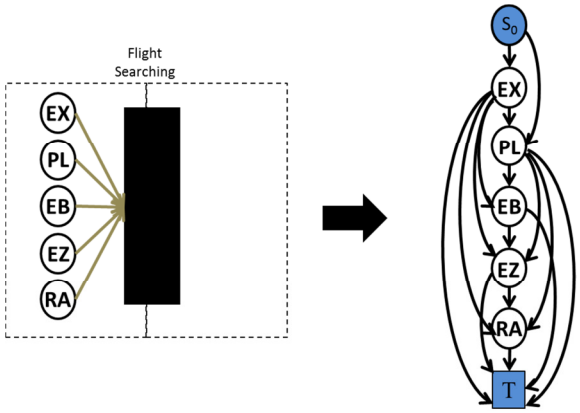


Figure 6.3: RCOBDD Form of Services for FlightSearching

The remaining service compositions are described as follows. *HotelSearching* incorporates services from *Expedia* (S_{XP}), *Priceline* (S_{PL}), *Ebookers* (S_{EB}), *Agoda* (S_{AG}), and *Booking* (S_{BO}) where at least one of these services must be invoked. *Agoda* (S_{AG}), and *Booking* (S_{BO}) also provide *HotelRating* information. *CarSearching* is executed by *Expedia* (S_{XP}), *Priceline* (S_{PL}), and *Avis* (S_{AV}) where at least one of S_{XP} or S_{PL} must be activated. *CruiseSearching* has only one service *WorldCruise* (S_{WC}). *TrainSearching* uses services from *WorldTrain* (S_{WT}), and *EuroStar* (S_{ES}), where S_{WT} must always be activated. The other **Reservation* and **Booking* use the same set of services as specified in **Searching*. *Location* and *WeatherForecast* employ services from either *GoogleMap* (S_{GM}), or *BingMap* (S_{BM}), and *BBCWeather* (S_{BBC}) or *Weather* (S_{WT}), respectively. *PromotionPack* is executed by at least one of *TravelPro* (S_{TP}), *Expedia* (S_{XP}), or *Priceline* (S_{PL}) services. *Events, Restaurant,* and *GuideandLocalTour*, incorporate with *WorldEvents* (S_{WE}), *WorldRestaurant* (S_{WR}), and *WorldLocalTour* (S_{WG}) and possibly with some other local services (S_{E1} for *Events*, S_{R1} for *Restaurant*, S_{G1} for *GuideandLocalTour*, …). *EazyJet* (S_{EZ}) provides insurance to who travels by airplanes, Avis provides insurance to who rents

a car from Avis, and *WorldInsurance* (S_{WI}) provides all kinds of traveling insurance. Alternatively, if the *PromotionPack* is chosen, only one of services *TravelPro* (S_{TP}), *Expedia* (S_{XP}), and *Priceline* (S_{PL}) will be invoked. S_{TP} is the service operated by the workflow owner.

There are three *Payment* services including *Visa* (S_{VS}), *Master* (S_{MS}), and *Paypal* (S_{PP}). *TravelingDocs* is executed by the workflow owner with two services S_{TD1} and S_{TD2}. They perform the same functionality and are redundantly composed to ensure availability. *TourisInfo* and *Statistics* are optional which are executed by *TourisInfo* (S_{TI}), and *DataMagagement* (S_{DM}) and *BusinessIntel* (S_{BI}), respectively. Finally, special services S_{D1} and S_{D2} at *Start* and *End* perform no operation; their purpose is to enable the correctness of the workflow semantics.

6.4.2 Service Separation

On the other hand, service separation, or *service-split*, (the right dash rectangular of each task in Figure 6.2) describes what and how services are connected from a task. Each set of service separation depends on each task, indicating which services connected from that task are potentially part of an execution. Service separation and composition formulas are constructed based on the split and join formulas of task specification (see Table E.3 in Appendix E). The task specification formulas in Section 6.2 denote service composition, specified by < > brackets. For example, <*FlightSearching*> appeared in *Start.split* is the set of services composed for *FlightSearching* which can be activated from *Start*. The *Start.split* formula is also represented by RCOBDDs, derived from the formulas of all intermediate services connected to the tasks from *Start* (see Figure 6.4 for example). The benefit of RCOBDDs is the reduction of the number of services since service duplications are eliminated. For instance, the presences of several S_{XP} and S_{PL} in almost all tasks connected from *Start* are reduced into one.

In this scenario, *Start* is the first task which, based on the interaction with external choices such as customer decision in this case, will determine which path to be activated. All services composed for the subsequent tasks are dynamically invoked. All **Searching* tasks have all paths directed through **Reservation* and then **Booking*. Services for

Location and *WeatherForecast* are all connected by all **Searching*, while *HotelRating* services can be reachable only from *HotelSearching*. There are two choices after **Booking*, either to select additional items offered by *Insurance, Events, GuidenadLocalTour*, and *Restaurant,* or directly advance to *Payment*. At this point, the process of service acquisition is complete and the workflow is ready to function.

Here, the example is provided to explain how *service-split* formulas are derived from *service-join*. Considering the formula ψ in $Start.\mathrm{f} = (\psi, \mathrm{split})$ from Section 6.2,

$$\mathrm{Start.f} = (\psi, \mathrm{split}), \text{ where}$$
$$\psi = (<\textit{FlightSearching}> \text{ OR } <\textit{HotelSearching}> \text{ OR } <\textit{CarSearching}> \text{ OR } <\textit{Cruise Searching}> \text{ OR } <\textit{TrainSearching}>) \text{ XOR } <\textit{PromotionPack}>$$

Service-join for each task is presented by each bracket <*FlightSearching*>, <*HotelSearching*>, <*CarSearching*>, <*CruiseSearching*>, <*TrainSearching*>, and <*PromotionPack*> that can be derived as follows.

$$<\textit{FlightSearching}> = \textit{FlightSearching}.\mathrm{f} = (\psi, \mathrm{join}), \text{ where}$$
$$\psi = ((\mathrm{S_{XP}} \text{ OR } \mathrm{S_{PL}}) \text{ AND } (\mathrm{S_{EB}} \text{ OR } \mathrm{S_{EZ}} \text{ OR } \mathrm{S_{RA}})) \text{ OR } \mathrm{S_{XP}} \text{ OR } \mathrm{S_{PL}}$$

$$<\textit{HotelSearching}> = \textit{HotelSearching}.\mathrm{f} = (\psi, \mathrm{join}), \text{ where}$$
$$\psi = \mathrm{S_{XP}} \text{ OR } \mathrm{S_{PL}} \text{ OR } \mathrm{S_{EB}} \text{ OR } \mathrm{S_{AG}} \text{ OR } \mathrm{S_{BO}}$$

$$<\textit{CarSearching}> = \textit{CarSearching}.\mathrm{f} = (\psi, \mathrm{join}), \text{ where}$$
$$\psi = ((\mathrm{S_{XP}} \text{ OR } \mathrm{S_{PL}}) \text{ AND } \mathrm{S_{AV}}) \text{ OR } \mathrm{S_{XP}} \text{ OR } \mathrm{S_{PL}}$$

$$<\textit{Cruise Searching}> = \textit{CruiseSearching}.\mathrm{f} = (\psi, \mathrm{join}), \text{ where}$$
$$\psi = \mathrm{S_{WC}}$$

$$<\textit{TrainSearching}> = \textit{TrainSearching}.\mathrm{f} = (\psi, \mathrm{join}), \text{ where}$$
$$\psi = (\mathrm{S_{WT}} \text{ AND } \mathrm{S_{EU}}) \text{ OR } \mathrm{S_{WT}}$$

$$<\textit{PromotionPack}> = \textit{PromotionPack}.\mathrm{f} = (\psi, \mathrm{join}), \text{ where}$$
$$\psi = \mathrm{S_{XP}} \text{ OR } \mathrm{S_{PL}} \text{ OR } \mathrm{S_{TP}}$$

Consequently, the full formula ψ in $Start.\mathrm{f} = (\psi, \mathrm{split})$ can be obtained by combinding the above *service-join* sub-formulas as (< > brackets are retained for the ease of reading without logical meaning).

$$\psi = (<((S_{XP}\ \mathrm{OR}\ S_{PL})\ \mathrm{AND}\ (S_{EB}\ \mathrm{OR}\ S_{EZ}\ \mathrm{OR}\ S_{RA}))\ \mathrm{OR}\ S_{XP}\ \mathrm{OR}\ S_{PL}> \mathrm{OR} <S_{XP}\ \mathrm{OR}\ S_{PL}\ \mathrm{OR}\ S_{EB}\ \mathrm{OR}\ S_{AG}\ \mathrm{OR}\ S_{BO}> \mathrm{OR} <((S_{XP}\ \mathrm{OR}\ S_{PL})\ \mathrm{AND}\ S_{AV})\ \mathrm{OR}\ S_{XP}\ \mathrm{OR}\ S_{PL}> \mathrm{OR} <S_{WC} > \mathrm{OR} <(S_{WT}\ \mathrm{AND}\ S_{EU})\ \mathrm{OR}\ S_{WT}>)\ \mathrm{XOR} <S_{XP}\ \mathrm{OR}\ S_{PL}\ \mathrm{OR}\ S_{TP}>$$

The RCOBDD tree of this formula is shown in Figure 6.4. From now on, let assume that all *service-split* formulas in this workflow are pre-processed into RCOBDDs (presented by relations (x, y), please see the complete detail in Table E.3 and graphical illustration in Appendix E).

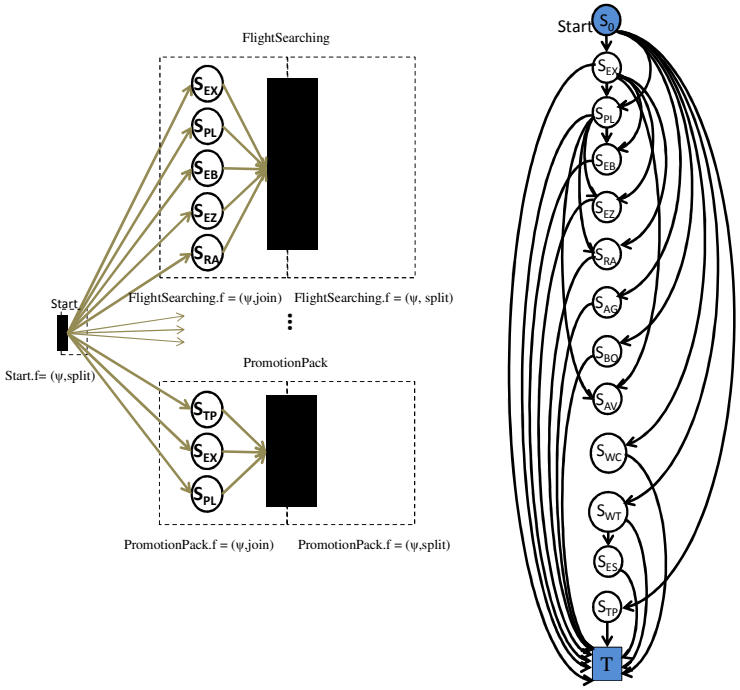


Figure 6.4: The RCOBDDs Form of $Start.\mathrm{f} = (\psi, \mathrm{split})$

6.4.3 Requirement Specifications

In this setting, services in the workflow possess properties and express their requirements in the form of TS formulas. All services retrieve the SWN description of the travel agent describing workflow structure, tasks and their dependencies, and services connected in each task encoded by RCOBDDs in both join and split types. As already mentioned, it is not always the case that each service will receive a complete view of SWN, such that the same requirement can be expressed in different forms of TS formula depending on the degree of visibility. The following shows some examples of important requirement specification cases.

1. S_{PL} at *FlightSearching HotelSearching* and *CarSearching* does not collaborate with S_{XP} due to the possibility of price comparison (Separation of Duty: R2).

   TS formula:

   $$\mathcal{HPA}_{\mathrm{FlightSearching}}(!\,\mathrm{XP})$$
   $$\mathcal{HPA}_{\mathrm{HotelSearching}}(!\,\mathrm{XP})$$
   $$\mathcal{HPA}_{\mathrm{CarSearching}}(!\,\mathrm{XP})$$

   Result: Partially satisfied as there is a possibility that S_{XP} is not part of an execution

2. The workflow owner does not trust HotChilly restaurant due to its bad reputation, so he requires that this restaurant service is not part of the workflow (Association between Tasks and Service: R3).

   TS formula:

   $$\mathcal{HPA}_{\mathrm{Restaurant}}(!\,\mathrm{HotChilly})$$

   Result: Satisfied as there is no HotChilly at *Restaurant*

3. The workflow owner requires that *Events* will be provided by services registered under local event authorities. Services that possess certificate singed by the local event authority (C_{LEV}) are trusted (Association between Tasks and Service: R3).

   TS formula:

   $$\mathcal{HPA}_{\mathrm{Events}}(\mathrm{CLEV})$$

Result: Partially satisfied as there is a possibility that services with C_{LEV} are part of an execution

4. S_{AG} and S_{BO} at *HotelReservation* trust only S_{MS} or S_{VS} for *Payment* and only one service must be invoked (Association between Tasks and Service: R3).

   TS formula:

   $$\mathcal{H}\forall \Diamond \mathcal{P}A_{\text{Payment}}(\text{VS} \boxplus \text{MS})$$
   $$\equiv \mathcal{H}\forall \Diamond (!\left(\mathcal{P}E_{\text{Payment}}(!\,\text{VS} \sqcup \text{MS}\right) \,\&\, (\mathcal{P}E_{\text{Payment}}(\text{VS} \sqcup !\,\text{MS}))$$

   Result: Partially satisfied as there is a possibility that S_{VS} and S_{MS} are not part of an execution which is the case that S_{PP} is activated

5. S_{PL} at *HotelSearching* requires that *HotelRating* must be provided by at least two different services (Association between Tasks and Service: R3).

   TS formula:

   $$\mathcal{H}\exists \Diamond (\mathcal{P}A_{\text{HotelRating}}(\text{D} \sqcap !\,\text{D})$$
   $$\equiv \mathcal{H}\exists \Diamond (\mathcal{P}A_{\text{HotelRating}}(\text{D} \sqcup !\,\text{D}) \,\&\, \mathcal{P}E_{\text{HotelRating}}(\text{D}) \,\&\, \mathcal{P}E_{\text{HotelRating}}(!\,\text{D}))$$
   $$\equiv \mathcal{H}\exists \Diamond (\,\mathcal{P}E_{\text{HotelRating}}(\text{D}) \,\&\, \mathcal{P}E_{\text{HotelRating}}(!\,\text{D}))$$

   Result: Partially satisfied as there is a possibility that both services are activated (this is because all services at *HotelRating* are connected by OR)

6. S_{XP} at *FlightReservation*, *CarReservation*, and *HotelReservation* incorporates with only S_{WI} for *Insurance* (Association between Tasks and Service: R3).

   TS formula:

   $$\mathcal{H}\exists \Diamond \mathcal{P}A_{\text{Insurance}}(\text{WI})$$

   Result: Partially satisfied as there is a possibility that S_{WI} is not activated

7. The workflow owner will operate with all car rental services with availability ratings over 0.8 (Association between Tasks and Service: R3).

   TS formula:

   $$\mathcal{H}\mathcal{P}A_{\text{CarSearching}}(\text{gt}(0.8))$$

$$\mathcal{HP}\mathrm{A}_{\mathrm{CarReservation}}(\mathrm{gt}(0.8))$$
$$\mathcal{HP}\mathrm{A}_{\mathrm{CarBooking}}(\mathrm{gt}(0.8))$$

Result: Satisfied as all services has reputation over 0.8 as their properties

8. S_{BO} at *HotelBooking* handles any request only made from S_{AG} and S_{BO} (Association between Tasks and Service: R3, and Flow- and Task-oriented Specification: R4).

   TS formula:

$$\mathcal{B}\exists \Diamond \mathcal{P}\mathrm{A}_{\mathrm{HotelSearching}}(\mathrm{AG} \sqcup \mathrm{BO}) \wedge$$
$$\mathcal{B}\exists \Diamond \mathcal{P}\mathrm{A}_{\mathrm{HotelReservation}}(\mathrm{AG} \sqcup \mathrm{BO})$$

   Result: Partially satisfied as there is a possibility that S_{AG} and S_{BO} are not the ones sending a request

9. S_{PL} at *HotelBooking* requires that *HotelSearching* must be provided by S_{AG} or S_{PL} (Association between Tasks and Service: R3, and Flow- and Task-oriented Specification: R4).

   TS formula:

$$\mathcal{B}\exists \Diamond \mathcal{P}\mathrm{A}_{\mathrm{HotelSearching}}(\mathrm{AG} \sqcup \mathrm{PL})$$

   Result: Partially satisfied as there is a possibility that S_{AG} and S_{PL} are not part of *HotelSearching*

10. S_{WI} provides *Insurance* only for the flight services certified by Business Airline Registry ($\mathrm{C}_{\mathrm{BAR}}$), hotel services certified by Business Hotel Registry ($\mathrm{C}_{\mathrm{BHR}}$), and Car rental services must operate in USA or Europe (Association between Tasks and Service: R3, and Flow- and Task-oriented Specification: R4).

    TS formula:

$$\mathcal{B}\exists \Diamond \mathcal{P}\mathrm{A}_{\mathrm{FlightBooking}}(\mathrm{BAR}) \wedge$$
$$\mathcal{B}\exists \Diamond \mathcal{P}\mathrm{A}_{\mathrm{HotelBooking}}(\mathrm{BHR}) \wedge$$
$$\mathcal{B}\exists \Diamond \mathcal{P}\mathrm{A}_{\mathrm{CartBooking}}(\mathrm{USA} \sqcup \mathrm{EU})$$

Result: Partially satisfied as there is a possibility that service at *FlightBooking* and *HotelBooking* have no C_{BAR} and C_{BHR} and some service at *CarBooking* are not from USA or EU

11. S_{BO} at *HotelBooking* can handle any reservation request from S_{AG} and S_{BO} and reservation must be made before booking (Association between Tasks and Service: R3, and Flow- and Task-oriented Specification: R4).

    TS formula:

    $$\mathcal{B}\forall \Diamond \mathcal{P}\mathrm{A}_{\mathrm{HotelReservation}}(\mathrm{AG} \sqcup \mathrm{BO})$$

    Result: Partially satisfied as there is a possibility that service at *HotelReservation* are not from S_{AG} or S_{BO}

12. S_{EZ} only provides *Insurance* to who travels by EazyJet meaning that *FlightReservation* and *FlightBooking* can be executed by only S_{EZ} (Flow- and Task-oriented Specification: R4).

    TS formula:

    $$\mathcal{B}\forall_{\mathrm{FlightBooking}} \odot \mathcal{P}\mathrm{A}_{\mathrm{FlightReservation}}(\mathrm{EZ}) \wedge \mathcal{B}\mathcal{P}\mathrm{A}_{\mathrm{FlightBooking}}(\mathrm{EZ})$$

    Result: Partially satisfied as there is a possibility that service at *FlightBooking* and *FlightReservation* are not from S_{EZ}

13. S_{EZ} also requires that it cooperates with only S_{WI} for providing *Insurance* (Association between Tasks and Service: R3).

    TS formula:

    $$\mathcal{H}\mathcal{P}\mathrm{A}_{\mathrm{Insurance}}(\mathrm{EZ} \sqcup \mathrm{WI})$$

    Result: Partially satisfied as there is a possibility that S_{WI} are not part of an execution

14. S_{BO} requires that after *HotelBooking* is done, subsequent services must belong to it itself, the workflow owner, or services executing *Payment* (Flow- and Task-oriented Specification: R4).

    TS formula:

$$\mathcal{HF}\mathrm{A}_{\mathrm{HotelBooking}}(\mathrm{BO} \sqcup \mathrm{OWNER} \sqcup \mathrm{PAYMENT})$$

Result: Partially satisfied as there is a possibility that *Events*, *Restaurant*, *GuideandLocalTour*, or *Insurance* are activated and all services for these tasks are not from S_{BO}, the workflow owner, or payment

15. All services at **Booking* require that all paths eventually reach to *Payment* (Flow- and Task-oriented Specification: R4, and Enforcement of Sequence: R5).

    TS formula:

    $$\mathcal{H}\forall \Diamond \mathcal{P}\mathrm{A}_{\mathrm{Payment}}()$$

    Result: Satisfied as all paths will eventually go to *Payment*

16. S_{TI} at *TouristInfo* will be invoked only after *Payment* is done (Enforcement of Sequence: R5).

    TS formula:

    $$\mathcal{B}\forall \Diamond \mathcal{P}\mathrm{A}_{\mathrm{Payment}}()$$

    Result: Satisfied as *TouristInfo* can be reached only after *Payment* is done

17. S_{VS}, S_{MS}, and S_{PP} execute *Payment* only if the service owner generates and sends the receipt to the customer right after *Payment* is done (Enforcement of Sequence: R5).

    TS formula:

    $$\mathcal{H}\forall \odot \mathcal{P}\mathrm{A}_{\mathrm{TravellingDocs}}(\mathrm{OWNER})$$

    Result: Satisfied as *TravellingDocs* is connected right after *Payment*

18. It is not necessary to have services S_{VS}, S_{MS}, and S_{PP} obtained the complete view of this SWN. In this scenario, these services receive the requests from **Booking* or *PromotionPack* before making *Payment* (Flexible Degree of Restriction: R6).

    TS formula:

    $$\mathcal{B}\forall \Diamond (\mathcal{P}\mathrm{A}_{\mathrm{FlightBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{HotelBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{CarBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{TrainBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{CruiseBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{PromotionPack}}())$$

Result: Satisfied as all paths to *Payment* are originated from **Booking* or *PromotionPack*

19. S_{BO} at *HotelSearching* requires all communication messages will be operated on SSL until *Payment* is made (Protection of Workflow Data: R7).

    TS formula:

    $$\mathcal{H}\forall[\mathcal{F}\mathrm{A}(\mathrm{SSL}) \Cup \mathcal{P}\mathrm{A}_{\mathrm{Payment}}()]$$

    Result: Partially satisfied as there are some services along the part from *HotelSearching* are not able to operate on SSL such as S_{PL}

20. S_{AV} requires that all **Booking* detail of its customer information will not be disseminated to other services, except services for *Payment* and services that belong to the workflow owner (Protection of Workflow Data: R7).

    TS formula:

    $$\mathcal{H}\forall\Box\mathcal{F}\mathrm{A}(\mathrm{PAYMENT} \sqcup \mathrm{OWNER} \sqcup \mathrm{AV})$$

    Result: Partially satisfied as there is a possibility that services not belong to these are part of an execution such as S_{E1}

21. S_{AV} provides *Insurance* to who rents a car from Avis (Flow- and Task-oriented Specification: R4).

    TS formula:

    $$\mathcal{B}\mathcal{P}\mathrm{A}(\mathrm{AV})$$

    Result: Partially satisfied as there is a possibility that services for CarBooking are not from S_{AV}

(Interoperability of Local Requirement: R1 is implicitly addressed by the above requirements as all TS formulas can incorporate with local requirements)

The cases above are a snapshot of some of the requirements that exist in the workflow domain. Table 6.1 shows the complete list of TS formulas and the compilation results of compliance checking discussed above.

Table 6.1: TS Formulas and Their Results

No.	Imposing Service	Originating Service	Trust Modes	Target Task	Requirements and Corresponding TS formulas	Result
1	S_{PL}	S_{PL}	BPT	*Searching	S_{PL} does not collaborate in all **Searching* with S_{XP}. $\mathcal{HP}A_{FlightSearching}(!XP)$ $\mathcal{HP}A_{HotelSearching}(!XP)$ $\mathcal{HP}A_{CarSearching}(!XP)$	*p* *p* *p*
2	Owner	-	ET	Restaurant	The workflow owner does not trust Hotchilly. $\mathcal{HP}A_{Restaurant}(!HotChilly)$	*S*
3	Owner	-	ET	Events	Only *Events* will be provided by services with C_{LEV}. $\mathcal{HP}A_{Events}(CLEV)$	*P*
4.	S_{AG}, S_{BO}	S_{AG}, S_{BO}	ET	Payment	S_{AG} and S_{BO}trust only S_{MS} or S_{VS} for *Payment*. $\mathcal{H}\forall \Diamond \mathcal{P}A_{Payment}(VS \boxplus MS)$	*P*
5	S_{PL}	S_{PL}	HPT	Hotel Rating	*HotelRating* must be provided by at least two different services. $\mathcal{H}\exists \Diamond (\mathcal{P}A_{HotelRating}(D \sqcap !D)$	*P*
6	S_{XP}	S_{XP}	HPT	Insurance	S_{EX} incorporates with only S_{WI} for *Insurance.* $\mathcal{H}\exists \Diamond \mathcal{P}A_{Insurance}(WI)$	*P*
7	Owner	-	ET	Car*	The workflow owner will operate with all car rental services with rating over 0.8. $\mathcal{H} \Diamond \mathcal{P}A_{CarSearching}(gt(0.8))$ $\mathcal{HP}A_{CarReservation}(gt(0.8))$ $\mathcal{HP}A_{CarBooking}(gt(0.8))$	*s* *s* *s*
8	S_{BO}	S_{BO}	BPT	Hotel Searching Hotel Reservation	S_{BO} can handle any request from S_{AG} and S_{BO}. $\mathcal{B}\exists \Diamond \mathcal{P}A_{HotelSearching}(AG \sqcup BO) \wedge$ $\mathcal{B}\exists \Diamond \mathcal{P}A_{HotelReservation}(AG \sqcup BO)$	*P*
9	S_{PL}	S_{PL}	BPT	Hotel Searching	S_{PL} requires that *HotelSearching* must be provided by S_{AG} or S_{PL}. $\mathcal{B}\exists \Diamond \mathcal{P}A_{HotelSearching}(AG \sqcup PL)$	*p*
10	S_{WI}	S_{WI}	BPT	Flight Booking Hotel Booking Car Booking	S_{WI} provides *Insurance* only for the flight services with C_{BAR}, hotel services with C_{BHR}, and Car rental services must operate in USA or Europe. $\mathcal{B}\exists \Diamond \mathcal{P}A_{FlightBooking}(BAR) \wedge$ $\mathcal{B}\exists \Diamond \mathcal{P}A_{HotelBooking}(BHR) \wedge$ $\mathcal{B}\exists \Diamond \mathcal{P}A_{CartBooking}(USA \sqcup EU)$	*p*
11	S_{BO}	S_{BO}	BPT	*Reservation	S_{BO} at *HotelBooking* can handle any reservation request from S_{AG} and S_{BO}.	*p*

					$\mathcal{B}\forall \Diamond \mathcal{P}\mathrm{A}_{\mathrm{HotelReservation}}(\mathrm{AG} \sqcup \mathrm{BO})$	
12	S_{EZ}	S_{EZ}	BPT	Flight Reservation Flight Booking	S_{EZ} only provides *Insurance* to who travels by EazyJet . $\mathcal{B}\forall_{\mathrm{FlightBooking}} \odot \mathcal{P}\mathrm{A}_{\mathrm{FlightReservation}}(\mathrm{EZ}) \wedge \mathcal{B}\mathcal{P}\mathrm{A}_{\mathrm{FlightBooking}}(\mathrm{EZ})$	*p*
13	S_{EZ}	S_{EZ}	BPT HPT	Insurance	S_{EZ} also requires that it cooperates with only S_{WI} for providing *Insurance*. $\mathcal{H}\mathcal{P}\mathrm{A}_{\mathrm{Insurance}}(\mathrm{EZ} \sqcup \mathrm{WI})$	*p*
14	S_{BO}	S_{BO}	ET	N/A	S_{BO} requires that after *reservation* is done, subsequent services must belong to it itself, workflow owner, or services for *Payment*. $\mathcal{H}\mathcal{F}\mathrm{A}_{\mathrm{HotelBooking}}(\mathrm{BO} \sqcup \mathrm{OWNER} \sqcup \mathrm{PAYMENT})$	*p*
15	All services from **Booking*		HPT	Payment	All services for **Booking* require that all paths eventually reach to *Payment.* $\mathcal{H}\forall \Diamond \mathcal{P}\mathrm{A}_{\mathrm{Payment}}()$	*s*
16	S_{TI}	S_{TI}	BPT	Payment	S_{TI} will be invoked only after *Payment* is done. $\mathcal{B}\forall \Diamond \mathcal{P}\mathrm{A}_{\mathrm{Payment}}()$	*s*
17	S_{VS},S_{MS}, S_{PP}	S_{VS},S_{MS}, S_{PP}	HPT	Travelling Docs	S_{VS}, S_{MS}, and S_{PP} execute *Payment* only if the owner will immediately generate and send the receipt to the customer. $\mathcal{H}\forall \odot (\mathcal{P}\mathrm{A}_{\mathrm{TravellingDocs}}(\mathrm{OWNER}))$	*s*
18	S_{VS},S_{MS}, S_{PP}	N/A	BPT	N/A	It is not necessary to have S_{VS}, S_{MS}, and S_{PP} obtained the full workflow description. In this scenario these services only receive **Booking* or *PromotionPack* before *Payment*. $\mathcal{B}\forall \Diamond (\mathcal{P}\mathrm{A}_{\mathrm{FlightBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{HotelBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{CarBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{TrainBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{CruiseBooking}}() \vee \mathcal{P}\mathrm{A}_{\mathrm{PromotionPack}}())$	*s*
19	S_{BO}	S_{BO}	HPT	N/A	S_{BO} requires all communication messages will be operated on SSL until *Payment* is made. $\mathcal{H}\forall[\mathcal{F}\mathrm{A}(\mathrm{SSL}) \Cup \mathcal{P}\mathrm{A}_{\mathrm{Payment}}()]$	*p*
20	S_{EB}	S_{EB}	HPT	*Searching	S_{AV} requires that all **Booking* detail of its customers information will not be disseminated to other services, except services for *Payment* and services that belong to the workflow owner $\mathcal{H}\forall\Box\mathcal{F}\mathrm{A}(\mathrm{PAYMENT} \sqcup \mathrm{OWNER} \sqcup \mathrm{AV})$	*p*
21	S_{AV}	S_{AV}	HPT	CarBooking	S_{AV} provides *Insurance* to who rents a car from Avis . $\mathcal{B}\mathcal{P}\mathrm{A}(\mathrm{AV})$	*p*

6.5 Characteristics Elicitation of Travel Agent Workflow

From this scenario, a number of characteristics can be elicited.

- *Workflow Formation and Enactment*: During workflow formation, TS formulas play a major role in assisting the workflow owner in the process of service acquisition. The owner uses TS formulas as a means to control and enforce services acquired for a task (requirements 2, 3 and 7), and the services impose their requirements for trust-based decision in participating the workflow (requirements 1, 4-6, and 8-21). After services are acquired, new TS formula can be created and existing formulas can be modified and terminated depending on changes with respect to current requirements. Service's properties can also be changed which requires the framework to monitor and compliance checking to be reactivated to check for further compliance.
- *Association between Tasks and Services*: As the workflow allows one competent service to be associated with multiple tasks, different TS formulas are applied differently to that service based on the task associated. For example, the requirement for service S_{AG} to provide *HotelReservation* in requirement 8 is different from the requirement 9 for *HotelSearching*.
- *Inter-domain and Inter-service relationship*: Inter-domain relationship between the workflow owner and prospective services occurs during service acquisition which can be determined from requirements by the workflow owner (requirement 2, 3 and 7). On the other hand, after services are acquired, inter-service relationship can be evaluated by their requirements. Compliance of these requirements is locally determined from each service that reflects the relationship inside a workflow (requirement 1, 4-6, and 8-21).
- *Dynamic and Static*: This travel agent workflow can exhibit in both static and dynamic characteristics. Services associated with *Events, Restaurant,* and *GuideandLocalTour* behave more dynamically by interoperating with local and constantly-changing services to adapt to the environment changes (e.g. S_{E1}, S_{E2}, S_{R1}, S_{G1}, …), while the rest of other services are considered more static. For instance, services for *Payment* are more static because it is unlikely to change, or have the small amount of changes overtime.

- *Delegation*: In this example, service delegation is not explicitly illustrated but can be observed at services for *Payment*. The workflow owner assigns the payment task with services from VISA, MASTER, and PAYPAL. PAYPAL, furthermore, can delegate its internal payment workflow to other services such as banks.
- *Global and Local Management*: The workflow owner has a full authority to coordinate the workflow tasks and acquire services. TS formulas are used to globally regulate how and which services are eligible determined from their properties. On the other hand, participating services preserve autonomy to locally manage their internal operations. They can specify arbitrary requirements to other services and tasks in the workflow.

6.6 Expressiveness of TS Formulas

From the previous section, it should already be clear how TS language can be used to represent many types of requirements in service workflow. To discuss the expressiveness of the language, consider the above twenty-one requirements as the examples to show how these formulas can represent the properties described in Chapter4, Section 4.6.

Self-complementary: Requirement 5 exemplifies self-complementary property by stating that *HotelRating* must be provided by at least two different services ($\mathcal{H}\exists \Diamond (\mathcal{P}\mathrm{A}_{\mathrm{HotelRating}}(\mathrm{D} \sqcap !\,\mathrm{D}))$.

Property-exclusive: Requirement 4 is property-exclusive as the formula $\mathcal{H}\forall \Diamond \mathcal{P}\mathrm{A}_{\mathrm{Payment}}(\mathrm{VS} \boxplus \mathrm{MS})$ restricts either S_{VS} or S_{MS} must be part of the *Payment* execution.

Property Composition: To be more specific, requirement 5 can be modified to address the property composition. For example, the formula is modified as $\mathcal{H}\exists \Diamond (\mathcal{P}\mathrm{A}_{\mathrm{HotelRating}}(\mathrm{AG} \sqcap \mathrm{BO})$ meaning that there must be S_{AG} and S_{BO} to provide *HotelRating* information.

Property Collaboration: Requirement 1 exemplifies property collaboration. For example, $\mathcal{H}\mathcal{P}\mathrm{A}_{\mathrm{FlightSearching}}(!\,\mathrm{XP})$ meaning that service S_{XP} is not part in *FlightSearching*.

6.7 Chapter Discussion

This chapter presents the case study of the travel agent workflow which provides online travelling services to its customers. This example illustrates how the SWN is constructed with task coordination and their dependencies, formulated by workflow connectives. Then, services are acquired if their properties are satisfied by the requirements from the workflow owner. The logic for service acquisition is based on the workflow owner that again utilises workflow connectives for *service-join* and *service-split*. Each service also possesses its own requirements presented in the form of TS formulas. In this scenario, some examples of requirement specification are presented which show how TS language can be incorporated and used in practice. The results of the compliance checking for each part of TS formulas are provided with detail. This scenario will be evaluated by the prototype implementation to be developed in the next chapter.

Chapter 7

Prototype Implementation and Evaluation

7.1 Chapter Introduction

In this chapter, the prototype of the framework is developed based on the algorithms outlined in Chapter 5. Its realisation and evaluation are demonstrated in the complete scenario of the travel agent workflow described in Chapter 6. In the first part of this chapter, the system architecture is presented. Then, the framework is evaluated with 4 experiments. The first experiment demonstrates the time performance of many types of the twenty-one requirements from Chapter 6, Section 6.3.3, which to some extent evidences that the compliance checking is practically feasible. The second experiment is to test the result of the performance between the atomic and non-atomic types of TS formulas, which corresponds to the 1st class and 2nd class algorithms. In terms of scalability, the third experiment evaluates the performance from the small to large number of services and TS formulas involved in the workflow. The final experiment provides the performance comparison of the algorithms for RCOBDDs with the existing CTT and EDNF algorithms. Finally, we demonstrate the power of TS language by showing that it can be used to address, or incorporate, the related works presented Chapter 3, Section 3.5.

7.2 Prototype Implementation

To confirm the applicability, the prototype is developed to evaluate the effectiveness of the framework in the travel agent workflow scenario. In this context, subjects, which are the workflow owner and services, are responsible to express their requirements in the form of TS formulas. The workflow owner uses the formulas as a mechanism to regulate and enforce how services are coordinated and what properties of services are required for participating in a workflow. On the other hand, the services use TS formulas to evaluate trust based on the compliance of requirements for participation decision.

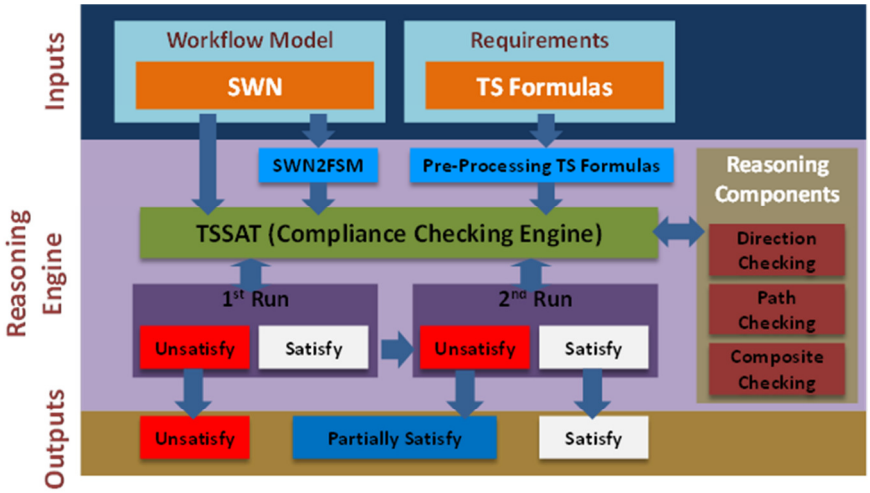


Figure.7.1: System Architecture

7.2.1 Usage Scenario

The usage scenario can be described briefly as follows. Firstly, a workflow owner creates a preliminary SWN with the coordination of tasks and imposes TS formulas according to his requirements. Then, potential services are acquired to complete the SWN. The compliance checking acts as a decision maker, checking for compliance of the owner's TS formulas based on properties of the prospective services[4]. After the SWN are plugged with services, its description will be reported back to the services to check for each service's TS formulas.

[4] At this point, all properties of services are assumed to be valid. There are several mechanisms to approve validity of properties; however these are not the essence in this thesis as we allow flexibility regarding various technical implementations. One common example is the use of attribute certificates for the proof of validity of properties.

Some services with positive compliance are likely to participate the SWN. For other services with negative compliance, conflicts will be reported back to the owner where further actions can be determined. For example, the conflicting services modify their TS formulas to be complied with the SWN, the workflow owner adjusts the SWN to avoid those conflicts, or simply remove the conflicting services from the SWN.

7.2.2 System Implementation

In this section, the system for the usage scenario outlined in the previous section is implemented for a proof of concept. The technical view of system architecture is shown in Figure 7.1. The core part of the implementation is the compliance checking engine which incorporates three key components including Direction Checking, Path Checking, and Composite Checking. To facilitate the development of this prototype, we developed a set of data structures to provide information encapsulation for the system. For example, the class TSFormula which is used to represent TS formulas can be inherited to the classes DirectionFormula, PathFormula, and CompositeFormula, which are all implemented by using binary tree. Processing of these classes is done by a depth-first search tree traversing from a single root node. Type of a visited node indicates which component to be activated. The components are recursively called until reaching a leaf node traversing up to the root node from which the final result is produced.

All compliance checking related functions are implemented on the basis of the algorithms outlined in Chapter 5. A flowchart in Figure 7.2 presents the flow of the activation processes of internal functions which can be described in the following steps.

1. The inputs consisting of TS formulas and SWN are prepared such that services are represented by RCOBDDs and TS formulas are transformed in the clove tree format.
2. TSSat reads these input files to start the checking operation. The formula tree is traversed and the corresponding functions, DirectionSat, PathSat, or CompositeSat, are activated based on the type of the current visited node.
3. CompositeSat that is the main function of the Composite Checking pane is activated to assess the composite part of the formula. Based on type of a TS formula, Classifier is the decision maker on which class of Composite algorithms to be activated. In the

1st class, Reachable is a core element in ForSomeSat1 and ForAllSat1, while in the 2nd class, ForSomeSat2 and ForAllSat2 are the least sub-functions in checking compliance for each possible execution enumerated by ExtractPathCheck. LogicalCheck is developed to deal with operators, & and ||.

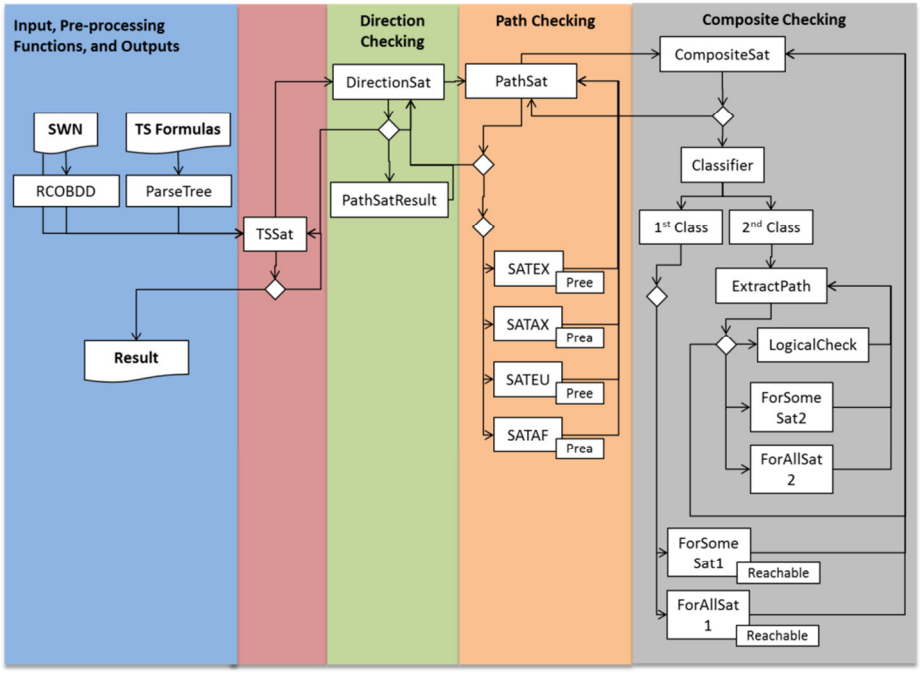


Figure 7.2: The Flowchart of Activation of Functions

4. CompositeSat marks the tasks that satisfies a Composite part of a TS formula which will later be used in the Path Checking pane.
5. The main function in the Path Checking pane is PathSat that is responsible for processing Path operators. For each operator encountered, it determines which sub-functions to be called. The sub-functions SATEX, SATAX, SATEU, and SATAF based on PREE and PREA are adequate to deal with all types of Path operators.
6. PathSat determines a set of tasks that satisfy the Path part of a TS formula where the results will be returned to the Direction Checking pane.
7. DirectionSat is responsible for processing all Direction operators. It obtains the results (a set of tasks) from the Path Checking pane and then PathSatResult is called

to evaluate the truth value of each Path term in a formula. From each result, DirectionSat determines the answer resulted from Direction operators.

8. After the truth value is assigned, the result will be sent back to TSSat. At this point, DirectionSat produces the result either *satisfied* or *unsatisfied*.
9. Finally, TSSat takes place to determine the final result according to the decision tree in Chapter 5, Section 5.6. If *ussatisfied* is obtained from the first run, we have *unsatisfied* as the final answer. Otherwise, TSSat will reactivate the checking process. Based on the result from the second run, If *unsatisfy* is obtained, the final answer is *partially satisfied*; otherwise it is *satisfied*.

7.3 Evaluation and Analysis

This section reports the evaluation results of the framework. The primary function of the prototype is to evaluate the feasibility o and scalability of the implementation. The prototype is applied to the collaborative travel agent workflow in the previous chapter. All functions are written in the MATLAB environment. (All program codes are attached in Appendix F.) The system runs on a Windows 7, Intel® Core™ i5-2435M CPU @ 2.40 GHz, 4 GB RAM, 64-bit Operating System.

The design of the evaluation consists of four experiments labelled by A, B, C, and D. In experiment A, the performance of the twenty-one requirements from Chapter 6, Section 6.3.3, is presented. Experiment B evaluates scalability in terms of the number of TS formulas and also illustrates the performance comparison between atomic and none-atomic types of Composite formulas corresponding to the use of the 1st class and the 2nd class algorithms. In experiment C, the performance is evaluated when a small to large number of services are involved. The number of services in an SWN is varied to assess the impact of time performance. Finally, the performance comparison of the algorithms CTTSat, EDNFSat, and RCOBDDSat specific to Composite part of TS formulas are presented. All results of each experiment are run with 50 iterations. The confidence interval of 95% ($P(-z \leq Z \leq z) = 0.95$) to indicate the reliability of an estimate is shown in Table 7.1. In this context the variation of the 50 iterations are assumed to be normally distributed.

Table 7.1. Confidence Interval of Each Experiment

	Types	$\bar{X}$	σ	$1.96\frac{\sigma}{\sqrt{n}}$	Bound	%
Experiment A (requirement 19)	CTTSat	0.785	0.123	0.034	0.75≤ μ ≤0.82	4.345%
	EDNFSat	0.552	0.065	0.018	0.53≤ μ ≤0.57	3.224%
	RCOBDDSat	0.613	0.061	0.017	0.6≤ μ ≤0.63	2.846%
Experiment B (Figure 7.5(e) at 1000 formulas)	Atomic	14.124	4.78	1.324	12.8≤ μ ≤15.45	9.375%
	Non-atomic	26.534	8.09	2.24	24.29≤ μ ≤28.77	8.445%
Experiment C (100 services in the worst case)	Atomic	22.65	3.98	1.1	21.55≤ μ ≤23.75	4.872%
	Non-atomic	898.23	285.92	79.2	819.03≤ μ ≤977.43	8.817%
Experiment C (100 services in the average case)	Atomic	21.85	3.14	0.87	20.98≤ μ ≤22.72	3.983%
	Non-atomic	676.18	100.77	27.91	648.27≤ μ ≤704.09	4.128 %
Experiment D (Worst case, Atomic)	CTTSat	1196	389.75	107.96	1088.04≤ μ ≤1303.96	9.027%
	EDNFSat	987	364.19	100.88	886.12≤ μ ≤1087.88	10.221%
	RCOBDDSat	896	183.43	50.81	845.19≤ μ ≤946.81	5.671%
Experiment D (Worst case, Non-atomic)	CTTSat	1196	389.75	107.96	1088.04≤ μ ≤1303.96	9.027%
	EDNFSat	987	364.19	100.88	886.12≤ μ ≤1087.88	10.221%
	RCOBDDSat	23.18	6.05	1.68	21.50≤ μ ≤24.86	7.229%
Experiment D (Average case, Atomic)	CTTSat	1196	389.75	107.96	1088.04≤ μ ≤1303.96	9.027%
	EDNFSat	685	191.55	53.06	631.94≤ μ ≤738.06	7.746%
	RCOBDDSat	22.98	5.08	1.41	21.57≤ μ ≤23.39	6.128%
Experiment D (Average case, Non-atomic)	CTTSat	1196	389.75	107.96	1088.04≤ μ ≤1303.96	9.027%
	EDNFSat	685	191.55	53.06	631.94≤ μ ≤738.06	7.746%
	RCOBDDSat	676	180.3	49.94	626.06≤ μ ≤725.94	7.388%

Experiment A

This experiment involves the test of time performance of the twenty-one requirements presented in Chapter 6, Section 6.3.3. The purpose of this experiment is to see if the running time is practically feasible for several different types of TS formulas, and to show the comparison among three algorithms, i.e., CTTSat, EDNFSat, and RCOBDDSat. This experiment is quite static as the number of services, tasks, and TS formulas, are fixed (79 services, 29 tasks, and 21 TS formulas). This test is run 50 iterations with the maximal bounds of confidence interval of 4.345% for CTTSat, 3.224% for EDNFSat, and 2.846% for RCOBDDSat.

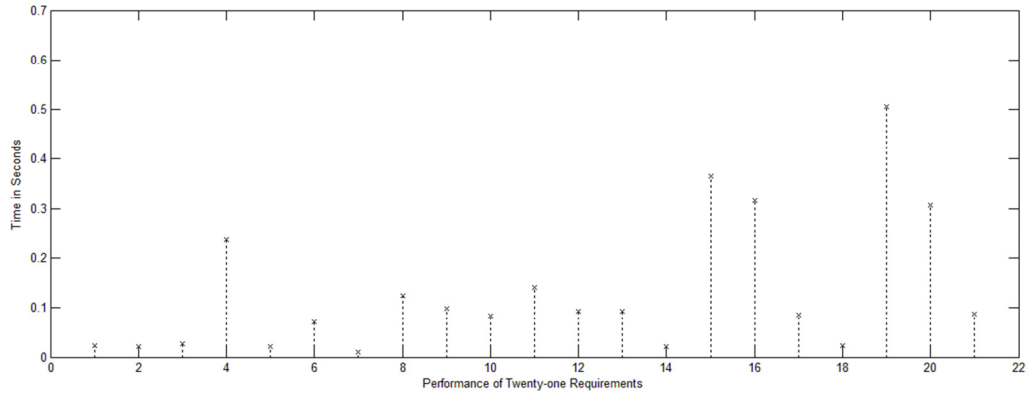


Figure 7.3: Performance of the Twenty-one Requirements in Section 6.3.3

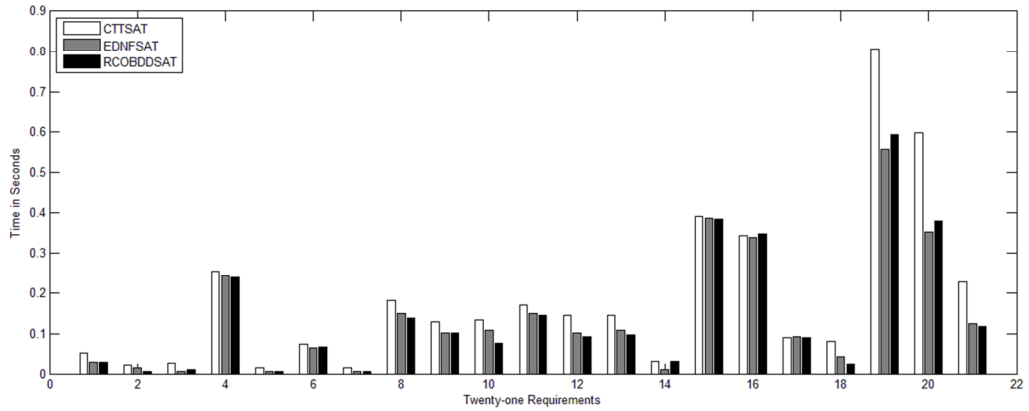


Figure 7.4: Comparison of Algorithm RCOBDDSat, EDNFSat, *and* CTTSat

Figure 7.3 shows that the time used for compliance checking of algorithm RCOBDDSat corresponding to the twenty-one requirements is practically feasible which is bound within 1 second, where most of them are lower than 0.5 (this will run much faster if implemented by low level languages such as C). In this experiment, it can be noticed that the atomic type of TS formula without Path operator in requirements 1, 2, 3, 7, and 14 has the best performance bound within 0.1. The non-atomic types of TS formula and the formula with Path operators require more processing time. Requirement 19 has the worst performance as the presence of $\mathbb{U}$ requires many iterations towards the complete workflow to determine a set of tasks that satisfy the formula.

The comparison of the three algorithms is illustrated in Figure 7.4. It can be noticed that no significant difference can be detected between algorithm EDNFSat and RCOBDDSat.

However, algorithm CTTSat has the worst performance, corresponding to the expected theoretical running time in Table 5.3, Chapter 5.

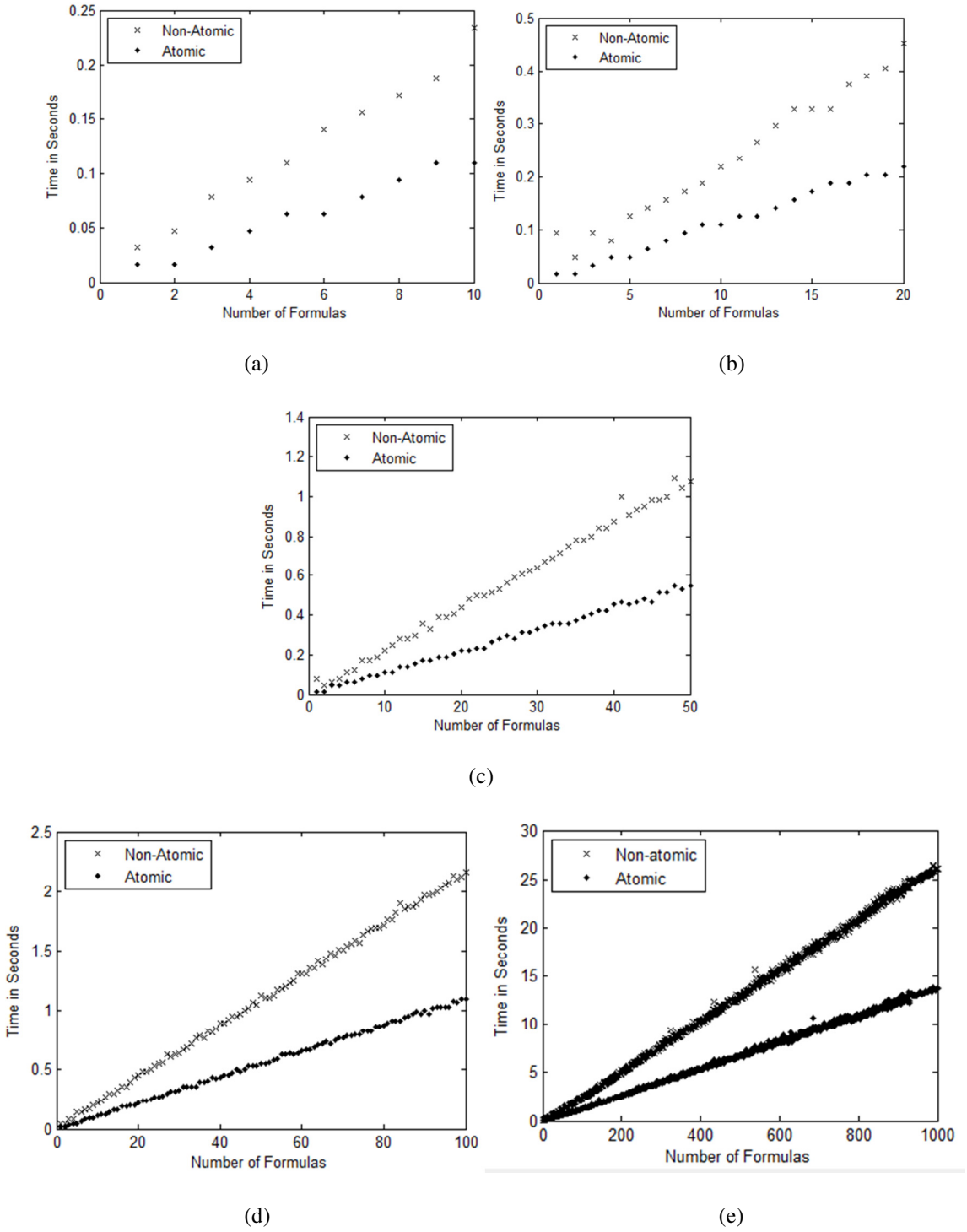


Figure 7.5: Performance Evaluation of 10, 20, 50, 100, and 1,000 Formulas

Experiment B

Experiment B studies the scalability of algorithm RCOBDDSat in terms of the number of TS formulas involved in an SWN. The amount of 10, 20, 50, 100, and 1,000 formulas is evaluated with the comparison between atomic and non-atomic types of TS formulas. The atomic and non-atomic formulas with the same meaning are used in this test which are $\mathcal{BP}\mathrm{A}_{\mathrm{FlightSearching}}(\mathrm{S}_1 \sqcup \mathrm{S}_1)$ and $\mathcal{B}(\mathcal{P}\mathrm{A}_{\mathrm{FlightSearching}}\mathrm{S}_1 \,||\, \mathcal{P}\mathrm{A}_{\mathrm{FlightSearching}}\mathrm{S}_2)$. Here, Path operators are not involved because the purpose of this test is to study the scalability of the algorithm for Composite formula.

In his experiment, the results in Figure 7.5 (a)-(e) indicate that the performance of atomic and non-atomic types is proportionally linear to the number of formulas with the maximal bounds of confidence interval 9.375% and 8.445% for atomic and non-atomic types. When 1000 formulas are presented, the time used for compliance checking of the non-atomic type is bound within 30 seconds, and atomic type is within 15 seconds. Compliance checking for the non-atomic type is computationally more intense than the atomic one. In average, the programme spends 0.025 and 0.01 second for the non-atomic and atomic types, respectively. This could vary depending on what TS formula is used in the test. In this analysis, checking for the non-atomic formula approximately requires 2.5 times longer compared to the atomic one.

Experiment C

The same formulas from Experiment B are used with 50 iterations. In this experiment, the number of services in the amount of 10, 20, 50, and 100 is evaluated for scalability. The performance comparison between non-atomic and atomic types is also given. If all Composite formulas are in the form of non-atomic type, theoretically, the performance of RCOBDDSat is similar to EDNFSat. However, the benefit of the RCOBDDs representation in RCOBDDSat is that it drastically reduces the processing time when the atomic type is presented. The comparison of the 1st class and 2nd class of RCOBDDSat is evaluated in the worst and average cases. The worst case is when all links are saturated in the RCOBDDs tree, while, in the average case, any link between two services is randomly generated.

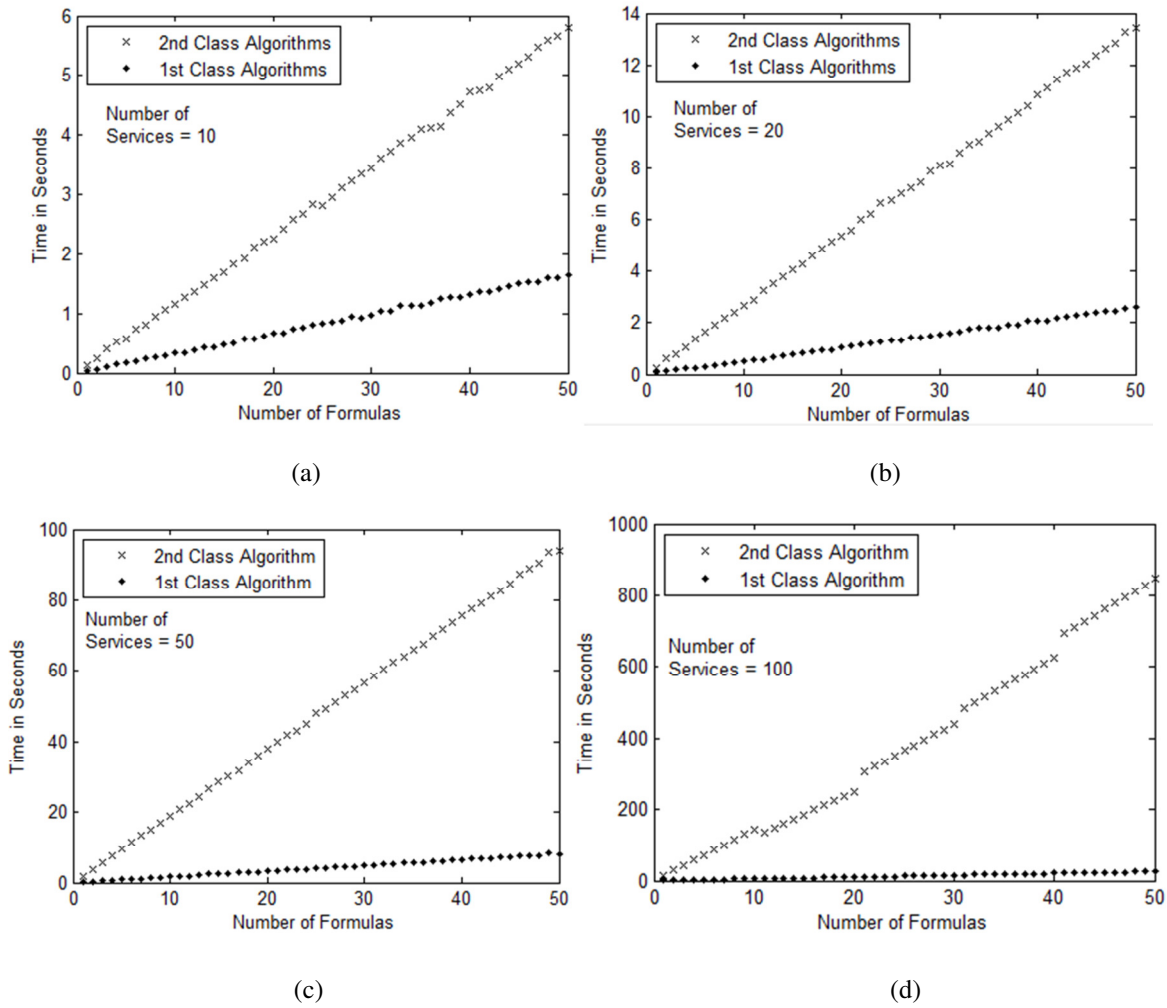


Figure 7.6: Performance Evaluation of 10, 20, 50, and 100 Services Involved

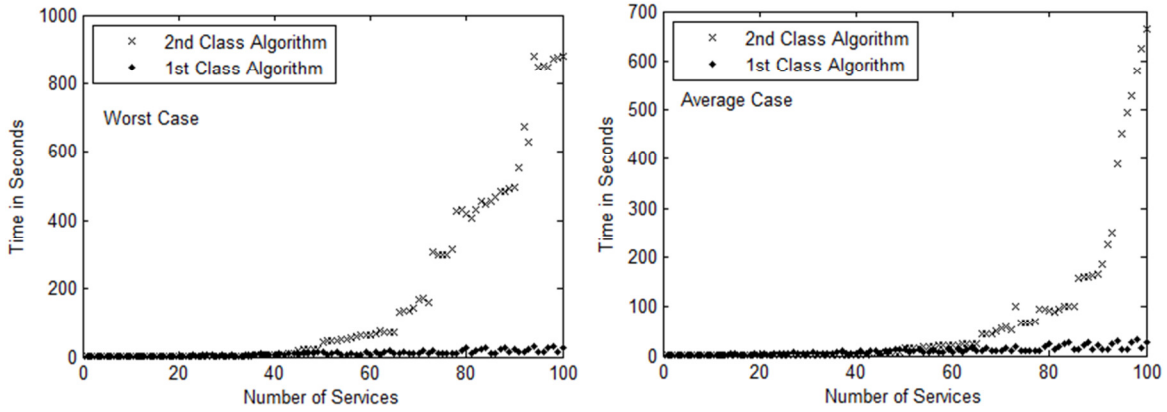


Figure 7.7: Comparison of the 1st and 2nd Class Algorithms

Figure 7.6(a)-(d) illustrate the experiment results corresponding to the number of 10, 20, 50, and 100 services, respectively, with the maximal bounds of confidence shown in Table 7.1. The experiment shows that when more services are involved, the checking time of the non-atomic type increases much faster than the atomic one. The reason is that the non-atomic type requires the activation of the 2^{nd} class algorithm that would extract every possible path of executions, which increases exponentially to the number of services. Figure 7.7(a) and (b) show the comparison between these two classes in the worst and average cases. The primary advantage of algorithm RCOBDDSat is that when a large number of services are involved, the algorithms can still operate in an acceptable time.

From this experiment, it can be concluded that the current implementation of the 1^{st} class algorithms can handle 10^4-10^5 services, which is more efficient than the 2^{nd} class algorithms that can handle 10^2 services in the worst case. (In reality, the higher number of services can be handled if it is implemented in C or C++ which is 10^2-10^3 faster [181]. As a result, the number of services that the algorithms can handle is approximately 10^7-10^8 for the 1^{st} class and 10^4-10^5 for the 2^{nd} class.) The higher number is expected in the average case.

Experiment D

The experiments detailed above were intended to ascertain the feasibility and scalability of the framework. In this experiment, the performance comparison of RCOBDDSat with the existing approaches, i.e., EDNFSat and CTTSat, is conducted. The same settings from the previous experiment are used with 100 services and 50 TS formulas. The algorithms run with the comparison in the worst and average cases and with the atomic and non-atomic types of TS formulas. The worst case here is set up by having all services in SWN formulas connected by OR, which implies that the size of CTT is 2^n-1 in the case of CTTSat, maximum terms (2^n-1) of EDNF in the case of EDNFSat, and all links are saturated in an RCOBDDs tree in the case of RCOBDDSat. In the average case, workflow formulas are randomly generated with the mixture of AND, OR, and XOR, which implies the average size of CTT (after the application of the reduction rules), the average number of terms of EDNF formulas, and the average number of links in an RCOBDDs tree.

In the worst case, the experiment depicted in Figure 7.8(a) and (b) shows the results of the performance comparison of CTTSat, EDNFSat, and RCOBDDSat for non-atomic and atomic types with confidence interval shown in Table 7.1. It can be noticed in Figure 7.8(a) that the performance of CTTSat, EDNFSat and RCOBDDSat using the 2nd class, in checking the non-atomic formula increases exponentially to the number of services, where CTTSat requires additional time for constructing CTT. This corresponds to the expected time in theory which is $O(|C||T||V|2^{|V|})$. However, in the atomic type of TS formulas in Figure 7.10(b), RCOBDDSat using the 1st class yields the greatest performance as expected to run with $O(|C||V| + |V|^2)$. As we can see, even for very dense links in RCOBDDs tree, RCOBDDSat works more efficiently than the other approaches.

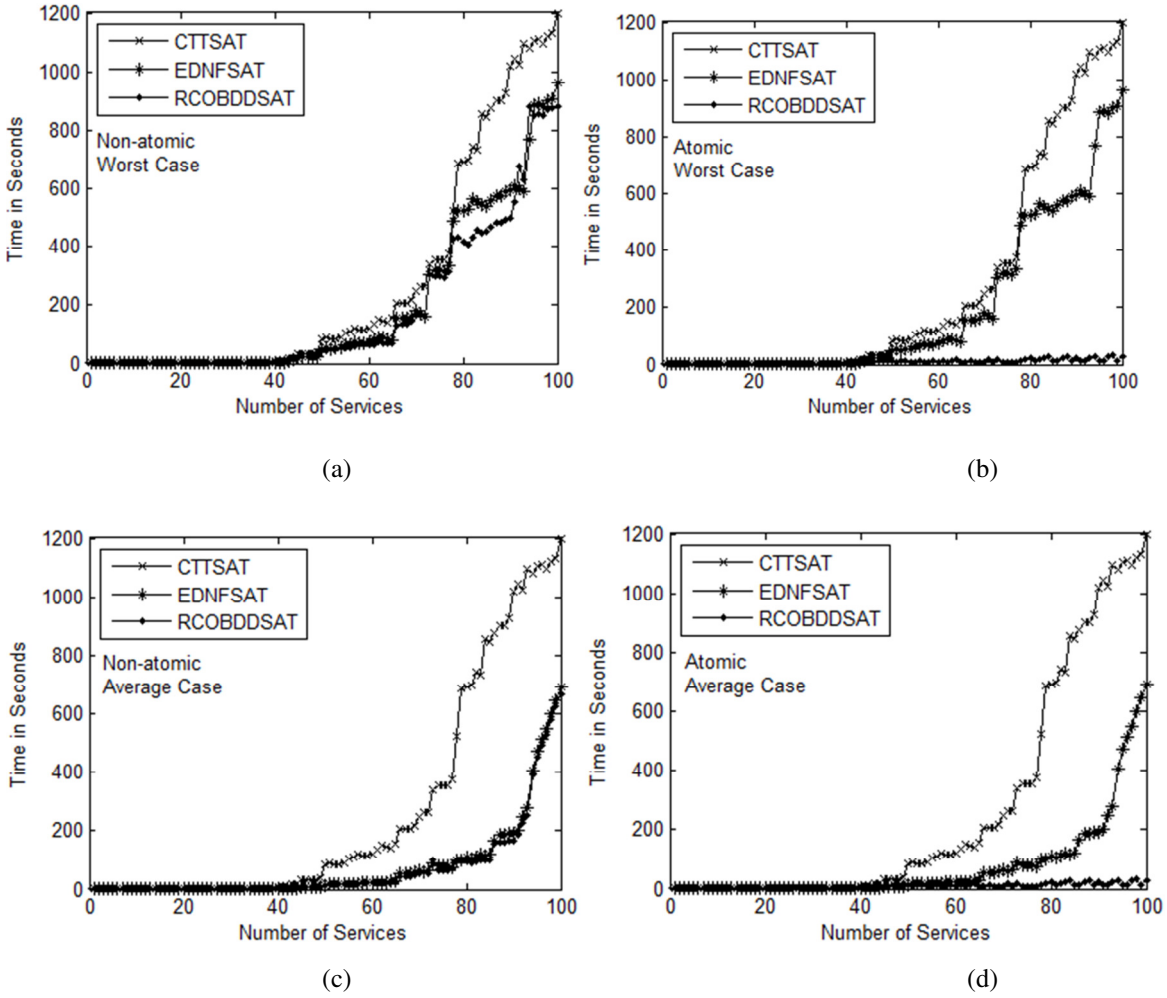


Figure 7.8: Comparison of CTTSat EDNFSat*, and* RCOBDDSat

Figure 7.8(c) and (d) depict the average running time of the three algorithms. Figure 7.8(c) demonstrates that time complexity for EDNFSat and RCOBDDSat reduces into polynomial, where the performance of CTTSat is the same as in the worst case. This is because the time spent for building CTT and applying reduction rules in CTTSat is still exponential to the number of services. Only EDNFSat and RCOBDDSat can gain the benefit from this setting due to the mixture of workflow connectives that reduces the number of terms in EDNF and the links in an RCOBDDs tree.

7.4 Chapter Discussion

This chapter presents the prototype implementation of the framework incorporating all elements from Chapter 3, 4, and 5. All functions are written in the MATLAB environment. To illustrate its practicality, four experiments are conducted in the travel agent scenario described in Chapter 6 with the following aspects. Firstly, it illustrates the practicality of the compliance checking algorithms with the performance of the twenty-one requirements from Chapter 6, Section 6.3.3. To make the compliance checking scalable to address large-scale service workflows, performance is always a major concern. The second and third experiments evaluate the scalability in terms of the number of TS formulas and services, respectively, with the comparison between 1^{st} and 2^{nd} class of algorithm RCOBDDSat. Additionally, the performance comparison of the existing approaches, i.e., CTTSat and EDNFSat is also presented in the last experiment. It is important to note that, in reality, the higher number of services can be handled if the framework is implemented in C or C++ which is expected to run 10^2-10^3 times faster [181]. As a result, the number of services that the algorithms can handle is approximately 10^7-10^8 for the 1^{st} class and 10^4-10^5 for the 2^{nd} class. In addition to this evaluation, the relation to the related works from Chapter 3, Section 3.5, is discussed. Although the framework is evaluated in the contrived scenario, it to some extent provides the identification of the possibility that the framework can be deployed in practice.

In future, we plan to develop an extension of trace semantics to support conflict analysis, which provides detection and identification of the parts of an SWN that cause conflicts.

There are two important features of the conflict analysis. Firstly, if a TS formula is not *satisfied* by an SWN, a counterexample would be generated. The second feature is to detect conflicts in TS formulas as well as to obtain the valid subsets, and possibly the invalid ones, of an SWN in the case of *partially satisfied*. The conflict analysis is important to provide a workflow owner with the useful information in a way that an SWN can be resolved. Please refer to Chapter 8 regarding deeper discussion of this issue.

In practice, each involved service is only to verify if its imposed requirements are complied with an SWN such that the compliance checking process can be locally computed. Furthermore, since requirements are independent from each other, using parallel computing will significantly improve the overall performance. From these two characteristics, the performance in practice would be much better than the results shown in the experiments.

Chapter 8

Conclusion and Discussion

8.1 Conclusion

The ‘everything as a service’ concept enables the proliferation of service interoperations in the form of service workflows. How to determine suitable services to be part of the workflow has become a challenging problem especially in the openness of environments. There are two implications from this statement. (1) In the viewpoint of the workflow owner, services to be acquired for a task are determined from requirements (indicated by TS formulas), reflecting trust-based decision making in service selection. (2) In the viewpoint of the participating services, the compliance of their requirements affects trust-based decision for workflow participation. Nowadays, the proliferation of services in open environments makes service workflows highly dynamic, as to be adaptable to environment changes. This thesis aims to solve, or at least alleviate, this problem by offering a framework to support trust-based decision for service workflow collaborations. The final solution is expected to encourage the proliferation of service provisions and consumption over the Internet.

The heart of this thesis is the requirement-oriented compliance checking framework in service workflows. Requirements from each party are encoded by TS formulas, serving as a formal and uniformed representation that can be automatically reasoned about a service workflow. The algorithms for the compliance checking are implemented based on this

language, coupled with the extension of syntax and semantics to be specific to service composition (*service-join*) and separation (*service-split*). The advantages of the framework in facilitating service workflow collaborations are described in the following ways. It supports (1) workflow scalability as the uniformed representation makes it possible for each domain to handle requirements in a comprehensive and consistent manner; enabling the same interpretation in different domains; (2) dynamicity as the logical representation makes the automatic compliance checking feasible, decreasing the cost of checking time, especially when the large number of requirements are involved and when there is a high frequency of changes of the requirements; and (3) consistency since dealing with disparate representations of requirements is no longer a problem.

The first part of this thesis introduces TS language for requirement specification in service workflow. It captures the important criteria needed for specification language in service workflows, such as Interoperability with Local Requirements (R1), Separation of Duty (R2), Association between Tasks and Services (R3), Flow- and Task-oriented Specification (R4), etc. (these criteria are specified in Chapter 3, Section 3.4). Essential algebraic operators are invented in three modes: HPT, BPT, and ET which are applied to service workflows abstracted by SWNs. The operators are classified into three categories. (1) Composite operators focus on the specification of the properties of service composition and separation associated with a task, (2) Path operators are used to quantify path and events over an SWN, and (3) Direction operators indicate the direction a formula to be applied in an SWN. To be able to reason about SWNs, syntax and semantics of TS language are formulated in Chapter 4 with satisfiability relations, algebraic properties, and expressiveness, providing a strong foundation for reasoning.

In the second part, the compliance checking algorithms are developed to check the compliance between TS formulas and SWNs. They return three possible answers: *satisfied*, *partially satisfied*, and *unsatisfied*, depending on the properties of services which constitute evidences that a TS formula complies, partly complies, or does not comply with an SWN. The algorithms are built corresponding to each part of operators. In the Composite part, a new algorithm has been developed based on a special data structure called RCOBDDs. From the result of Composite checking, which is considered the property of a task in an SWN, Path formulas takes place to determine the compliance in terms of path

quantification and sequence of tasks. The algorithm is built on the classical CTL labelling algorithm with the extension to deal with constrained tasks. Finally, the algorithm for Direction part is presented on the basis of SAT solver.

To confirm the applicability, the prototype is implemented and tested in the travel agent workflow scenario consisting of 29 tasks and 79 services. The performance is illustrated in four aspects: (1) with several types of requirements, (2) with different number of TS formulas and (3) services, and (4) with the comparison of the existing works, i.e., CTTSat and EDNFSat. The first experiment shows that compliance checking is practically feasible which is bound within 1 second. Most of them are lower than 0.5. The second and third experiments involve the test of scalability where the result illustrates that the current implementation of the 1^{st} class algorithms is anticipated to handle 10^4-10^5 services, which is more efficient than the 2^{nd} class algorithms that can handle 10^2 in the worst case scenario. Even though the prototype can handle 10^2 services which may be questionable in terms of the scalability support, it drastically reduces the time of manually verifying of requirements by automating the process of compliance checking. This supports scalability from the beginning stage when the absence of automatic compliance checking requires the high overhead of human intervention for manually checking. Moreover, the higher number of services can be handled if implemented in the low level languages such as C or C++ which is expected to run 10^2-10^3 faster [181]. As a result, the anticipation of the number of services to be handled is approximately 10^7-10^8 for the 1^{st} class and 10^4-10^5 for the 2^{nd} class. The final experiment shows that the algorithm based on RCOBDDs provides the best performance in checking time compared to the existing works, i.e., CTT and EDNF.

As the experiments illustrated, the evaluation shows that the algorithms has the acceptable performance and provides guaranteed results. In terms of optimisation, there are two characteristics that amortise the cost of compliance checking. Firstly, since each service only concerns to verify whether its requirements are complied with an SWN, the compliance checking process can be distributedly and locally computed. Secondly, since requirements are independent from each other, using parallel computing will significantly improve the overall performance.

In future, we plan to (1) improve the algorithms to deal with the larger number of services, and (2) apply the framework to the real cases to further validate the applicability and usability. Although the framework is designed to be used in the restricted domain of service workflows on the basis of Petri Net, it can be extended to broader domains such as business process. It is also applicable to other graph-based workflow or any modelling language that can be semantically translated to SWNs such as Pi calculus and UML activity diagrams. The following section identifies and discusses the important issues and future developments.

8.2 Discussion

8.2.1 Potential Threats and Vulnerabilities

The openness of environments places strong needs on a service workflow to be adaptable to environment changes. This includes the change of service providers and their workflow participation requirements. As such, a workflow is likely to be governed by requirements, and in a large-scale service workflow with a large number of requirements, the verification of such requirements is a critical problem for dynamic workflow interoperation.

One potential threat occurs when the verification process is not finished and a workflow is needed to be deployed. This creates a serious vulnerability that the workflow can be executed with some invalid requirements which can result in several threats such as data leakage and modification, low quality of services, financial fraud, etc.

Another threat is the dynamic change of the workflow structure which affects the coordination of tasks and services. Any TS formula specified before the change might no longer be useful to address the same requirements. This also creates a serious vulnerability since the workflow can be executed with no conflicts while the existing formulas are not yet to be modified according to the requirements.

It can be observed that the above threats are rooted from the problem of managing requirements in a large-scale dynamic environment where the intended requirements cannot be applied or verified correctly during workflow executions. This requires a new

requirement management solution and an effective requirement verification technique such that the requirements can be updated according to the change of workflow structure and the verification of these requirements are done before a workflow is executed.

8.2.2 Petri Net as a Workflow Language

There is an on-going debate on the choice of the foundation for workflow modelling. Among these many choices, Petri Net, Pi Calculus, and UML Activity Diagrams receive the most attention in research and commercial areas. Clearly, all of them offer advantages for reasoning since they come with formal semantics and analysis methods that can be exploited in workflow modelling and verification. In the following paragraphs, key aspects regarding important features of such languages are identified with comparison.

Commercial Applications and Research Areas

In the commercial sector, Petri Net has been widely used to model workflow systems in a variety of application domains [152], including protocols, manufacturing systems, and business processes. At the moment, there are more than 200 workflow products commercially available based on Petri Net to support their business processes [182]. It is obvious that Petri Net is more popular and highly adopted in the commercial area than the others, i.e., Pi Calculus and UML Activity Diagrams.

In the research aspect, majority of research literature on workflow modelling languages focuses on Petri Net and Pi Calculus. They have been used to model and analyse several types of workflow-related processes such as Web Service and service composition, Business Process, Network protocols, hardware, and embedded systems. Since both of them have the well-defined process modelling techniques with strong mathematical foundation, they are considered more attractive choices than UML Activity Diagrams.

Usability by Visualisation

UML Activity Diagrams and Petri Net offers visualisation capability based on their formal semantics support at the background. This makes them self-documented and powerful

design tools [129]. No further, or little, verbal description is unnecessary to describe a workflow. Regarding the implementation, the graphical representation assists a workflow designer to design, test, and validate workflows at a conceptual level which can be easily tied into design specification and algorithms [183]. Another benefit of visualisation is that it is compatible with the nature of WFMS which is the explicit representation of the business process logic. In summary, graphical representations provide a high-level yet precise language which allows ones to express and reason about the concepts at their natural level of abstraction [131].

Reactive and Non-Reactive Systems

Workflow systems are considered reactive which can run in parallel with its environment and responds to internal/external input events. Since Pi Calculus is an event-based modelling language, it is suitable for modelling a workflow in this situation. However, although the original notations of UML Activities Diagrams and Petri Net is relatively closed to environments, their semantics can be extended efficiently to capture external events ([184], [185], and [186]).

Domain of Ownership

Workflow management originally focuses on improving the efficiency of business processes within a single domain. However, today's collaborations are challenged to cross organizational boundaries [182]. Each of the parties has its own local workflow which is under controlled within the domain of ownership. These local workflows need to communicate vertically and horizontally to form the collaborations. Vertical communication situates in a nested workflow where one activity in a parent workflow is executed by a sub-workflow, while horizontal communication situates among peer workflows at the same level. Workflow modelling languages should explicitly support the identification of these communications. Among the three workflow-modelling languages, Petri Net and UML Activity Diagrams have an explicit capability to separate domain of ownership compared to Pi Calculus.

Extensibility

Since Petri Net, Pi Calculus, UML Activity Diagrams have already existed before the WFMS technology was invented, their original semantics is not specifically developed to support workflow modelling. Many works ([128] and [184]) attempted to extend these semantics into the workflow domain. Petri Net and UML Activity Diagrams support extensibility by introducing a new set of elements into their definitions, while Pi Calculus uses its fundamental elements to capture workflow behaviour. Among these modelling languages, Petri Net has captured a great amount of interests and is still an active research area. There are a number of conferences mainly focused on the Petri net-based modelling of workflows [129]. In terms of usability, workflow composers usually prefer Petri Net and UML Activity Diagrams due to their visualisation capability making them less complexity from the design point of view.

Despite their wide range of applications, workflow systems still suffer from the lack of an agreed and standard modelling technique [129]. Among the other techniques, Petri Net is widely used with both qualitative and quantitative analysis to abstract workflows. In our respect, there are several reasons for selecting Petri Net as a workflow modelling language.

1. Petri Net can be used to model a workflow in the control and resource dimensions. The control dimension concerns the ordering of tasks and their execution logic while the resource dimension concerns with semantics of resources being used during workflow executions. These can be abstracted into the SWN representation that explicitly incorporate the control dimension by workflow connectives, and the resource dimension by using the place notion in Petri Net as services.
2. Petri Net sharply distinguishes between places (services) and transitions (tasks). This clearly expresses Association between Tasks and Services (R3) and assists security analysis such as Separation of Duty (R2).
3. Petri Net has mathematically-based formal semantics despite the graphical nature, making it a powerful design tool which is consequently widely adopted in commercial workflows than the others.

4. Petri Net is state-based which facilitates compliance checking algorithms. Automatic compliance checking is needed, allowing workflows to become flexible and adaptable to cope with rapid changing environments.
5. Petri Net has been adopted in a wide range of applications. Building compliance checking on this ground would yield a good benefit and contribution.
6. Petri Net has graph-like representations that have useful links to both graph theory and algebra. On the other hand, any graph-based workflow can also be mapped into Petri Net, where the theories developed on Petri Net can be applied.

In summary, the right choice of the language for workflow modelling is an open on-going argument. Different languages have different advantages depending on which aspects being approached. Instead of event-based which is considered more reactive in Pi Calculus and UML Activity Diagrams, Petri Net is a state-based language that seems to be rigorous in modelling workflow control flows. A claim of using Pi Calculus to be able to accommodate all types of workflow patterns [128] at the low level of abstraction and some variants to accommodate resource patterns make it complexity overwhelming for modelling and reasoning about a workflow in our context Moreover, usability is also problematic as witnessed by which there are a number of workflow products based on Petri Net rather than Pi Calculus and UML Activity Diagrams. In term of usability, people familiar with formal methods are fighting religious wars driven by commercial arguments instead of the desire to build a solid foundation [187]. Consequently, it can be concluded that Petri Net and Pi Calculus are better together as their semantics can be translated from one to another [6]. By doing so, their inherent weaknesses can be complement.

Finally, we do not argue that Petri Net as a workflow language is the best choice for workflow modelling. Our focus neither intends to describe a detailed behaviour of a workflow, nor the external inputs, but instead, the intension of using Petri Net is to abstract and advertise a workflow in a way that it can formally encode the information about the association between services and tasks which can be used for compliance checking. Among other modelling techniques, or at least to our current knowledge, Petri Net is the most attractive choice. A workflow modelled by other languages can be translated into Petri Net for checking satisfiability.

8.2.3 Conflict Discovery Enhancement

In practice, conflicts may arise when an SWN does not comply with a TS formula. This becomes more problematic in open environments when there is a high frequency of changes in requirements, or services, which potentially increase the amount of conflicts. This calls for the activation of checking process each time when a change is identified. In the case of manually checking, a significant amount of time is consumed in identifying parts of workflows affected by a new change. As a result, it is desirable for compliance checking algorithms to be enhanced with the ability to detect and identify conflicts along the side of providing verification results. This information is valuable for a workflow owner and participating services to efficiently find conflict resolution. It enables a quicker reaction to address the conflict parts of an SWN and TS formulas.

However, one limitation of our compliance checking algorithms is that they are unable to produce the information about such conflicts. Although the algorithms based on RCOBDDs have decent efficiency, they cannot identify points of conflicts in an SWN or TS formula. Therefore, the identification regarding the possible subsets of an SWN for valid executions cannot be obtained. As a result, the detailed analysis in the results of *partially satisfied* and *unsatisfied* cannot be given. This means that the framework neither support the generation of counter examples, or traces, nor produce the valid subsets of an SWN.

In future, we plan to develop an extension of trace semantics to support conflict analysis, which provides the detection and identification of parts of an SWN and TS formulas that cause conflicts. There are two important features of the conflict analysis. Firstly, if a TS formula is not *satisfied* by an SWN, a counterexample should be generated. However, for many service providers, this counterexample may not be meaningful. A high level virtualisation tool must be provided to map the conflicts back to an SWN. The second feature is to detect conflicts in TS formulas as well as to obtain the valid subsets, and possibly the invalid ones, of an SWN in the case of *partially satisfied*. We believe that conflict analysis is essential in dynamic service composition in the service workflow context. It provides a workflow owner with the information in a way that an SWN can be

resolved. From the service side, the conflicted requirements are identified which consequently affects the trust-based decision in workflow participation.

8.2.4 Performance Optimisation

As mentioned, high frequency of changes in open environments requires the compliance checking to be periodically activated. To make the compliance checking framework more applicable to large-scale service workflows, performance is always a major concern. Therefore, optimising the running time of the checking operations would yield a great contribution, especially during runtime monitoring. The cost depends on how many checks to be performed, and the number of services, tasks, and TS formulas. In this section, three potential ways are discussed for optimisation. Firstly, when services in an SWN are changed due to the inclusion of new services, termination of existing services, and the changes of their properties, the identification of the affected TS formulas would reduce the scope of TS formulas to be rechecked. This approach guides a service provider with the affected TS formulas to be checked for further compliance. Secondly, state space can be reduced due to the effect of constrained tasks that reduce the scope of the SWN to be checked. Tasks that are not part of a TS formula can be excluded from the check. Thirdly, the 2^{nd} class algorithms which currently run in exponential time can be improved. We plan to develop new algorithms that can reduce the checking time. Achieving these would reduce the time for compliance checking and support lightweight monitoring for runtime verification.

8.2.5 Scalability, Usability, and Extendibility

In the past few years, many organisations employ WFMS to improve their internal business process efficiency. Due to the openness of environments nowadays, the process of service acquisition is more complicated as a result from the availability of the large number of services. This consequently makes the compliance checking process to be intensively exercised. In order to support scalability, not only an automated compliance checking is needed, but also the checking process must run with acceptable performance. At present, our compliance checking algorithms can handle 10^7-10^8 for the 1^{st} class and 10^4-10^5 for the

2^{nd} class, which is considered sufficient as the average number of services in a service workflow today is approximately 10^2. However, the number of services is expected to be increased in the near future. More efficient algorithms are required, which are left for future investigation.

In terms of usability, because the framework is intended to be used by workflow owners and participating services, which are considered unfamiliar with logical languages and model checking, there are two aspects to be addressed. The first aspect is the understandability. Since logical languages cannot be avoided for reasoning purpose, there should be a visual tool to assist workflow owner and participating services in not only translating requirements into TS formulas, but also positioning the error points within an SWN. If *partially satisfied*, or *unsatisfied*, is the final answer, this tool should generate counterexamples at the level that can be comprehended by workflow owners and services. The use of the high-level tool that automatically transforms requirements into low level formulas yields easier, and more intuitive in the process of requirement specification; as a result, reducing the risk of errors and improving the efficiency of specification. The second aspect is related to the performance of compliance checking algorithms. As service workflows are anticipated to be more complex in future, optimisation of algorithms helps improve the overall performance.

In future, we plan to integrate the framework into Business Process Execution Language (BPEL4WS) [188]. Although there is no formal proof that BPEL is powerful enough to express all requirements related to business processes, BPEL has been applied in many real customer cases and enjoys broad industry acceptance [174]. It becomes a de facto standard in describing the communication behaviour of an individual service at the message level. As a result, integrating TS formulas into BPEL would equip ones with capability of specifying finer-grained requirements to each communicated message. Furthermore, since BPEL can be translated into Petri Net [189], and vice versa, developing compliance checking on the basis of Petri Net implies the applicability to BPEL.

8.2.6 Service Explosion Problem

Like the state explosion problem that traditional model checking encounters, our framework experiences another aspect of service explosion problem that could cause the compliance checking prohibitively expensive. In traditional model checking, several techniques have been exercised to address this problem ([169] and [170]). However, most of them produce a nondeterministic result which cannot guarantee that a system is free from conflicts. This is undesirable in our context since we require deterministic results in order to generate the valid subsets of an SWN that make executions free from conflicts. Therefore, the algorithms have to exhaustively search all the space of an SWN to discover possible violations. The number of services is the major factor to the expensiveness of our compliance checking.

There are two aspects to alleviate this problem. Firstly, we exploit workflow formulas to reduce the number of workflow executions, which simultaneously reduces the time of compliance checking. For example, suppose that there are three services *A*, *B* and *C* involved in a task, the maximum number of possible executions without the workflow formulas could be $2^3 - 1 = 7$, where -1 means all *A*, *B*, and *C* are not presented. To reduce the number of executions, one example formula $\mathrm{A\ OR\ (B\ AND\ C)}$ whittles the number of executions down to 3, i.e., *A,* *B* and *C,* and *A* and *B* and *C*. The second aspect is the development of a data structure that can improve efficiency of the compliance checking process. Unlike symbolic model checking that employs OBDDs as a representation of formulas, we introduce RCOBDDs, a simplified version of OBDDs, to be applicable to represent workflow formulas. By applying the reduction rules, the size of an RCOBDD tree is radically reduced and linear to the number of services.

In the aspect of system characteristics, since the number of services in a workflow is comparatively much smaller than the states in complex software systems, we encounter less impact of the explosion problem.

8.2.7 Expressiveness and Expensiveness

As mentioned, one limitation of our framework is that it lacks of capability to distinguish between the quantifications of workflow paths and instances. Specifically, consider the

definition of $\exists_t$ which only describes that it is always possible to reach the target from a current position, but it cannot guarantee that the target is always executed. $\forall_t$, on the other hand, describes that it always gets to the targets from a current position, with guarantee that the targets are always executed in any execution. To explain the differences, consider the following example. Two snapshots of SWNs are illustrated in Figure 8.1(a) and (b) with OR-split and AND-split at t_1 and then joined at t_6. Suppose that t_4 has property q_1. For formula $\exists_{t_1} \Diamond q_1$, the same *satisfied* results are obtained from both SWNs in Figure 8.1(a) and (b), even though not all executions in Figure 8.1(a) involve t_4, such as one sequence of an execution of Figure 8.1(a) can be t_1, t_3, t_5, t_6, t_7. Unlike in Figure 8.1(b), t_4 is always executed.

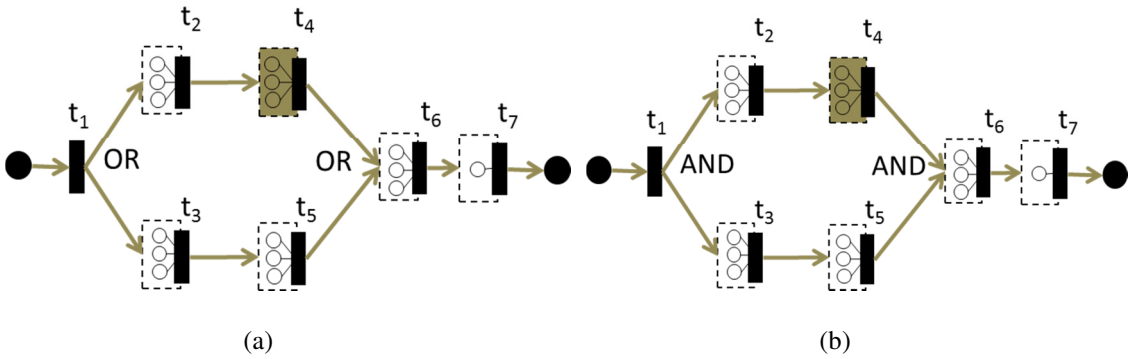


Figure 8.1: Two Example Snapshots of SWNs

To differentiate the quantification of paths and instances, more expressive operators are required. For example, TS language can be extended with the operators *possibly* and *necessarily* defined in modal logic. Operator *possibly* can be used to address for some instances and operator *necessarily* can be used for all instances. Incorporating with for some path ($\exists_t$) and for all path operators ($\forall_t$), we are able to quantify paths and instances over an SWN. However, an important issue to be considered is the trade-off between the expressiveness and expensiveness. In future research, we plan to explore this issue by extending the compliance checking algorithms to be able to deal with all quantification types and conducting evaluation and comparison with the existing approach.

Bibliography

1. G. C. Fox and D. Gannon, Workflow in Grid Systems, Concurrency and Computation: Practice and Experience, vol 18, pp 1009-1019, 2006.
2. A. D'Agostino, R. Serafini, and M. Ward-Warmedinger, Sectoral Explanations of Employment in Europe. The Role of Services, ECB Working Paper Series no. 625, European Central Bank, 2006.
3. S. Rajbhandari, O. F. Rana, and I. Wootten, A Fuzzy Model for Calculating Workflow Trust Using Provenance Data, In ACM Mardi Gras Conf., pp.10, 2008.
4. W. Viriyasitavat, and A. Martin, A Survey of Trust in Workflows and Relevant Contexts, IEEE Communications Surveys and Tutorial, vol.14, no. 3, pp. 911 –940, 2011.
5. S. D. Ramchurn, C. Sierra, L. Godo, and N.R. Jennings, A Computational Trust Model for Multi-Agent Interactions based on Confidence and Reputation, In Proc. of the 6th Intl. Workshop of Deception, Fraud and Trust in Agent Societies, pp. 69-75, 2003.
6. N. Busi and R. Gorrieri, A Petri Net Semantics for Pi-Calculus, In Proc. of the 6th Intl. Conf. on Concurrency Theory (CONCUR '95), LNCS, Springer-Verlag, vol. 962, pp. 145-159, 1995.
7. J. O'Donovan and B. Smyth, Trust in Recommender Systems, In Proc. of the 10th Intl. Conf. on Intelligent User Interfaces, pp 167-174. ACM Press, 2005.
8. J. D. Lewis and A. J.Weigert, Trust as a Social Reality. Social Forces, Social Atomism, Holism, and Trust, The Sociological Quarterly, vol. 63, pp. 967-985, 1985.
9. S. P. Shapiro, The Social Control of Impersonal Trust, American Journal of Sociology, vol. 93, pp. 623-658, 1987.
10. J. B. Rotter, A New Scale for the Measurement of Interpersonal Trust, Journal of Personality, vol. 35, pp. 651-665, 1967.
11. J. G. Holmes, Trust and the Appraisal Process in Close Relationships, In Jones, W. H. and Perlman, D. (Eds.), Advances in personal relationships, vol. 2, pp. 57-104, 1991.
12. O. E. Williamson, Calculativeness, Trust, and Economic Organization, Journal of Law and Economics, vol. 34, pp. 453-502, 1993.

13. M. Deutsch, Trust and Suspicion, Journal of Conflict Resolution, vol. 2, pp. 265-279, 1958.
14. R. C. Mayer, J. H. Davis, and F. D. Schoorman, An Integrative Model of Organizational Trust, Acad. Management Rev., vol. 20, pp. 709-734, 1995.
15. T. Yu, M. Winslett, and K. E. Seamons, Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation, ACM Transactions on Information System Security, vol. 6, pp. 1-42, 2003.
16. A. Kini and J. Choobineh, Trust in Electronic Commerce: Definition and Theoretical Consideration, In Proc. of the 31st Intl. Conf. on System Sciences, IEEE, 1998.
17. R. Falcone and C. Castelfranchi, Social Trust: A Cognitive Approach, Kluwer, pp. 55-99, 2001.
18. D. Olmedilla, O. Rana, B. Matthews, and W. Nejdl, Security and Trust Issues in Semantic Grids, In Proc. of the Dagsthul Seminar, Semantic Grid: The Convergence of Technologies, vol. 05271, 2005.
19. T. Grandison and M. Sloman, A Survey of Trust in Internet Applications, IEEE Communications Surveys and Tutorials, 2000.
20. L. Lau and B. Whyte, Technological Aspects of TRUST in Virtual Organisations, School of Computer Studies Research Report Series, Report 98.27, University of Leeds, 1998.
21. D. Gambetta, Can We Trust Trust?, in Trust: Making and Breaking Cooperative Relations, D. Gambetta (ed.), Basil Black-well, Oxford, England, pp. 213-237, 1988.
22. L. Mui, M. Mohtashemi, and A. Halberstadt, A Computational Model of Trust and Reputation, In Proc. of the 35th Intl. Conf. on System Science, pp. 280-287, 2002.
23. A. Jøsang, Trust-Based Decision Making for Electronic Transactions, In the 4th Nordic Workshop on Secure IT Systems (NORDSEC'99), 1999.
24. A. Josang, R. Ismail, and C. Boyd, A Survey of Trust and Reputation Systems for Online Service Provision, Decision Support Systems, vol. 43, Emerging Issues in Collaborative Commerce, pp. 618-644, 2007.
25. W. Viriyasitavat, and A. Martin, In the Relation of Workflow and Trust Characteristics, and Requirements in Service Workflows, In Abd Manaf, Azizah, Akram Zeki, Mazdak Zamani, Suriayati Chuprat and Eyas El−Qawasmeh, editors, Informatics Engineering and Information Science, vol. 251 of Communications in Computer and Information Science, pp. 492−506, Springer Berlin Heidelberg. 2011.
26. M. I. Smirnov, Trust by Workflow in Autonomic Communication, Computer Network Security, pp. 136-150, 2005.
27. Z.-M. Qiu and Y. S. Wong, Dynamic Workflow Change in PDM Systems, Computers in Industry, vol. 58, pp. 453-463, 2007.
28. S. D. Ramchurn, D. Huynh, and N. R. Jennings, Trust in Multi-Agent Systems, The Knowledge Engineering Review, 2004.
29. R. Botha and J. Eloff, A Security Interpretation of the Workflow Reference Model, In Proc. of the Intl. Secur. from Small Syst. to Management of Secure Infrastr., pp 43-51, 1998.

30. E. Bertino, E. Ferrari, and V. Atluri, A Flexible Model Supporting the Specification and Enforcement of Role-Based Authorization in Workflow Management Systems, In Proc. of the 2nd ACM Workshop on Role-Based Access Control, pp. 1-12, 1997.
31. H. Koshutanski, A Survey on Distributed Access Control Systems for Web Business Processes, Intl. Journal of Network Security, vol. 9, pp. 61-69, 2009.
32. G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Gunthor, and M. Kamath, EXotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management, In Proc. of the IFIP WG8.1 Working Conf. on Information Systems for Decentralized Organizations, 1995.
33. S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah, ORBWork: A Reliable Distributed CORBA-Based Workflow Enactment System for METEOR2, Technical Report UGA-CS-TR-97-001, University of Georgia, 1997.
34. P. Muth, D. Wodtke, and J. Weissenfels, From Centralized Workflow Specification to Distributed Workflow Execution, Journal of Intelligent Information Systems, vol. 10, 1998.
35. D. Wodtke and G. Weikum, A Formal Foundation For Distributed Workflow Execution Based on State Charts, In Proc. Intl. Conf. on Database Theory, 1997.
36. Y. L. Lederer-Antonucci, P. S. Greenberg, M. zur Muehlen, and G. Ralph, Establishing Trust in a Business- to-Business Collaboration: Results from an Intl. Simulation, In Proc. of the IRMA 2003 Conf., 2003.
37. X. Ao and N. H. Minsky, Flexible Regulation of Distributed Coalitions, In Proc. of the 8th European Symposium on Research in Computer Security, pp. 39-60, 2003.
38. A. Jøsang and S. Pope, Semantic Constraints for Trust Transitivity. In Proc. of Asia-Pacific Conf. of Conceptual Modelling (APCCM), 2005.
39. R. Guha, R. Kumar, P. Raghavan, and A. Tomkins, Propagation of Trust and Distrust, In Proc. of the 13th Intl. Conf. on World Wide Web (ed), pp. 403-412, 2004.
40. Z. Cai-Nicolas and L. Georg, Propagation Models for Trust and Distrust in Social Networks, Information Systems Frontiers, vol. 7, pp. 337-358, 2005.
41. A. Jøsang, S. Marsh, and S. Pope, Exploring Different Types of Trust Propagation, In Proc. of the 4th Intl. Conf. on Trust Management (iTrust), 2006.
42. A. Jsang, E. Gray, and M. Kinateder, Analysing Topologies of Transitive Trust, In Proc. of the Workshop of Formal Aspects of Security and Trust (FAST), 2003.
43. J. Huang and M. S. Fox, An Ontology of Trust: Formal Semantics and Transitivity, In Proc. of the 8th Intl. Conf. on Electronic commerce, pp. 259-270, ACM Press, 2006.
44. M. L. Bote-Lorenzo, Y. A. Dimitriadis, and E. Gómez-Sánchez, Grid Characteristics and Uses: A Grid Definition, Across Grids 2003, LNCS 2970, pp. 291-298, 2004.
45. H. Chivers and J. McDermid, Refactoring Service-Based Systems: How to Avoid Trusting a Workflow Service: Research Articles, Workflow Service, Concurrency and Computation: Practice and Experience this Issue, vol. 18, no. 10, pp. 1255-1275, 2006.

46. M. Adamski, A. Arenas, and A. Bilas, Trust and Security in Grids: A State of the Art, CoreGRID WhitePaper Number WHP-0001, 2008.
47. A. E. Arenas, Survey Material on Trust and Security in Grids, CoreGRID Project Deliverable D.IA.03, 2005.
48. I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, A Security Architecture for Computational Grids, In Proc. of the 5th ACM Conf. on Computer and Communications Security, pp. 83-92, 1998.
49. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, A Community Authorization Service for Group Collaboration, IEEE the 3rd Intl. Workshop on Policies for Distributed Systems and Networks, 2002.
50. M. Lorch, D. Adams, D. Kafura, M. Koneni, A. Rathi, and S. Shah, The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments, In the 4th Int. Workshop on Grid Computing, 2003.
51. J. Basney, W. Nejdl, D. Olmedilla, V. Welch, and M. Winslett, Negotiating Trust on the Grid, In the 2nd WWW Workshop on Semantics in P2P and Grid Computing, 2004.
52. I. Constandache, D. Olmedilla, and W. Nejdl, Policy-Based Dynamic Negotiation for Grid Services Authorization, Technical report, L3S Research Center, 2005.
53. Globus Security Policy and Implementation. 1997, http://www.globus.org/security/.
54. I. Foster and C. Kessehnan, The Globus Project: A Progress Report, In Heterogeneous Computing Workshop, 1998.
55. L. Tie-Yan, Z. HuaFei, and L. Kwok-Yan, A Novel Two-Level Trust Model for Grid, In Proc. of the Information and Communications Security, pp. 214-225, 2003.
56. S. Langella, S. Oster, S. Hastings, F. Siebenlist, T. Kurc, and J. Saltz, Enabling the Provisioning and Management of a Federated Grid Trust Fabric, In Proc. of the 6th Annual PKI R&D Workshop, 2007.
57. S. Naqvi. P. Massonet, and A. Arenas, A Study of Languages for Specification of Grid Security Policies, CoreGRID Technical Report TR0037, 2006.
58. S. Naqvi, A.E. Arenas, and P. Massonet, Deriving Policies from Grid Security Requirements Model, In Proc. of the 2nd CoreGRID Integration Workshop, 2006.
59. S. Naqvi, C. Ponsard, P. Massonet, and A.E. Arenas, Security Requirements Elaborations for Grid Data Management Systems, Intl. Journal of System of Systems Engineering, 2009.
60. F. Azzedin and M. Maheswaran, Integrating Trust into Grid Resource Management Systems, In 2002 Intl. Conf. Parallel Processing (ICPP '02), pp. 47-54, 2002.
61. A. A. Sedrakian, R. M. Badia, J. M. Pérez, R. Sirvent, T. Kielmann, and A. Merzky, Reliability and Trust Based Workflows' Job Mapping on the Grid, Institute on Grid Systems, Tools and Environments, CoreGRID Technical Report, TR-0069, 2007.
62. S. Song and K. Hwang, Dynamic Grid Security with Trust Integration and Optimized Resource Allocation, Internet and Grid Computing Laboratory, University of Southern California, 2004.

63. S. Song and K. Hwang, Fuzzy Trust Integration for Security Enforcement in Grid Computing, In Proc. of the Intl. Symp., on Network and Parallel Computing (NPC 2004), LNCS 3222, pp. 9-21, 2005.
64. S. Song, K. Hwang, and Y.-K. Kwok, Trusted Grid Computing with Security Binding and Trust Integration, Journal of Grid Computing, vol. 3, pp. 53-73, 2005.
65. L. M. Camarinha-Matos and H. Afsarmanesh, A Roadmap for Strategic Research on Virtual Organizations, In Proc. of IFIP Working Conf. on Virtual Enterprises - PRO-VE'03, pp. 33-46, 2003.
66. J.L. Griffin, T. Jaeger, R. Perez, R. Sailer, L. Van Doorn, and R. Caceres, Trusted Virtual Domains: Toward Secure Distributed Services, In Proc. of the 1st Workshop on Hot Topics in System Dependability, 2005.
67. T. Dimitrakos, M. Wilson, and S. Ristol, TrustCoM – A Trust and Contract Management Framework Enabling Secure Collaborations in Dynamic Virtual Organizations, ERCIM News 59, 2004.
68. A. Svirskas, M.D. Wilson, A.E. Arenas, E.C Lupu, P. Giambiagi, N. Tuptuk, D. Chadwick, T. Dimitrakos, and B.Roberts, Aspects of Trusted and Secure Business-Oriented VO Management in Service Oriented Architectures, In the 1st IEEE Workshop on Service Oriented Solutions for Cooperative Organizations Munich, 2005.
69. Trusted Computing Group. www.trustedcomputinggroup.org.
70. Z. Song, S. Kotani, and R. Masuoka, Building an Independent Integrated Authentication Service, In Proc. of the 9th Intl. Conf. For Young Computer Scientists, ICYCS, IEEE Computer Society, 2008.
71. P. -W. Yau, A. Tomlinson, S. Balfe, and E. Gallery, Securing Grid Workflows with Trusted Computing, In M. Bubak,G. D. van Albada, J. Dongarra, P. M. A. Sloot (Eds.), ICCS (3), vol. 5103 of Lecture Notes in Computer Science, Springer, pp. 510-519, 2008.
72. P. -W Yau and A. Tomlinson, Enhancing Grid Security using Workflows, Trusted Computing and Virtualisation, In Proc. of the 2009 Intl. Conf. on Grid Computing and Applications (GCA 2009), Worldcomp '09, pp.113-119, 2009.
73. N. Kuntze, A. Schmidt, Z. Velikova, and C. Rudolph, Trust in Business Processes, In Proc. of the 9th Intl. Conf. For Young Computer Scientists, ICYCS, IEEE Computer Society, 2008.
74. A. Alkassar and R. Husseiki, D3.9: Study on the Impact of Trusted Computing on Identity and Identity Management, FIDIS Deliverable D3.9, 2008.
75. M. Alawneh and I. M. Abbadi, Sharing but Protecting Content Against Internal Leakage for Organisations, In Proc. of the 22nd annual IFIP WG 11.3 Working Conf. on Data and Applications Security, 2008.
76. S. Marsh, Formalising Trust as a Computational Concept, PhD thesis, University of Stirling, 1994.
77. M. Carbone, M. Nielsen, and V. Sassone, A Formal Model for Trust in Dynamic Networks, In Proc. of the 1st Intl. Conf. on Software Engineering and Formal Methods, pp. 54-63, 2003.
78. H. Davulcu, M. Kifer, L. Pokorny, C. R. Ramakrishnan, I. V. Ramakrishnan, and S. Dawson, Modeling and Analysis of Interactions in Virtual Enterprises, RIDE 1999.

79. T. Dimitrakos, A Service-Oriented Trust Management Framework, In Proc. of the Intl. Workshop on Trust in Agent Societies, 2003.
80. A. E. Arenas, I. Djordjevic, T. Dimitrakos, L. Titkov, J. Claessens, C. Geuer-Pollman, E.C. Lupu, N.Tuptuk, S. Wesner, and L. Schubert, Towards Web Services Profiles for Trust and Security in Virtual Organisations, In Proc. of the IFIP Working Conf. on Virtual Enterprises, 2005.
81. S. Galizia, A. Gugliotta, and J. Domingue, A trust based methodology for web service selection, In Proc. of the Intl. Conf. on Semantic Computing, pp. 193–200, 2007.
82. E. M. Maximilien and M.P. Singh, Toward Autonomic Web Services Trust and Selection, In Proc. of the Intl. Conf. on Service Oriented Computing, 2004.
83. Y. Kim and K. Doh, Trust Type based Semantic Web Services Assessment and Selection, In Proc. of the ICACT, IEEE Computer, pp. 2048-2053, 2008.
84. S. J. H. Yang, J. S. F. Hsieh, B. C. W. Lan, and J.-Y. Chung, Composition and Evaluation of Trustworthy Web Services, In Proc. of the BSN '05, pp. 5, IEEE Press, 2005.
85. W. Liu, Trustworthy Service Selection and Composition - Reducing the Entropy of Service-Oriented Web, In the 3rd IEEE Intl. Conf. on Industrial Informatics (INDIN), 2005.
86. S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels,P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, A. Nadalin, N. Nagaratnam, H. P.M. Nottingham, C. von Riegen, J. Schlimmer, C. Sharp, and J. Shewchuk, Web Services Policy Framework (WS-Policy), 2006.
87. S. Anderson, J. Bohren, and T. Boubez, Web Services Trust Language (WS-Trust), available at http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf., 2004.
88. L. Moreau, S. Chapman, A. Schreiber, R. Hempel, O. Rana, L. Varga, U. Cortes, and S. Willmott, Provenance-Based Trust for Grid Computing, Electronics and Computer Science, University of Southampton, Position paper, 2004.
89. C. A. Lynch, When Documents Deceive: Trust and Provenance as New Factors for Information Retrieval in a Tangled Web, In Journal of the American Society for Information Science and Technology, vol. 52, pp. 12-17, 2001.
90. L. Ding, P. Kolari, T. Finin, and A. Joshi, On Homeland Security and the Semantic Web: A Provenance and Trust Aware Inference Framework, In Proc. of the AAAI Spring Symposium on AI Technologies for Homeland Security, 2005.
91. J. Zhao, C. Wroe, C. Goble, R. Stevens, D. Quan, and M. Greenwood, Using Semantic Web Technologies for Representing E-Science Provenance, In Proc. of the 8th annual IEEE Intl. Symposium on Wearable Computers, pp. 92–106, 2004.
92. I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao, Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation, In Proc. Of the 14th Intl. Conf. on Scientific and Statistical Database Management, pp. 37-46, 2002.
93. R. D. Stevens, A. J. Robinson, and C.A. Goble, Mygrid: Personalized Bioinformatics on the Information Grid, Bioinformatics 19, pp. i302-i304, 2003.

94. C. Pancerella, J. Hewson, W. Koegler, D. Leahy, M. Lee, L. Rahn, C. Yang, J. D. Myers, B. Didier, R. McCoy, K. Schuchardt, E. Stephan, T. Windus, K. Amin, S. Bittner, C. Lansing, M. Minkoff, S. Nijsure, G. v. Laszewski, R. Pinzon, B. Ruscic, Al Wagner, B. Wang, W. Pitz, Y. L. Ho, D. Montoya, L. Xu, T. C. Allison, W. H. Green, and Jr. M. Frenklach, Metadata in the Collaboratory for Multi-Scale Chemical Science, In Dublin Core Conf., 2003.
95. J. Myers, C. Pancerella, C. Lansing, K. Schuchardt, and B. Didier, Multi-Scale Science, Supporting Emerging Practice with Semantically Derived Provenance, In Proc. of the ISWC workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, 2003.
96. J. Widom, Trio: A System for Integrated Management of Data, Accuracy, and Lineage, In Conf. on Innovative Data Systems Research, 2005.
97. S. Rajbhandari , A. Contes , O. F. Rana , V. Deora , and I. Wootten, Trust Assessment Using Provenance in Service Oriented Applications, In Proc. of the 10th IEEE on Intl. Enterprise Distributed Object Computing Conf. Workshops, pp. 65, 2006.
98. S. Rajbhandari , I. Wootten , A. Shaikh, and O. F. Rana, Evaluating Provenance-based Trust for Scientific Workflows, In Proc. of the 6th IEEE Intl. Symposium on Cluster Computing and the Grid, pp.365-372, 2006.
99. S. Rajbhandari, A. Contes, O. F.Rana, V. Deora, and I. Wootten, Establishing Workflow Trust Using Provenance Information, In the 1st IEEE Intl. Workshop on Modeling Autonomic Communications Environments, 2006.
100. A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang, Scientific Workflow Provenance Querying with Security Views, Web-Age Information Management, In Proc. of the 9th Intl. Conf. on, pp. 349–356, 2008.
101. J. Lyle and A. Martin, Trusted Computing and Provenance: Better Together, In Proc. of the 2nd Workshop on the Theory and Practice of Provenance, Usenix, 2010.
102. S. M. da Cruz, P. M. Barros, Bisch, M. L. Campos, and M. Mattoso, Provenance Services for Distributed Workflows, In Proc. of the 8th IEEE Intl. Symposium on Cluster Computing and the Grid, 2008.
103. D. Cvrcek, Mandatory Access Control in Workflow Systems, In Proc. of the JCKBSE Conf., pp. 247-254, 2000.
104. J. B. D. Joshi, W. G. Aref, A. Ghafoor, and E. H. Spafford, Security Models for Web-Based Applications, Communications of the ACM, vol. 44, pp. 38-72, 2001.
105. J. Wainer, A. Kumar, and P. Barthelmess, WRBAC- A Workflow Security Model Incorporating Controlled Overriding of Constraints, Int. J. Coop. Inf. Syst., vol 12, pp. 455-486, 2003.
106. L. He and N. Zhang, An Exposition of Authentication Services in Mobile Networks, In the 2nd Annual Post-Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting, Liverpool, pp.125-130, 2001.
107. S. Chaari, F. Biennier, C. B. Amar, and J. Favrel, An Authorization and Access Control Model for Workflow, In Intl. Symposium on Control, Communications and Signal Processing, pp. 141-148, 2004.

108. N. Nobelis, K. Boudaoud, M. Riveill, and P. Dubois, Decentralized Access Right Management for Workflow Applications, In Proc. of the 12th HP Open view University Association Plenary Workshop (HPOVUA), 2005.

109. X. Wei, W. Jun, L. Yu, and L. Jing, SOWAC: A Service-Oriented Workflow Access Control Model, In Proc. of the 28th Annual Intl. Computer Security and Applications Conf., pp. 128-134, 2004.

110. L. Wang and X. Xie, Authorization Model for Multilevel Security Workflow based on RTBAC, In Proc. of the 3rd Intl. Conf. on Anti-Counterfeiting, security, and identification in communication, pp.520-523, 2009.

111. I. Moodahi, Eh. Gudes, O. Lavee, A. Meisels, A Secure Workflow Model Based on Distributed Constrained Role and Task Assignment for the Internet, Information and Communications Security, pp. 171-186, 2004.

112. T. Ahmed and R. Tripathi, Security Policies in Distributed CSCW and Workflow Systems, Technical Report, University of Minnesota, Minneapolis, MN, 2002.

113. D. Huynh, N. R. Jennings, and N. R. Shadbolt, FIRE: An Integrated Trust and Reputation Model for Open Multi-Agent Systems, In Proc. of the 16th European Conf. on Artificial Intelligence, 2004.

114. V. Atluri and W. Huang, An Authorization Model for Workflows, In Proc. of the 4th European Symposium on Research in Computer Security, pp. 25-27, 1996.

115. J. Wainer, A. Kumar, and P. Barthelmess. Dw-rbac: A Formal Security Model of Delegation and Revocation in Workflow Systems, Inf. Syst., vol. 32, pp. 365-384, 2007.

116. L. Kagal, T. Finin, and Y. Peng, A Delegation-Based Model for Distributed Trust, In Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents, Intl. Joint Conf. on Artificial Intelligence, 2001.

117. Z. Xie and C. Chi, Quantifying Trust through Delegation in Service Oriented Architecture, Services, IEEE Congress on, pp. 308-315, 2007.

118. W. Viriyasitavat, Modeling Delegation in Requirements-Driven Trust Framework, Services, IEEE Congress on, pp. 522-529, Congress on Services - I, 2009.

119. M. Altunay, D. Brown, G. Byrd, and R. Dean, Trust-Based Secure Workflow Path Construction, In Proc. of Intl. Conf. on Service Oriented Computing, 2005.

120. M. Blaze, J. Feigenbaum, and J. Lacy, Managing Trust in Medical Information Systems, AT&T, 1996.

121. J. J. Ordille, When agents roam, who can you trust?, In the 1st Annual Conf. on Emerging Technologies and Applications in Communications, 1996.

122. U. G. Wilhelm, S. Staamann, and L. Buttyan, On the Problem of Trust in Mobile Agent Systems, In IEEE Symposium on Network And Distributed System Security, 1998.

123. N. Kuntze and J. Sch, Securing Decentralized Workflows in Ambient Environments, Embedded and Ubiquitous Computing, IEEE/IFIP Intl. Conf. on, pp. 361-366, 2008.

124. R. Falcone, G. Pezzulo, and C. Castelfranchi, A Fuzzy Approach to a Belief-Based Trust Computation, In Lecture Notes on Artificial Intelligence, pp. 73-86, 2003.

125. H. Koshutanski, F. Martinelli, P. Mori, and A. Vaccarelli, Fine-Grained and History-Based Access Control with Trust Management for Autonomic Grid Services. In ICAS, pp. 34-44, 2006.

126. D. Hollingsworth, Workflow Management Coalition: The Workflow Reference Model, WFMC Document WFMCTC00-1003, 1994.

127. Workflow Management Coalition, Workflow Management Coalition Specification: Terminology & Glossary, WFMC document WFMC-TC-1011, 1999.

128. F. Puhlmann and M. Weske, Using the Pi-Calculus for Formalizing Workflow Patterns, In van der Aalst, W., Benatallah, B., Casati, F., eds. In Proc. of the 3rd Intl. Conf. on Business Process Management, vol. 3649, LNC, Springer-Verlag, pp. 153-168, 2005.

129. K. Salimifard and M. Wright, Petri-Net-Based Modeling of Workflow Systems: An Overview. European Journal of Operational Research, vol. 134, pp. 218-230, 2001.

130. R. Eshuis and R. Wieringa, Comparing Petri Net and Activity Diagram Variants for Workflow Modeling -A Quest for Reactive Petri Nets, Petri Net Technology for Comm. Based Systems, H. Ehrig,W. Reisig, G. Rozenberg, and H. Weber, eds., 2003.

131. R. Hamadi and B. Benatallah, A Petri Net-Based Model for Web Service Composition. In Proc. of the 14th Australasian Database Conf., K. Schewe and X. Zhou, Eds. ACM Intl. Conf. Proc. Series, vol. 143. Australian Computer Society, pp. 191-200, 2003.

132. A. Jøsang, The Right Type of Trust for Distributed Systems, In ACM New Security Paradigms Workshop, 1996.

133. Y. Tang, L. Chen, K-T He, N. Jing, SRN: An Extended Petri-Net-Based Workflow Model for Web Service Composition, IEEE Intl. Conf. Web Services, pp. 591-599, 2004.

134. S. Jajodia, P. Samarati, and V. S. Subrahmanian, A Logical Language for Expressing Authorizations, In SympSecPr, Research in Security and Privacy, 1997.

135. A. A-Rahman, and S. Hailes, Supporting Trust in Virtual Communities. In Proc. of the 33rd Hawaii International Conference on System Sciences, 2000.

136. P.V. Rangan, An Axiomatic Basis of Trust in Distributed Systems, In Symposium on Security and Privacy, IEEE Computer Society Press, 1988.

137. A.J. Jones, and B.S. Firozabadi, On the Characterisation of a Trusting Agent – Aspects of a Formal Approach, In Proc. of the Formal models for Electronic Commerce (FMEC) workshop, 1999.

138. B.S. Firozabadi and M. Sergot, Power and Permissions in Security Systems. In B. Christianson, B. Crispo, J. Malcolm, and M. Roe (Eds.), In Proc. of 7th Intl. Workshop on Security Protocols, pp. 48-59, 1999.

139. Y. H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss, REFEREE: Trust Management for Web Applications, AT&T Research Labs, 1997.

140. M. Blaze M, J. Feigenbaum, and J. Lacy, Decentralized Trust Management, In Proc. of IEEE Conf. on Security and Privacy, 1996.

141. M. Blaze M, J. Feigenbaum, and A. D. Keromytis, KeyNote: Trust Management for Public-Key Infrastructures, In Security Protocols Intl. Workshop, 1998.

142. M. Blaze, Using the KeyNote Trust Management System, AT&T Research Labs, 1999.

143. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis, RFC2704 - The KeyNote Trust Management System (version 2), 1999.

144. W.H. Winsborough, K.E. Seamons, and V.E. Jones, Negotiating Disclosure of Sensitive Credentials, In 2nd Conf. on Security in Communication Networks, 1999.

145. G. Governatori and Z. Milosevic, Dealing with Contract Violations: Formalism and Domain Specific Language, In EDOC 2005, pp. 46-57. IEEE Press, 2005.

146. W.M.P. van der Aalst and K.M. van Hee, Workflow Management: Models, Methods, and Systems. MIT press, 2002.

147. W.M.P. van der Aalst and A.H.M. ter Hofstede, Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages, In K. Jensen, editor, In Proc. of the 4th Workshop on the Practical Use of Coloured Petri Netsand CPN Tools (CPN 2002), vol. 560 of DAIMI, pp. 1-20, 2002.

148. W.M.P. van der Aalst, Petri-Net-Based Workflow Management Software, In Proc. of the NFS Workshop on Workflow and Process Automation in information Systems, pp. 114-118, 1996.

149. W.M.P. van der Aalst, Chapter 10: Three Good reasons for Using a Petri-Net-Based Workflow Management System, In T.Wakayama, S. Kannapan, C.M. Khoong, S. Navathe, and J. Yates, editors, Information and Process Integration in Enterprises: Rethinking Documents, vol. 428 of The Kluwer Intl. Series in Engineering and Computer Science, pp. 161-182, 1998.

150. W.M.P. van der Aalst, The Application of Petri Nets to Workflow Management, The Journal of Circuits, Systems and Computers, vol. 8, pp. 21-66, 1998.

151. K. Klai and S. Tata, Abstraction-Based Workflow Cooperation using Petri Net Theory, In Proc. of the 14th IEEE Intl. Workshops on Enabling Technologies: infrastructure For Collaborative Enterprise, 2005.

152. W.M.P. van der Aals, K.M. van Hee, and G.J. Houben, Modelling and Analyzing Workflow using a Petri-Net Based Approach, In Proc. of the 2nd Workshop on Computer-Supported Cooperative Work, Petri Nets and Related Formalisms, pp. 31-50, 1994.

153. E. Best, R. Devillers, and M. Koutny, Petri Net Algebra, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 2001.

154. E. A. Emerson, Temporal and Modal Logic, In J. van Leeuwen, ed., Handbook of Theoretical Computer Science, Elsevier, pp. 997-1072, 1990.

155. H. H. Bi and J. L. Zhao, Applying Propositional Logic to Workflow Verification, Information Tech. Management, vol. 5, pp. 293–318F, Leymann and D. Roller Production Workflow - Concepts and Techniques, Prentice Hall, 2000.

156. U. Maurer, Modelling a Public-Key infrastructure, Lecture Notes in Computer Science, Springer-Verlag, vol. 1146, pp. 325-350, 1996.

157. Y. Ren and A. Boukerche, Modeling and Managing the Trust for Wireless and Mobile Ad-Hoc Networks, In IEEE Intl. Conf. on Communications, pp. 2129-2133, 2008.

158. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, Eigenrep: Reputation Management in P2P Networks, In World-Wide Web Conf., 2003.

159. R. Sherwood, S. Lee, and B. Bhattacharjee, Cooperative Peer Groups in Nice, IEEE INFOCOM, vol. 2, pp. 1272-1282, 2006.

160. M. Richardson, R. Agrawal, and P. Domingos, Trust Management for the Semantic Web, In the 2nd Intl. Semantic Web Conf., 2003.

161. N. Bogunovic and E. Pek, Verification of Mutual Exclusion Algorithms with SMV System, In Proc. of IEEE Region8 Eurocon 2003: Computer as a Tool, B. Zajc and M. Tkalcic, Eds., vol. 2, pp. 12-25, IEEE, 2003.

162. P. Clements, A Survey of Architecture Description Language, In the 8th Intl. Workshop on Software Specification and Design, pp. 16-28, 1996.

163. M. Huth, M. Ryan, Logic in Computer Science, the 2nd Ed, CUP, 2004.

164. E. M. Clarke and B-H Schlingloff, Model Checking, Handbook of Automated Reasoning, pp. 1635-1790, 2001.

165. L. D. Xu, W. Viriyasitavat, P. Ruchikachorn, and A. Martin, Using Propositional Logic for Requirements Verification of Service Workflow, IEEE Transactions on Industrial Informatics, vol.8, no.3, pp.639-646, 2012.

166. W. Viriyasitavat and A. Martin, An Improvement of Requirement-Based Compliance Checking Algorithm in Service Workflows, International Conference on Advances in Information Technology (AIT), UACEE, 2012.

167. E. M. Clarke and E. A. Emerson, Synthesis of Synchronization Skeletons for Branching Time Temporal Logic. In Logic of Programs: Workshop, vol. 131 of LNC Springer-Verlag, 1981.

168. E. M. Clarke and I. A. Draghicescu, Expressibility Results for Linear Time and Branching Time Logic, In Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, vol. 354, LNCS, Springer, pp 428-437, 1988.

169. J. R. Burch, E. M. Clarke, K. L. McMillan, D. Dill, and L. J. Hwang, Symbolic Model Checking: 10^{20} States and Beyond, Information and computation, vol. 98, no. 2, pp. 142-170, 1992, also in 5th IEEE LICS 1990.

170. K. L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.

171. E. M. Clarke, O. Grumberg, and D. E. Long, Model Checking and Abstraction, In Proc. of the 19th Annual ACM Symposium on Principles of Programming Languages, 1992.

172. D. L. Long, Model Checking, Abstraction, and Compositional Reasoning, PhD Thesis, Carnegie Mellon University, 1993.

173. G. Governatori, Z. Milosevic, and S. Sadiq, Compliance Checking between Business Processes and Business Contracts. In the 10th Intl. Conf. of Enterprise Distributed Object Computing, IEEE Press, pp. 221-232, 2006.

174. Y. Lui, S. Mller, and K. Xu, A Static Compliance-Checking Framework for Business Process Models, IBM System Journal, vol. 46, no. 2, pp. 335–362, 2007.

175. S. Kerrigan and K. H. Law, Logic-Based Regulation Compliance-Assistance. In the 9th Intl. Conf. on Artificial Intelligence and Law, pp. 126-135, 2003.

176. G. Stalmarck and M. Saflund, Modelling and Verifying Systems and Software in Propositional Logic, In B. K. Daniels, editor, Safety of Computer Control Systems (SAFECOMP'90), pp. 31-36, Pergamon Press, 1990.

177. D. Jackson, Alloy: A Lightweight Object Modelling Notation, Technical Report 797, MIT Laboratory for Computer Science, 2000.

178. A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, NuSMV: A New Symbolic Model Verifier, Intl. Journal on Software Tools for Technology Transfer, vol. 2, pp. 2000, 2000.

179. Y. Chen and Y. Chen, An Efficient Algorithm for Answering Graph Reachability Queries, In Proc. of the 2008 IEEE 24th Intl. Conf. on Data Engineering, IEEE Computer Society, pp. 893-902, 2008.

180. R. Jin, Y. Xiang, N. Ruan, and H. Wang, Efficiently Answering Reachability Queries on Very Large Directed Graphs, In Proc. of the 2008 ACM SIGMOD Intl. Conf. on Management of data (SIGMOD '08). ACM, pp. 595-608, 2008.

181. T. Andrew, Computation Time Comparison Between MATLAB and C++ Using Launch Windows, 2012, http://digitalcommons.calpoly.edu/aerosp/78.

182. W.M.P. van der Aalst, Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets, Systems Analysis, Modelling, and Simulation, vol. 34, no. 3, pp. 335-367, 1999.

183. V. Atluri and W-K. Huang, Enforcing Mandatory and Discretionary Security in Workflow Management Systems, Journal of Computer Security, vol. 5, pp. 303–339, 1997.

184. R. Eshuis and R. Wieringa, A Comparison of Petri Net and Activity Diagram Variants, In Proc. of the 2nd Intl. Conf. on Petri Net Technologies for Modelling Communication Based Systems, pp. 93-104, 2001.

185. R. Eshuis and R. Wieringa, A Real-Time Execution Semantics for UML Activity Diagrams, In Proc. of the 4th Intl. Conf. on Fundamental Approaches to Software Engineering (FASE2001) ed. H. Hussmann, Lecture Notes in Computer Science, vol. 2029, pp. 76-90, Springer, 2001.

186. W.M.P. van der Aalst, Workflow Verification: Finding Control-Flow Errors using Petri-Net-Based Techniques, In W. van der Aalst, J. Desel, and A. Oberweis, editors, Business Process Management, Springer, 2000.

187. W.M.P. van der Aalst, Pi calculus versus Petri Nets: Let us Eat "Humble Pie" rather than Further Inflate the "Pi Hype", 2004.

188. Web Services Business Process Execution Language Version 2.0 – Committe Specification, Technical Report, OASIS, 2007.

189. C. Ouyang, W.M.P van der Aalst, S. Breutel, M. Dumas, A.H.M ter Hofstede, and H.M.W Verbeek, Formal Semantics and Analysis of Control Flow in WS-BPEL, Technical Report BPM-05-13, BPM Center Report, 2005.

190. http://www.w3.org/2002/04/17-ws-usecase

191. http://www.w3.org/2002/06/ws-example

192. H. Li, H. Wang, Z. Shang, and Y. Dong, Virtual travel agency based on web services, In Proc. of CSCWD, pp.445-451, 2008.

Appendix A: Theorem Proofs for TS Properties

Assume that Δ is an abbreviation of $\mathcal{F}E_t, \mathcal{P}E_t, \mathcal{F}A_t$, or $\mathcal{P}A_t$, Δ' is of $\mathcal{F}$ or $\mathcal{P}$, ∇ is of $\mathcal{H}$ or $\mathcal{B}$, and β is a workflow formula.

Theorem 4.1: The equivalence $\Delta' A_t(\phi \sqcup \omega) \equiv \Delta' A_t(\phi \sqcap \omega) \,||\, \Delta' A_t \phi \,||\, \Delta' A_t \omega$ holds for all Composite formulas ϕ and ω.

Proof: Suppose $S_1, \ldots, S_n$ are service connected to a task t. ∇ is omitted as directions have nothing to do with this proof. One common way to prove the equivalence by proving that $\Delta' A_t(\phi \sqcup \omega) \leftrightarrow \Delta' A_t(\phi \sqcap \omega) \,||\, \Delta' A_t \phi \,||\, \Delta' A_t \omega$ is tautology for each type of workflow connectives. we prove by constructing a truth table in the following conditions: (1) p_1 represents 'some $S_1 \ldots S_n$ satisfy ϕ', such that $!\,p_1$ represents 'no $S_1 \ldots S_n$ satisfies ϕ', (2) q_1 represents 'some $S_1 \ldots S_n$ satisfy ω', such that $!\,q_1$ represents 'no $S_1 \ldots S_n$ satisfies ω', (3) p_2 represents 'all $S_1 \ldots S_n$ satisfy ϕ', such that $!\,p_2$ represents 'some $S_1 \ldots S_n$ satisfy $!\,\phi$', (4) q_2 represents 'all $S_1 \ldots S_n$ satisfy ω', such that $!\,q_2$ represents 'some $S_1 \ldots S_n$ satisfy $!\,\omega$', and (5) s_1 represents ' some $S_1 \ldots S_l$ satisfy ϕ and $S_m \ldots S_n$ satisfies ω' where $m \leq l$. From these conditions, we can see that the relation between p_1 and p_2 can be inferred as $p_2 \rightarrow p_1$, and between q_1 and q_2 as $q_2 \rightarrow q_1$. Any row in the following truth table not complied with these conditions is highlighted below. S and US denotes *satisfied* and *unsatisfied*.

p_1	p_2	q_1	q_2	s_1	O = $\Delta' A_t(\phi \sqcup \omega)$	P = $\Delta' A_t(\phi \sqcap \omega)$	Q = $\Delta' A_t \phi$	R = $\Delta' A_t \omega$	P || Q || R	O ↔ P|| Q || R
F	F	F	F	-	US	US	US	US	US	**S**
F	F	F	T	-	-	-	-	-	-	**-**
F	F	T	F	-	US	US	US	US	US	**S**
F	F	T	T	-	S	US	US	S	S	**S**
F	T	F	F	-	-	-	-	-	-	**-**
F	T	F	T	-	-	-	-	-	-	**-**
F	T	T	F	-	-	-	-	-	-	**-**
F	T	T	T	-	-	-	-	-	-	**-**
T	F	F	F	-	US	US	US	US	US	**S**
T	F	F	T	-	-	-	-	-	-	**-**
T	F	T	F	F	US	US	US	US	US	**S**
T	F	T	F	T	S	S	US	US	S	**S**
T	F	T	T	-	S	S	US	S	S	**S**
T	T	F	F	-	S	US	S	US	S	**S**
T	T	F	T	-	-	-	-	-	-	**-**
T	T	T	F	-	S	S	S	US	S	**S**
T	T	T	T	-	S	S	S	S	S	**S**

Theorem 4.2: The equivalence $\Delta' A_t(\phi \sqcap \omega) \equiv \Delta' A_t(\phi \sqcup \omega)\ \&\ \Delta' E_t\phi\ \&\ \Delta' E_t\omega$ holds for all Composite formulas ϕ, ω and ψ.

Proof: **From theorem 4.1** $\Delta' A_t(\phi \sqcup \omega) \equiv \Delta' A_t(\phi \sqcap \omega)\ ||\ \Delta' A_t\phi\ ||\ \Delta' A_t\omega$, we have

$$\Delta' A_t(\phi \sqcap \omega) \equiv (\Delta' A_t(\phi \sqcap \omega)\ ||\ \Delta' A_t\phi\ ||\ \Delta' A_t\omega)\ \&\ \Delta' E_t\phi\ \&\ \Delta' E_t\omega$$
$$\equiv ((\Delta' A_t(\phi \sqcap \omega)\ \&\ \Delta' E_t\phi\ )\ ||\ (\Delta' A_t\phi\ \&\ \Delta' E_t\phi\ )\ ||\ (\Delta' A_t\omega\ \&\ \Delta' E_t\phi\ ))\ \&\ \Delta' E_t\omega$$

Since $\Delta' A_t(\phi \sqcap \omega) \rightarrow \Delta' E_t\phi$ and $\Delta' A_t\phi \rightarrow \Delta' E_t\phi$, such that

$$\Delta' A_t(\phi \sqcap \omega) \equiv ((\Delta' A_t(\phi \sqcap \omega)\ ||\ \Delta' A_t\phi\ \ ||\ (\Delta' A_t\omega\ \&\ \Delta' E_t\phi))\ \&\ \Delta' E_t\omega \quad (\textit{Distributive})$$
$$\Delta' A_t(\phi \sqcap \omega) \equiv (\Delta' A_t(\phi \sqcap \omega)\ \&\ \Delta' E_t\omega)\ ||\ (\Delta' A_t\phi\ \&\ \Delta' E_t\omega)\ ||\ (\Delta' A_t\omega\ \&\ \Delta' E_t\phi\ \&\ \Delta' E_t\omega)$$

Since $\Delta' A_t(\phi \sqcap \omega) \rightarrow \Delta' E_t\omega$ and $\Delta' A_t\omega \rightarrow \Delta' E_t\omega$, such that

$$\Delta' A_t(\phi \sqcap \omega) \equiv \Delta' A_t(\phi \sqcap \omega)\ ||\ (\Delta' A_t\phi\ \&\ \Delta' E_t\omega)\ ||\ (\Delta' A_t\omega\ \&\ \Delta' E_t\phi) \quad (\textit{Distributive})$$

Since $(\Delta' A_t\phi\ \&\ \Delta' E_t\omega) \rightarrow \Delta' A_t(\phi \sqcap \omega)$, we have

$$\Delta' A_t(\phi \sqcap \omega) \equiv \Delta' A_t(\phi \sqcap \omega)\ ||\ (\Delta' A_t\omega\ \&\ \Delta' E_t\phi)$$

Since $(\Delta' A_t\omega\ \&\ \Delta' E_t\phi) \rightarrow \Delta' A_t(\phi \sqcap \omega)$, we have

$$\Delta' A_t(\phi \sqcap \omega) \equiv \Delta' A_t(\phi \sqcap \omega)\ \square$$

Theorem 4.3: The equivalence

$\nabla\Delta' E_t(\phi \boxplus \omega) \equiv \nabla\big((\Delta' E_t\phi\ \&\ !\Delta' E_t\omega)\ ||\ (!\Delta' E_t\phi\ \&\ \Delta' E_t\omega)\big)$ holds for all Composite formulas ϕ and ω.

Proof: Suppose that $S_1, \ldots, S_n$ are service connected to a task t. ∇ is omitted as directions have no effect in this proof. One common way to prove the equivalence by proving that $\nabla\Delta' E_t(\phi \boxplus \omega) \leftrightarrow \nabla\big((\Delta' E_t\phi\ \&\ !\Delta' E_t\omega)\ ||\ (!\Delta' E_t\phi\ \&\ \Delta' E_t\omega)\big)$ is tautology for each type of workflow connectives. we prove by constructing a truth table in the following conditions: (1) p_1 represents 'some $S_1 \ldots S_n$ satisfy ϕ', such that $!\,p_1$ represents 'no $S_1 \ldots S_n$ satisfies ϕ', (2) q_1 represents 'for some $S_1 \ldots S_n$ satisfy ω', such that $!\,q_1$ represents 'no $S_1 \ldots S_n$ satisfies ω', (3) p_2 represents 'all $S_1 \ldots S_n$ satisfy ϕ', such that $!\,p_2$ represents 'some $S_1 \ldots S_n$ satisfy $!\,\phi$', and (4) q_2 represents 'all $S_1 \ldots S_n$ satisfy ω', such that $!\,q_2$ represents 'some $S_1 \ldots S_n$ satisfy $!\,\omega$'. From these conditions, we can see that the relation between p_1 and p_2 can be inferred as $p_2 \rightarrow p_1$, and between q_1 and q_2 as $q_2 \rightarrow q_1$. Any row in the following truth table not complied with these conditions is highlighted below. S, PS, and US denote Satisfy, Partially Satisfy, and Unsatisfy, respectively.

p_1	p_2	q_1	q_2	$O = \Delta' E_t(\phi \boxplus \omega)$	$\Delta' E_t \phi$	$!\Delta' E_t \phi$	$\Delta' E_t \omega$	$!\Delta' E_t \omega$	$P = \Delta' E_t \phi \,\&\, !\Delta' E_t \omega$	$Q = !\Delta' E_t \phi \,\&\, \Delta' E_t \omega$	P||Q	$O \leftrightarrow P\|\|Q$
F	F	F	F	US	US	S	US	S	US	US	US	**S**
F	F	F	T	-	-	-	-	-	-	-	-	**-**
F	F	T	F	S	US	S	S	US	US	S	S	**S**
F	F	T	T	S	US	S	S	US	US	S	S	**S**
F	T	F	F	-	-	-	-	-	-	-	-	**-**
F	T	F	T	-	-	-	-	-	-	-	-	**-**
F	T	T	F	-	-	-	-	-	-	-	-	**-**
F	T	T	T	-	-	-	-	-	-	-	-	**-**
T	F	F	F	S	S	US	US	S	S	US	s	**S**
T	F	F	T	-	-	-	-	-	-			**-**
T	F	T	F	US	S	US	S	US	US	US	US	**S**
T	F	T	T	US	S	US	S	US	US	US	US	**S**
T	T	F	F	S	S	US	US	S	S	US	S	**S**
T	T	F	T	-	-	-	-	-	-	-	-	**-**
T	T	T	F	US	S	US	S	US	US	US	US	**S**
T	T	T	T	US	S	US	S	US	US	US	US	**S**

Theorem 4.4: The equivalence

$\nabla\Delta' A_t(\phi \boxplus \omega) \equiv \nabla\big((\Delta' A_t \phi \,\&\, !\Delta' E_t \omega) \,||\, (!\Delta' E_t \phi \,\&\, \Delta' A_t \omega)\big)$ holds for all Composite formulas ϕ and ω.

Proof: Suppose that all variables setting and conditions are the same as defined in **theorem 4.1**. We have to prove that $\nabla\Delta' A_t(\phi \boxplus \omega) \equiv \nabla\big((\Delta' A_t \phi \,\&\, !\Delta' E_t \omega) \,||\, (!\Delta' E_t \phi \,\&\, \Delta' A_t \omega)\big)$ is tautology for each type of workflow connectives.

p_1	p_2	q_1	q_2	$O = \Delta' A_t(\phi \boxplus \omega)$	$\Delta' A_t \phi$	$!\Delta' E_t \phi$	$\Delta' A_t \omega$	$!\Delta' E_t \omega$	$P = \Delta' A_t \phi \,\&\, !\Delta' E_t \omega$	$Q = !\Delta' E_t \phi \,\&\, \Delta' E_t \omega$	P||Q	$O \leftrightarrow P\|\|Q$
F	F	F	F	US	US	S	US	S	US	US	US	**S**
F	F	F	T	-	-	-	-	-	-	-	-	**-**
F	F	T	F	US	US	S	US	US	US	US	US	**S**
F	F	T	T	S	US	S	S	US	US	S	S	**S**
F	T	F	F	-	-	-	-	-	-	-	-	**-**
F	T	F	T	-	-	-	-	-	-	-	-	**-**
F	T	T	F	-	-	-	-	-	-	-	-	**-**
F	T	T	T	-	-	-	-	-	-	-	-	**-**
T	F	F	F	US	US	US	US	S	US	US	US	**S**
T	F	F	T	-	-	-	-	-	-			**-**
T	F	T	F	US	US	US	US	US	US	US	US	**S**
T	F	T	T	US	US	US	S	US	US	US	US	**S**
T	T	F	F	S	S	US	US	S	S	US	S	**S**
T	T	F	T	-	-	-	-	-	-	-	-	**-**
T	T	T	F	US	S	US	US	US	US	US	US	**S**
T	T	T	T	US	S	US	S	US	US	US	US	**S**

Theorem 4.5: The equivalence $\nabla\Delta'E_t(\phi \sqcup (\omega \boxplus \psi)) \equiv \Delta'E_t\phi \,||\, (\Delta'E_t\omega \,\&\, !\Delta'E_t\psi) \,||\, (!\Delta'E_t\omega \,\&\, \Delta'E_t\psi)$ holds for all Composite formulas ϕ, ω and ψ.

Proof: This equivalence is proved by using the algebraic properties in section 4.6.1 as follows:

$\Delta'E_t(\phi \sqcup (\omega \boxplus \psi)) \equiv \Delta'E_t\phi \,||\, \Delta'E_t(\omega \boxplus \psi))$ ($\sqcup$ *Elimination*)

$\Delta'E_t(\phi \sqcup (\omega \boxplus \psi)) \equiv \Delta'E_t\phi \,||\, (\Delta'E_t\omega \,\&\, !\Delta'E_t\psi) \,||\, (!\Delta'E_t\omega \,\&\, \Delta'E_t\psi)$ ($\boxplus$ *Elimination*) □

Theorem 4.6: The equivalence

$\nabla\Delta'A_t(\phi \sqcup (\omega \boxplus \psi)) \equiv \nabla((\Delta'A_t(\phi \sqcup \omega) \,\&\, !\Delta'E_t\psi) \,||\, (\Delta'A_t(\phi \sqcup \psi) \,\&\, !\Delta'E_t\omega) \,||\, \Delta'A_t\phi))$

holds for all Composite formulas ϕ, ω and ψ.

Proof: This equivalence is proved by using the algebraic properties in section 4.6.1 as follows: (For the ease of reading, ∇ and Δ' are not presented in the proof). In what follows, we will prove that (1) = (2).

$A_t(\phi \sqcup (\omega \boxplus \psi))$

$\equiv A_t(\phi \sqcap (\omega \boxplus \psi)) \,||\, A_t\phi \,||\, A_t(\omega \boxplus \psi)$ ($\sqcup$ *Elimination*)

$\equiv A_t((\phi \sqcap \omega) \boxplus (\phi \sqcap \psi)) \,||\, A_t\phi \,||\, A_t(\omega \boxplus \psi)$ (*Distributive Property*)

$\equiv (A_t(\phi \sqcap \omega) \,\&\, !\,E_t(\phi \sqcap \psi)) \,||\, (!\,E_t(\phi \sqcap \omega) \,\&\, A_t(\phi \sqcap \psi)) \,||\, A_t\phi \,||\, A_t(\omega \boxplus \psi)$ ($\boxplus$ *Elimination*)

$\equiv (A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega \,\&\, !\,(E_t\phi \,\&\, E_t\psi)) \,||\, !\,(E_t\phi \,\&\, E_t\omega) \,\&\, A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi \,||\, A_t\phi \,||\, A_t(\omega \boxplus \psi)$ ($\sqcap$ *Elimination*)

$\equiv (A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega \,\&\, (!\,E_t\phi \,||\, !\,E_t\psi)) \,||\, (!\,E_t\phi \,||\, !\,E_t\omega) \,\&\, A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi \,||\, A_t\phi \,||\, A_t(\omega \boxplus \psi)$ (*De Morgan's Rule*)

$\equiv A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega \,\&\, !\,E_t\psi \,||\, A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi \,\&\, !\,E_t\omega \,||\, A_t\phi \,||\, A_t\omega \,\&\, !\,E_t\psi \,||\, !\,E_t\omega \,\&\, A_t\psi$ _____(1)

$A_t(\phi \sqcup (\omega \boxplus \psi))$

$\equiv A_t(\phi \sqcup \omega) \,\&\, !\,E_t\psi) \,||\, (A_t(\phi \sqcup \psi) \,\&\, !\,E_t\omega) \,||\, A_t\phi$

$\equiv (A_t(\phi \sqcap \omega) \,||\, A_t\phi \,||\, A_t\omega) \,\&\, !\,E_t\psi) \,||\, ((A_t(\phi \sqcap \psi) \,||\, A_t\phi \,||\, A_t\psi) \,\&\, !\,E_t\omega) \,||\, A_t\phi$ ($\sqcup$ *Elimination*)

$\equiv (((A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega) \,||\, A_t\phi \,||\, A_t\omega) \,\&\, !\,E_t\psi) \,||$
$((A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi) \,||\, A_t\phi \,||\, A_t\psi) \,\&\, !\,E_t\omega) \,||\, A_t\phi$ ($\sqcap$ *Elimination*)

$\equiv ((((A_t(\phi \sqcup \omega) \,||\, A_t\phi) \,\&\, (E_t\phi \,||\, A_t\phi) \,\&\, (E_t\omega \,||\, A_t\phi)) \,||\, A_t\omega) \,\&\, !\,E_t\psi) \,||$

$((((A_t(\phi \sqcup \psi) \,||\, A_t\phi) \,\&\, (E_t\phi \,||\, A_t\phi) \,\&\, (E_t\psi \,||\, A_t\phi)) \,||\, A_t\psi) \,\&\, !\,E_t\omega) \,||\, A_t\phi$

(*Distributive Property*)

Since $A_t\phi \rightarrow A_t(\phi \sqcup \omega)$, $A_t\phi \rightarrow A_t(\phi \sqcup \psi)$, and $A_t\phi \rightarrow E_t\phi$, such that

$\equiv (((A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, (E_t\omega \,||\, A_t\phi)) \,||\, A_t\omega) \,\&\, !\,E_t\psi) \,||$

$(((A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, (E_t\psi \,||\, A_t\phi)) \,||\, A_t\psi) \,\&\, !\,E_t\omega) \,||\, A_t\phi$ (*Distributive Property*)

$\equiv (((A_t(\phi \sqcup \omega) \,||\, A_t\omega) \,\&\, (E_t\phi \,||\, A_t\omega) \,\&\, (E_t\omega \,||\, A_t\phi \,||\, A_t\omega)) \,\&\, !\,E_t\psi) \,||$

$(((A_t(\phi \sqcup \psi) \,||\, A_t\psi) \,\&\, (E_t\phi \,||\, A_t\psi) \,\&\, (E_t\psi \,||\, A_t\phi \,||\, A_t\psi)) \,\&\, !\,E_t\omega) \,||\, A_t\phi$

(*Distributive Property*)

Since $A_t\omega \rightarrow A_t(\phi \sqcup \omega)$, $A_t\psi \rightarrow A_t(\phi \sqcup \psi)$, $A_t\omega \rightarrow E_t\omega$, and $A_t\psi \rightarrow E_t\psi$, such that

$\equiv ((A_t(\phi \sqcup \omega) \,\&\, (E_t\phi \,||\, A_t\omega) \,\&\, (E_t\omega \,||\, A_t\phi)) \,\&\, !\,E_t\psi) \,||$

$((A_t(\phi \sqcup \psi) \,\&\, (E_t\phi \,||\, A_t\psi) \,\&\, (E_t\psi \,||\, A_t\phi)) \,\&\, !\,E_t\omega) \,||\, A_t\phi$

$\equiv ((((A_t(\phi \sqcup \omega) \,\&\, E_t\phi) \,||\, (A_t(\phi \sqcup \omega) \,\&\, A_t\omega)) \,\&\, (E_t\omega \,||\, A_t\phi)) \,\&\, !\,E_t\psi) \,||$

$((((A_t(\phi \sqcup \psi) \,\&\, E_t\phi) \,||\, (A_t(\phi \sqcup \psi) \,\&\, A_t\psi)) \,\&\, (E_t\psi \,||\, A_t\phi)) \,\&\, !\,E_t\omega) \,||\, A_t\phi$

(*Distributive Property*)

$\equiv (((A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega) \,||\, (A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, A_t\phi) \,||$

$(A_t(\phi \sqcup \omega) \,\&\, A_t\omega \,\&\, E_t\omega) \,||\, (A_t(\phi \sqcup \omega) \,\&\, A_t\omega \,\&\, A_t\phi)) \,\&\, !\,E_t\psi\,) \,||$

$(((A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi) \,||\, (A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, A_t\phi) \,||$

$(A_t(\phi \sqcup \psi) \,\&\, A_t\psi \,\&\, E_t\psi) \,||\, (A_t(\phi \sqcup \psi) \,\&\, A_t\psi \,\&\, A_t\phi)) \,\&\, !\,E_t\omega\,) \,||\, A_t\phi$

(*Distributive Property*)

Since, $A_t\phi \rightarrow E_t\phi$, $A_t\omega \rightarrow E_t\omega$, $A_t\psi \rightarrow E_t\psi$, $(A_t(\phi \sqcup \omega) \,\&\, A_t\omega \,\&\, A_t\phi) \rightarrow (A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega)$, and $(A_t(\phi \sqcup \psi) \,\&\, A_t\psi \,\&\, A_t\phi) \rightarrow (A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi)$, we have

$\equiv ((A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega) \,||\, (A_t(\phi \sqcup \omega) \,\&\, A_t\phi) \,||\, (A_t(\phi \sqcup \omega) \,\&\, A_t\omega)) \,\&\, !\,E_t\psi)||$

$((A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi) \,||\, (A_t(\phi \sqcup \psi) \,\&\, A_t\phi) \,||\, (A_t(\phi \sqcup \psi) \,\&\, A_t\psi)) \,\&\, !\,E_t\omega) \,||\, A_t\phi$

$\equiv (A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega \,\&\, !\,E_t\psi) \,||\, (A_t(\phi \sqcup \omega) \,\&\, A_t\phi \,\&\, !\,E_t\psi) \,||\quad (A_t(\phi \sqcup$

$\omega) \,\&\, A_t\omega \,\&\, !\,E_t\psi\,)||\, (A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi \,\&\, !\,E_t\omega) \,||$

$(A_t(\phi \sqcup \psi) \,\&\, A_t\phi \,\&\, !\,E_t\omega) \,||\, (A_t(\phi \sqcup \psi) \,\&\, A_t\psi \,\&\, !\,E_t\omega) \,||\, A_t\phi$ (*Distributive Property*)

Again, since $A_t\phi \rightarrow (A_t(\phi \sqcup \omega)$, $A_t\omega \rightarrow (A_t(\phi \sqcup \omega)$, and $A_t\psi \rightarrow (A_t(\phi \sqcup \psi)$, we have

$\equiv (A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega \,\&\, !\,E_t\psi) \,||\, (A_t\phi \,\&\, !\,E_t\psi) \,||\, (A_t\omega \,\&\, !\,E_t\psi\,)||$

$(A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi \,\&\, !\,E_t\omega) \,||\, (A_t\phi \,\&\, !\,E_t\omega) \,||\, (A_t\psi \,\&\, !\,E_t\omega) \,||\, A_t\phi$

Since $A_t\phi \rightarrow (A_t\phi \,\&\, !\,E_t\psi)$ and $A_t\phi \rightarrow (A_t\phi \,\&\, !\,E_t\omega)$, the terms $(A_t\phi \,\&\, !\,E_t\psi)$ and $(A_t\phi \,\&\, !\,E_t\omega)$ are eliminate. We have

$\equiv (A_t(\phi \sqcup \omega) \,\&\, E_t\phi \,\&\, E_t\omega \,\&\, !\,E_t\psi) \,||\, (A_t\omega \,\&\, !\,E_t\psi\,)||$

$(A_t(\phi \sqcup \psi) \,\&\, E_t\phi \,\&\, E_t\psi \,\&\, !\,E_t\omega) \,||\, (A_t\psi \,\&\, !\,E_t\omega) \,||\, A_t\phi$_____(2)

Finally, we obtain (1) = (2) □

Appendix B: TS Semantics for Partially Satisfied

Def. B.1 (*Partially Satisfied*): Given *partially satisfied* from **Def. 3.5,** each TS formula ω is interpreted over an SWN $\mathcal{M}$. The definition of whether $\mathcal{M} \vDash' \omega$ holds can be understood with the condition that if $\mathcal{M}' \vDash \omega$ and $\mathcal{M}' \subseteq \mathcal{M}$, $\mathcal{M} \vDash' \omega$.

According to **Def. 3.5**, *partially satisfied* is related to the subset notation, $\mathcal{M}' \subseteq \mathcal{M}$. The following definition highlights a formal way to deduce the subsets of a workflow. Figure B.1 illustrates the examples to which the subset notation is applied.

Def. B.2 (Subsets of SWN): Given an SWN $\mathcal{M}$, the possible subset $\mathcal{M}'$ denoted by $\mathcal{M}' \subseteq \mathcal{M}$ is presented by each type of workflow connectives. The graphical examples are provided in Figure B.1. (Please note that the dot operator (.), e.g., $\mathcal{M}.\mathrm{F}$, enables access to the internal components of a model $\mathcal{M}$):

1. $\mathcal{M}' \subseteq \mathcal{M}$ iff $\mathcal{M}' = \mathcal{M}$, where $\mathcal{M}$ only consists of AND connectives in *service-join*, *task-join*, *service-split*, and *task-split*.
2. In *service-join*, if $\mathcal{M}$ consists of $\mathrm{OR}(\mathrm{p_j})$ or $\mathrm{XOR}(\mathrm{p_j})$ where $(\mathrm{p_j}, \mathrm{t}) \in \mathcal{M}.\mathrm{F}$, $\mathcal{M}' \subseteq \mathcal{M}$ iff $\{(\mathrm{p_k}, \mathrm{t}) | 0 < \mathrm{k} \leq \mathrm{j}\} \in \mathcal{M}'.\mathrm{F}$ and all elements in $\mathcal{M}'$ are reachable from start to end.
3. In *task-join*, if $\mathcal{M}$ consists of $\mathrm{OR}(\mathrm{t_j})$ or $\mathrm{XOR}(\mathrm{p_j})$ where $(\mathrm{t_j}, \mathrm{p}) \in \mathcal{M}.\mathrm{F}$, $\mathcal{M}' \subseteq \mathcal{M}$ iff $\{(\mathrm{t_k}, \mathrm{p}) | 0 < \mathrm{k} \leq \mathrm{j}\} \in \mathcal{M}'.\mathrm{F}$ and all elements in $\mathcal{M}'$ are reachable from start to end.
4. In *service-split,* if $\mathcal{M}$ consists $\mathrm{OR}(\mathrm{p_j})$ or $\mathrm{XOR}(\mathrm{p_j})$ where $(\mathrm{t}, \mathrm{p_j}) \in \mathcal{M}.\mathrm{F}$, $\mathcal{M}' \subseteq \mathcal{M}$ iff $\{(\mathrm{t}, \mathrm{p_k}) | 0 < k \leq \mathrm{j}\} \in \mathcal{M}'.\mathrm{F}$, and all elements in $\mathcal{M}'$ are reachable from start to end.
5. In *task-split,* if $\mathcal{M}$ consists $\mathrm{OR}(\mathrm{t_j})$ or $\mathrm{XOR}(\mathrm{p_j})$ where $(\mathrm{p}, \mathrm{t_j}) \in \mathcal{M}.\mathrm{F}$, $\mathcal{M}' \subseteq \mathcal{M}$ iff $\{(\mathrm{p}, \mathrm{t_k}) | 0 < \mathrm{k} \leq \mathrm{j}\} \in \mathcal{M}'.\mathrm{F}$ and all elements in $\mathcal{M}'$ are reachable from start to end.

It is important to note that branches that have experienced one type of connectives have to converge with the same type at a future, whether single or multiple, points. For example, branches that are separated from the AND must converge with AND. This can be checked by applying logic-based detection of workflow structure anomalies [174]. In this thesis, however, we do not emphasise the matter of how workflow is created, and also assume that a workflow is represented in a valid form.

Proposition B.1: Given a subset of SWN $\mathcal{M}'$, $\mathcal{M}' \subseteq \mathcal{M}$, the maximum number of $\mathcal{M}'$ is equal to the number of different markings of the formula in $\mathcal{M}.r$ which are evaluated to be true.

Proof: The most straightforward way to evaluate a truth value of a formula is to substitute services $(\mathrm{p_i})$ and tasks $(\mathrm{t_j})$ with propositional atoms $(\mathrm{a_k})$ and mark them with a value (true or false). If an atom is executed, its truth value is true; otherwise, it is false. Then, we proceed by using mathematical induction of a workflow structure formula (ψ). As noted above, because one type of connectives must converge with the same type, we only count on *service-split* and *task-split* (it can also be done on **-join*, which would yield the same result). OR split (the fourth example in Figure B.1) is illustrated in this proof where the number of different valuations that are evaluated to true is $2^{\mathrm{n}+1} - 1$, where n is the number of OR connective presented in the formula. For example, the formula $\mathrm{a_1}$ OR $\mathrm{a_2}$ OR $\mathrm{a_3}$ with

two OR connectives ($n = 2$) can be evaluated to true with $2^{2+1} - 1 = 7$ possible assignments. The subset derivation from the XOR type is identical to OR which can be argued in the same way. The proof of the AND type is straightforward and therefore the detail is omitted, as there is only one possible valuation (true for all a_k).

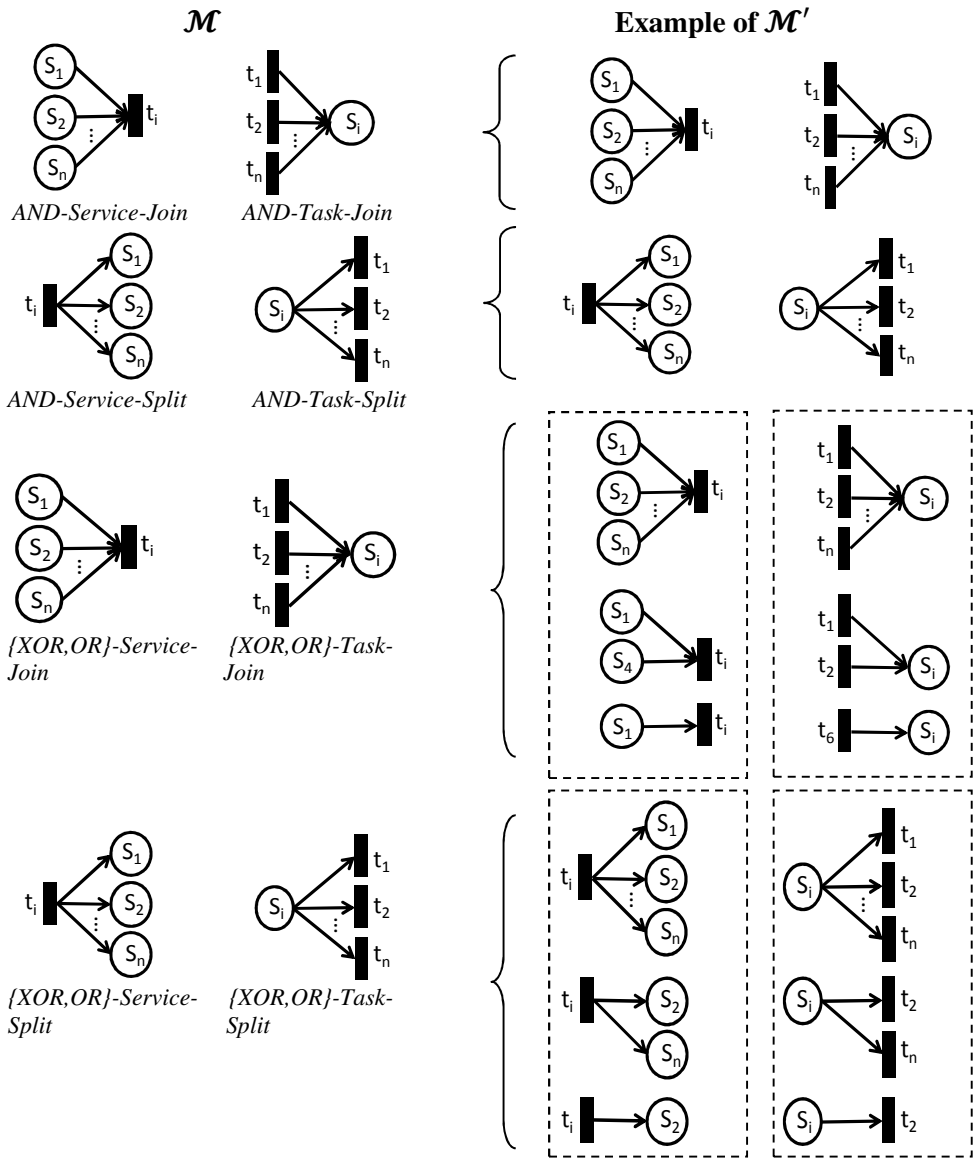


Figure B.1. Example of $\mathcal{M}' \subseteq \mathcal{M}$

Base Case: The base case contains one OR connective ($n = 1$) in the formula ψ. From **Def. 3.1,** ψ can be interpreted from $\text{range}(\mathcal{M}'.\text{F})$ that are evaluated to be true ($\mathcal{M}'.\text{F} \in \{\{(a_0, a_1)\}, \{(a_0, a_2)\}, \{(a_0, a_1), (a_0, a_2)\}\}$), i.e., $\psi = a_1 \text{ OR } a_2$. The possible truth assignments of $\psi = 2^{1+1} - 1 = 3$ (the left table below), which is identical to the number of possible subsets of $\mathcal{M}$ (the right table below).

a_1	a_2	$a_1\ OR\ a_2$
F	F	F
T	F	T
F	T	T
T	T	T

$range(\mathcal{M}'.F)$
$\{\}$
$\{a_1\}$
$\{a_2\}$
$\{a_1, a_2\}$

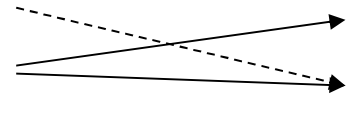
Inductive Step: Assume that there are $n-1$ OR connectives. Again, ψ is directly interpreted from $\text{range}(\mathcal{M}'.\text{F})$, where $\mathcal{M}'.\text{F} \in \{\{(a_0, a_1)\}, \{(a_0, a_2)\}, ..., \{(a_0, a_1), (a_0, a_2)\}, \dots, \{(a_0, a_1), (a_0, a_n)\}, \{(a_0, a_1), (a_0, a_2), \dots (a_0, a_n)\}\}$, i.e. $\psi = a_1\ \text{OR}\ \dots \text{OR}\ a_n$, such that the possible truth assignments of $\psi = 2^{(n-1)+1} - 1 = 2^n - 1$ are shown below:

a_1	$a_2 \dots$	a_n	$a_1\ OR \dots OR\ a_n$
F	F	F	F
T	F	F	T
⋮	⋮	⋮	⋮
T	T	T	T

$range(\mathcal{M}'.F)$
$\{\}$
$\{a_1\}$
⋮
$\{a_1, a_2, \dots, a_n\}$

Now we have to show that if having n of OR connectives, the possible number of assignments that evaluate the formula $a_1\ \text{OR} \dots \text{OR}\ a_{n+1}$ to true is equal to the number of subset of $\mathcal{M}$. As we know that the assignments to make the formula $a_1\ \text{OR} \dots \text{OR}\ a_n$ to true is $2^n - 1$, then obviously the possible truth assignments of $(a_1\ \text{OR} \dots \text{OR}\ a_n)\ \text{OR}\ a_{n+1} = (2^n - 1) + (2^n - 1) + 1 = 2^{n+1} - 1$. (Two $(2^n - 1)$ from the solid lines, and 1 from the dash line shown below)

a_1	$a_2 \dots$	a_n	$a_1\ OR \dots OR\ a_n$
F	F	F	F
T	F	F	T
⋮	⋮	⋮	⋮
T	T	T	T



a_{n+1}	
F	$(2^n - 1)$
T	$(2^n - 1) + 1$

The last step is to show that the number of $\text{range}(\mathcal{M}'.\text{F})$ is equal to $2^{n+1} - 1$ where there are n of OR connectives:

$range(\mathcal{M}'.F)$
$\{\}$
$\{a_1\}$
⋮
$\{a_1, a_2, \dots, a_n\}$

2^n $\cup$ $\{a_{n+1}\}$ $+$

$range(\mathcal{M}'.F)$
$\{\}$
$\{a_1\}$
⋮
$\{a_1, a_2, \dots, a_n\}$

$(2^n - 1)$

The number of $\text{range}(\mathcal{M}'.\text{F}) \cup \{a_{n+1}\}$ is 2^n. So, the possible number of subset of $\mathcal{M}$ is $2^n + (2^n - 1) = 2^{n+1} - 1$.□

Appendix C: Taylor's Approximation

$$\binom{n}{n} + \binom{n}{n-1} + \cdots + \binom{n}{0} = 2^n$$

$$\text{average} = E(V \times \binom{V}{V} + (V-1) \times \binom{V}{V-1} + \cdots + 0 \times \binom{V}{0})$$

$$\text{average} = \frac{V \times \binom{V}{V} + (V-1) \times \binom{V}{V-1} + \cdots + 0 \times \binom{V}{0}}{2^V}$$

Suppose that

$$y = V \times \binom{V}{V} + (V-1) \times \binom{V}{V-1} + \cdots + 0 \times \binom{V}{0}$$

$$y + V \times \binom{V}{0} + (V-1) \times \binom{V}{0} + \cdots + 0 \times \binom{V}{V} =$$

$$\left(V \times \binom{V}{V} + (V-1) \times \binom{V}{V-1} + \cdots + 0 \times \binom{V}{0}\right) + V \times \binom{V}{0} + (V-1) \times \binom{V}{0} + \cdots + 0 \times \binom{V}{V}$$

$$y + V \times \binom{V}{0} + (V-1) \times \binom{V}{0} + \cdots + 0 \times \binom{V}{V} = V(\binom{V}{0} + \binom{V}{1} + \cdots + \binom{V}{V})$$

$$y + V \times \binom{V}{0} + (V-1) \times \binom{V}{0} + \cdots + 0 \times \binom{V}{V} = V \times 2^V$$

Since $\binom{V}{r} = \binom{V}{V-r}$, we have

$$y + V \times \binom{V}{V} + (V-1) \times \binom{V}{V-1} + \cdots + 0 \times \binom{V}{0} = V \times 2^V$$

$$2y = V \times 2^V$$

$$y = \frac{2^V V}{2}$$

Such that,

$$\text{average} = \frac{V \times \binom{V}{V} + (V-1) \times \binom{V}{V-1} + \cdots + 0 \times \binom{V}{0}}{2^V} = \frac{2^V V}{2 \times 2^V} = \frac{V}{2}$$

Appendix D: Theorem, Proofs, and Algorithms

D.1 Basic Theorems and Proofs

Theorem D.1: The maximum number of bottommost terminal nodes in a COBDD tree is three.

Proof: The proof is straightforward and can be descriptively examined. Because the reduction rule 3 states that the results of the evaluation of a formula cannot be false; this implies that a true node is only appeared in a COBDD tree. Suppose that there is a set of the bottommost terminal nodes $\{x_n^1, x_n^2, \dots, x_n^m\}$ indexed by i, where $1 \le i \le m$, each element is connected to the true node specified by $lo(x_n^i) = \text{true}$ or $hi(x_n^i) = \text{true}$. As a result, $\{x_1^1, x_1^2, \dots, x_n^m\}$ can be grouped and collapsed into three types: (1) any x_n^i where $lo(x_n^i) = \text{true}$, (2) x_n^j where $hi(x_n^j) = \text{true}$, and (3) x_n^k where $lo(x_n^k)$ and $hi(x_n^k) = \text{true}$. As a result, the maximum number of the bottommost terminal nodes is three.□

Theorem D.2: When a new fresh variable is introduced in a COBDD tree, the number of nodes representing this variable is at most three.

Proof: Based on **theorem D.1** saying that the maximum number of bottommost terminal nodes is three. We only have to prove that if a new fresh variable node x_{n+1} is added in a COBDD tree, it will be the bottommost terminal nodes. Clearly, since the n variables COBDD tree with the order $< x_1, x_2, \dots, x_n >$ has no x_{n+1}, the introduction of node x_{n+1} will be placed at the bottommost according to the order.□

Theorem D.3: The size of an RCOBDD tree is linear to the number of SWN variables.

Proof: We construct the proof by assuming a workflow formula is in the form of EDNF (as being comparable to algorithm EDNF). A term is defined a single variable or a set of variables connected by AND connectives. We continue the proof by determining that the size of a RCOBDD tree is increased by the number of new fresh variables introduced.

Suppose that there is an RCOBDD tree with n variables $X = \{x_1, \dots, x_n\}$ and a new term with m variables $Y = \{y_1, \dots, y_m\}$ is introduced. We divide the set Y into two small sets. Existing variables of Y in the tree is a member of Y' where $Y' = \{y_i | y_i \in X\}$; otherwise, it becomes a member of Y'', $Y'' = \{y_j | y_j \in Y - Y'\}$. For each member in Y', we traverse along the path according to the ordered list $[x_1, \dots, x_n]$. For each $x_i \in Y'$ and $x_j \in Y'$, if there is no existing link between them, a new link is added from x_i to x_j ($i < j$). In the case of Y'', the tree is constructed by gradually adding new variables. For each member in Y'', a new node x_l is created and the links connected to the last node in Y' is connected to this node. If Y' is empty, this will be connected from x_0 (root node). Finally, the ordered list is updated according to the number of variables in the set X' where $X' = X \cup Y$.

From these two cases, it can be concluded that the maximum number of nodes (N) in the RCOBDD tree in terms of variables (V) is $N = |V| + 2$, where the 2 are from a starting (root) node x_0 and another is a labeled node T. □

For better understanding, the example in Figure D.1 describes how an RCOBDD tree is constructed such that it is linear to the number of workflow variables. The initial tree represents the workflow formula $X = (x_1 \text{ AND } x_2 \text{ AND } x_3 \text{ AND } x_4) \text{ XOR } (x_2 \text{ AND } x_3 \text{ AND } x_4)$ and a term $Y = (x_1 \text{ AND } x_4 \text{ AND } x_5 \text{ AND } x_6)$ is added to the tree by using XOR connective. As described in **theorem D.3**, Y is divided into the set of Y' and Y'' which are x_1 AND x_4 and x_5 AND x_6, respectively. A link (a red arrow) between the node x_1 and x_4 is introduced and the node x_5 and x_6 are newly created with the links (red arrows) from T.

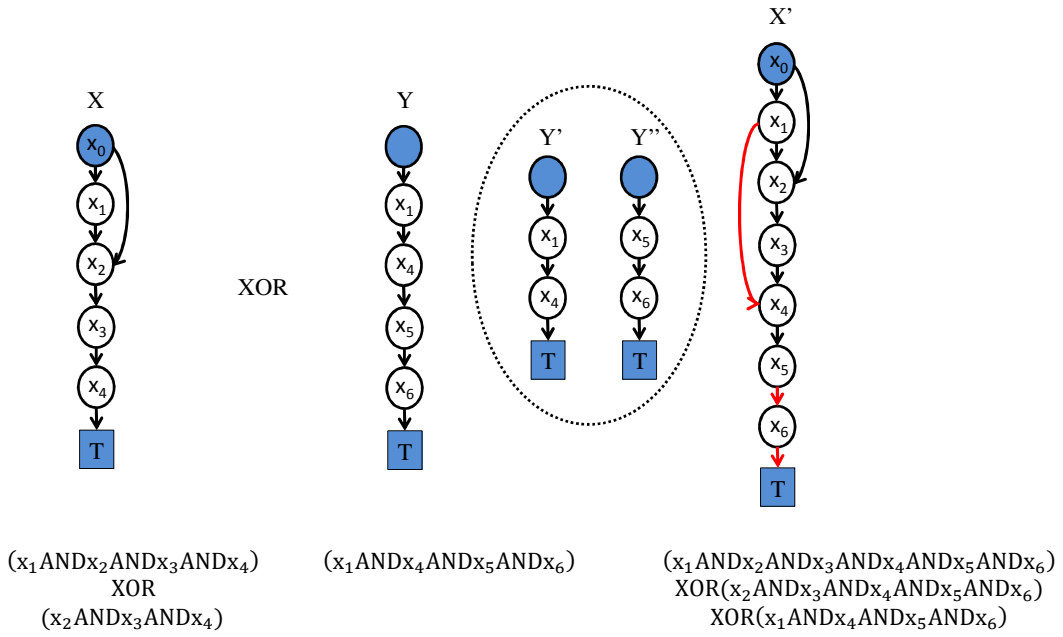


Figure D.1: Some Examples of Reduced COBDD Trees

D.2 COBDD-to-RCOBDD Transformation Algorithm

In this section, the algorithm for transforming COBDD to RCOBDD is presented. Given a COBDD tree in **Def. 5.5**, the algorithm produces an output in the RCOBDDs format (**Def. 5.6**). The algorithm is described below (see Figure D.2 for pseudo code):

1. Let C and RC be COBDD and RCOBDD trees respectively, initialise RC as $RC.N = \{x_0, \ldots x_m\} \cup \{T\}$.
2. t is a stack variable that is used to contain the sequence of variables, which will be pushed into the stack if the outgoing connection is true (connected by a solid line), that leads a COBDD to the true node T. t is initialised as $t.\text{push}(x_0)$.
3. Then, the algorithm CtoRC runs recursively, traversing along the paths using depth first search. If the considering node has a solid line to one successor, this node is pushed into the stack t.

4. If the true node T is reached, the set of relations RC. R is built by translating current values in the stack t (see line 10-16). The relation is determined from the two adjacent elements obtained from pop and top operations.

Figure D.3 depicts the example of COBDD-to-RCOBDD transformation

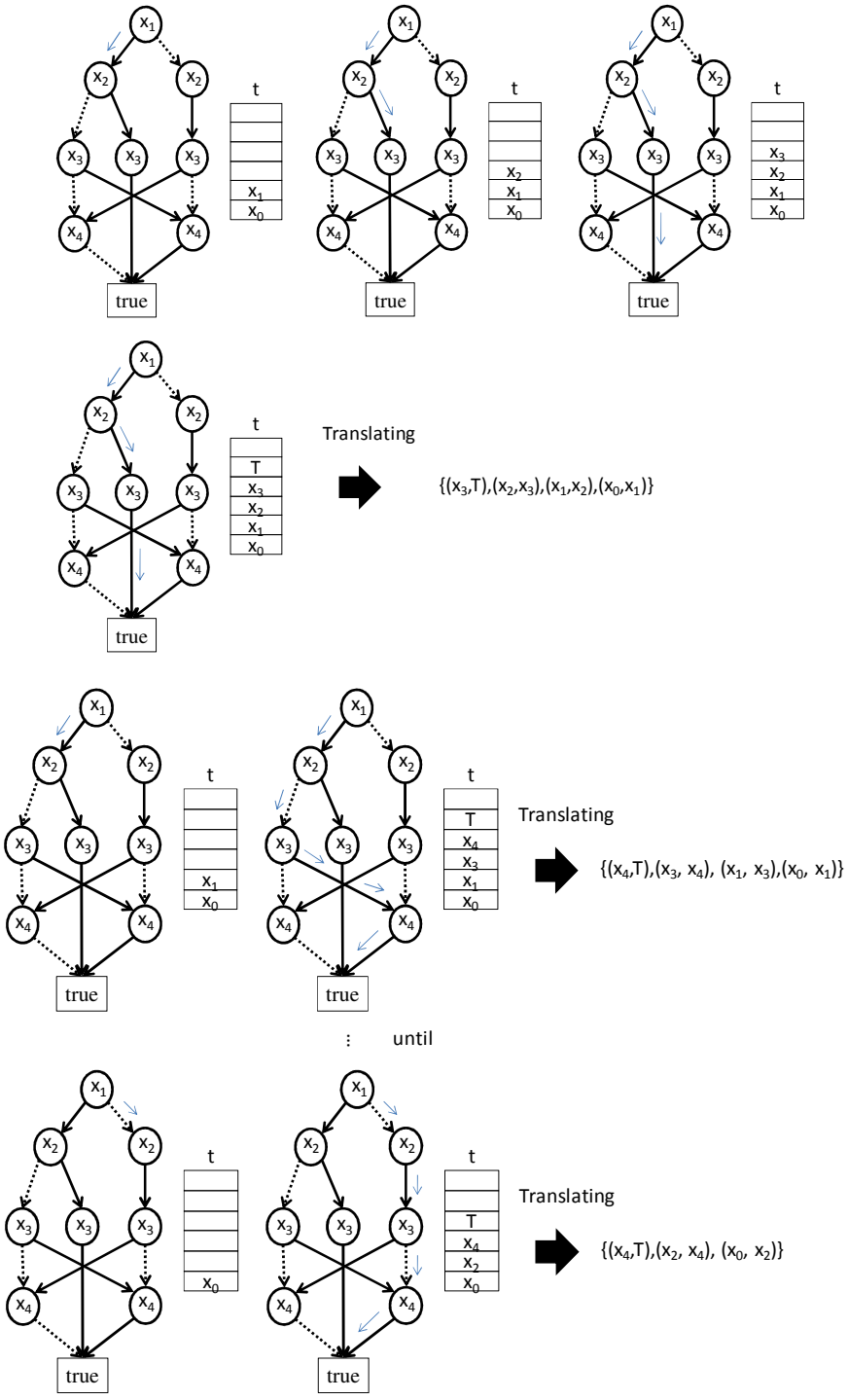


Figure D.3: Example of COBDD-to-RCOBDD Transformation

```
1.  Function C2RC(x, t)    // x is a COBDD root node and t is a temporary stack variable
2.  Begin
3.      If  hi(x)! = null
4.          t.push(x);
5.          C2RC(hi(x), t);
6.      End if
7.      If  lo(x)! = null
8.          C2RC(lo(x), t);
9.      End if
10.     If  x == T
11.         t.push(T);
12.         While t is not empty
13.             y = t.pop();
14.             RC.R = RC.R ∪ (t.top(), y);
15.         End while
16.     End if
17.     return;
18. End
```

Figure D.2: Algorithm for COBDD-to-RCOBDD Transformation

Obviously, this algorithm traverses all path of a given COBDD. The time complexity is $O(|E|)$ or in the worst case $O(|N|^2)$ where $|E|$ is the number of relations $|RH| + |RL|$ and $|N|$ is the number of nodes.

D.3 Algorithm SWN2FSM

In this section, the algorithm for converting an SWN into FSM is presented. The output will be used for checking the rest (Path and Direction parts) of a TS formula. An FSM is presented in Kripke frame (S, R, L), where S is a set of states, R is a set of transition relations, and L is a set of labelling function defined as $L: S \rightarrow P(S)$. $P(s)$ is a set of properties marked by the results of the Composite compliance checking.

Let the results (cr) from Composite compliance checking be stored in a relation $cr = (t, result)$, where t is a task and $result$ can be one of s_i, p_i, and u_i. Tasks in an SWN are considered states and the links are transition relations in FSM. The algorithm is described in the following steps (see Figure D.4 for algorithm SWN2FSM):

INPUT: an SWN $\mathcal{M} = (P, T, R, f, i, o, l)$, CR a set of cr where $cr = (t, result)$.
OUTPUT: an FSM $F = (S, R, L)$.

1. Let $\mathcal{M} = (P, T, R, f, i, o, l)$ be an SWN input, $F = (S, R, L)$ be an FSM output, and a set of relations $CR = \{cr \mid cr = (t, result)\}$.
2. $F.S = \mathcal{M}.T$. The set of tasks T is considered the set of states in F.
3. For each $cr \in CR$, $P(cr.t) = P(cr.t) \cup cr.result$. The state $s \in F.S$ is labelled with the $result$ from cr.
4. For each $p_i \in \mathcal{M}.P$
    i. If there is (t_a, p_i) and (p_i, t_c), a relation (t_a, t_c) is included into the set F.R.

```
1.  Function SWN2FSM(𝓜, F, CR)
2.  Begin
3.      F.S = 𝓜.T;
4.      For each cr ∈ CR
5.          P(cr.t) = P(cr.t) ∪ cr.result;
6.      End for
7.      For each p_i ∈ 𝓜.P
8.          For each q_j ∈ p_i.in
9.              For each r_k ∈ p_i.out
10.                 F.R = F.R ∪ (q_i, r_k);
11.             End for
12.         End for
13.     End for
14. End
```

Figure D.4: Pseudo Code for Algorithm SWN2FSM

Appendix E: Travel Agent Workflow Definitions

E.1 Travel Agent Workflow

Table E.1: Tasks Necessity, Dependencies, and Specifications

Tasks	Necessity	Types	Task Dependency Specification
Start	Yes	Split	(<FlightSearching> OR <HotelSearching> OR <CarSearching> OR <CruiseSearching> OR <TrainSearching>) XOR <PromotionPack>
		Join	N/A
FlightSearching	Yes	Split	<FlightReservation> XOR (<Location> OR <WeatherForecast>)
		Join	<Start>
HotelSearching	Yes	Split	<HotelReservation> XOR (<Location> OR <WeatherForecast> OR <HotelRating>)
		Join	<Start>
CarSearching	Yes	Split	<CarReservation> XOR (<Location> OR <WeatherForecast>)
		Join	<Start>
CruiseSearching	Yes	Split	<CruiseReservation> XOR (<Location> OR <WeatherForecast>)
		Join	<Start>
TrainSearching	Yes	Split	<TrainReservation> XOR (<Location> OR <WeatherForecast>)
		Join	<Start>
Location	No	Split	<FlightReservation> OR <HotelReservation> OR <CarReservation> OR <CruiseReservation> OR <TrainReservation>
		Join	<FlightSearching> OR <HotelSearching> OR <CarSearching> OR <CruiseSearching> OR <TrainSearching>
WeatherForecast	No	Split	<FlightReservation> OR <HotelReservation> OR <CarReservation> OR <CruiseReservation> OR <TrainReservation>
		Join	<FlightSearching> OR <HotelSearching> OR <CarSearching> OR <CruiseSearching> OR <TrainSearching>
HotelRating	No	Split	<HotelReservation>
		Join	<HotelSearching>
FlightReservation	Yes	Split	<FlightBooking>
		Join	<FlightSearching> XOR (<Location> OR <WeatherForecast>)
HotelReservation	Yes	Split	<HotelBooking>
		Join	<HotelSearching> XOR (<Location> OR <WeatherForecast> OR <HotelRating>)
CarReservation	Yes	Split	<CarBooking>
		Join	<CarSearching> XOR (<Location> OR <WeatherForecast>)
CruiseReservation	Yes	Split	<CruiseBooking>
		Join	<CruiseSearching> XOR (<Location> OR <WeatherForecast>)
TrainReservation	Yes	Split	<TrainBooking>
		Join	<TrainSearching> XOR (<Location> OR <WeatherForecast>)
FlightBooking	Yes	Split	<Payment> XOR (<Events> OR <Restaurant> OR <GuideandLocalTour> OR <Insurance>)
		Join	<FlightReservation>
HotelBooking	Yes	Split	<Payment> XOR (<Events> OR <Restaurant> OR

			<GuideandLocalTour> OR <Insurance>)
		Join	<HotelReservation>
CarBooking	Yes	Split	<Payment> XOR (<Events> OR <Restaurant> OR <GuideandLocalTour> OR <Insurance>)
		Join	<CarReservation>
CruiseBooking	Yes	Split	<Payment> XOR (<Events> OR <Restaurant> OR <GuideandLocalTour> OR <Insurance>)
		Join	<CruiseReservation>
TrainBooking	Yes	Split	<Payment> XOR (<Events> OR <Restaurant> OR <GuideandLocalTour> OR <Insurance>)
		Join	<TrainReservation>
PromotionPack	No	Split	<Payment>
		Join	<Start>
Events	No	Split	<Payment>
		Join	<FlightBooking> OR <HotelBooking> OR <CarBooking> OR <CruiseBooking> OR <TrainBooking>
Restaurant	No	Split	<Payment>
		Join	<FlightBooking> OR <HotelBooking> OR <CarBooking> OR <CruiseBooking> OR <TrainBooking>
Guideand LocalTour	No	Split	<Payment>
		Join	<FlightBooking> OR <HotelBooking> OR <CarBooking> OR <CruiseBooking> OR <TrainBooking>
Insurance	No	Split	<Payment>
		Join	<FlightBooking> OR <HotelBooking> OR <CarBooking> OR <CruiseBooking> OR <TrainBooking>
Payment	Yes	Split	<TravellingDocs>
		Join	((<FlightBooking> OR <HotelBooking> OR <CarBooking> OR <CruiseBooking> OR <TrainBooking>) XOR (<Events> OR <Restaurant> OR <GuideandLocalTour> OR <Insurance>)) XOR <PromotionPack>
TravellingDocs	Yes	Split	<End> XOR (<TouristInfo> OR <Statistics>)
		Join	<Payment>
TouristInfo	No	Split	<End>
		Join	<TravellingDocs>
Statistics	No	Split	<End>
		Join	<TravellingDocs>
End	Yes	Split	N/A
		Join	<TravellingDocs> XOR (<TourisInfo> OR <Statistics>)

Table E.2: Association between Service and Tasks of the Travel Agent Workflow

Tasks	Service Types	Acquired Services
Start	-	Dummy (S_{D1})
FlightSearching	Airline	Expedia (S_{XP}), Priceline (S_{PL}), Ebookers (S_{EB}), EazyJet (S_{EZ}), RyanAir (S_{RA})
HotelSearching	Hotel	Expedia (S_{XP}), Priceline (S_{PL}), Ebookers (S_{EB}), Agoda (S_{AG}), Booking (S_{BO})
CarSearching	Car Rental	Expedia (S_{XP}), Priceline (S_{PL}), Avis (S_{AV})
CruiseSearching	Cruise	WorldCruise (S_{WC})
TrainSearching	Train	EuroStar (S_{ES}), WorldTrain (S_{WT})
Location	Location	GoogleMap (S_{GM}), BingMap(S_{BM})

WeatherForecast	Weather	BBCWeather (S_{BBC}), Weather (S_{WTH})
HotelRating	Hotel Rating	Agoda (S_{AG}), Booking (S_{BO})
FlightReservation	Airline	Expedia (S_{XP}), Priceline (S_{PL}), Ebookers (S_{EB}), EazyJet (S_{EZ}), RyanAir (S_{RA})
HotelReservation	Hotel	Expedia (S_{XP}), Priceline (S_{PL}), Ebookers (S_{EB}), Agoda (S_{AG}), Booking (S_{BO})
CarReservation	Car Rental	Expedia (S_{XP}), Priceline (S_{PL}), Avis (S_{AV})
CruiseReservation	Cruise	WorldCruise (S_{WC})
TrainReservation	Train	EuroStar (S_{ES}), WorldTrain (S_{WT})
FlightBooking	Airline	Expedia (S_{XP}), Priceline (S_{PL}), Ebookers (S_{EB}), EazyJet (S_{EZ}), RyanAir (S_{RA})
HotelBooking	Hotel	Expedia (S_{XP}), Priceline (S_{PL}), Ebookers (S_{EB}), Agoda (S_{AG}), Booking (S_{BO}),
CarBooking	Car Rental	Expedia (S_{XP}), Priceline (S_{PL}), Avis (S_{AV})
CruiseBooking	Cruise	WorldCruise (S_{WC})
TrainBooking	Train	EuroStar (S_{ES}), WorldTrain (S_{WT})
PromotionPack	Travel Agent	TravelPro (S_{TP}), Expedia (S_{XP}), Priceline (S_{PL})
Events	Events	WorldEvents (S_{WE}), S_{E1}, S_{E2}, S_{E3}
Restaurant	Restaurant	WorldRestaurant (S_{WR}), S_{R1}, S_{R2}, S_{R3}
GuideandLocalTour	Tour	WorldLocalTour (S_{WG}), S_{G1}, S_{G2}, S_{G3}
Insurance	Insurance	EazyJet (S_{EZ}), Avis (S_{AV}), WorldInsurance (S_{WI})
Payment	Payment	Visa (S_{VS}), Master (S_{MS}), PayPal (S_{PP})
TravellingDocs	Document Processing	TravelDocs (S_{TD1}, S_{TD2})
TouristInfo	Tourist Information	TouristInfo (S_{TI})
Statistics	Statistics	DataMagagement (S_{DM}), BusinessIntel (S_{BI})
End	-	Dummy (S_{D2})

Table E.3: Logical Relations of Acquired Services

Tasks		Logical Relations
Start	Split	Order: S_{XP}, S_{PL}, S_{EB}, S_{EZ}, S_{RA}, S_{AG}, S_{BO}, S_{AV}, S_{WC}, S_{WT}, S_{ES}, S_{TP} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{AG}), (S_0, S_{BO}), (S_0, S_{WC}), (S_0, S_{WT}), (S_0, S_{TP}), (S_{XP}, S_{PL}), (S_{XP}, S_{EB}), (S_{XP}, S_{EZ}), (S_{XP}, S_{RA}), (S_{XP}, S_{AV}), (S_{XP}, T), (S_{PL}, S_{EB}), (S_{PL}, S_{EZ}), (S_{PL}, S_{RA}), (S_{PL}, S_{AV}), (S_{PL}, T), (S_{EB}, T), (S_{EZ}, T), (S_{RA}, T), (S_{AG}, T), (S_{BO}, T), (S_{AV}, T), (S_{WC}, T), (S_{WT}, S_{ES}), (S_{WT}, T), (S_{ES}, T), (S_{TP}, T)
	Join	(S_0, S_{D1}), (S_{D1}, T)
FlightSearching	Split	Order: S_{XP}, S_{PL}, S_{EB}, S_{EZ}, S_{RA}, S_{GM}, S_{BM}, S_{BBC}, S_{WHT} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{EZ}), (S_0, S_{RA}), (S_0, S_{GM}), (S_0, S_{BM}), (S_0, S_{BBC}), (S_0, S_{WHT}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{EZ}, T), (S_{RA}, T), (S_{GM}, T), (S_{BM}, T), (S_{BBC}, T), (S_{WHT}, T)
	Join	Order: S_{XP}, S_{PL}, S_{EB}, S_{EZ}, S_{RA} (S_0, S_{XP}), (S_0, S_{PL}), (S_{XP}, S_{PL}), (S_{XP}, S_{EB}), (S_{XP}, S_{EZ}), (S_{XP}, S_{RA}), (S_{XP}, T), (S_{PL}, S_{EB}), (S_{PL}, S_{EZ}), (S_{PL}, S_{RA}), (S_{PL}, T), (S_{EB}, T), (S_{EZ}, T), (S_{RA}, T)
HotelSearching	Split	Order: S_{XP}, S_{PL}, S_{EB}, S_{AG}, S_{BO}, S_{GM}, S_{BM}, S_{BBC}, S_{WHT} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{AG}), (S_0, S_{BO}), (S_0, S_{GM}), (S_0, S_{BM}), (S_0, S_{BBC}), (S_0, S_{WHT}), (S_{AG}, S_{BO}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{AG}, T), (S_{BO}, T), (S_{GM}, T), (S_{BM}, T), (S_{BBC}, T), (S_{WHT}, T)

	Join	Order: S_{XP}, S_{PL}, S_{EB}, S_{AG}, S_{BO} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{AG}), (S_0, S_{BO}), (S_{XP}, S_{PL}), (S_{XP}, S_{EB}), (S_{XP}, S_{AG}), (S_{XP}, S_{BO}), (S_{PL}, S_{EB}), (S_{PL}, S_{AG}), (S_{PL}, S_{BO}), (S_{EB}, S_{AG}), (S_{EB}, S_{BO}), (S_{AG}, S_{BO}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{AG}, T), (S_{BO}, T)
CarSearching	Split	Order: S_{XP}, S_{PL}, S_{AV}, S_{GM}, S_{BM}, S_{BBC}, S_{WHT} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{AV}), (S_0, S_{GM}), (S_0, S_{BM}), (S_0, S_{BBC}), (S_0, S_{WHT}), (S_{XP}, T), (S_{PL}, T), (S_{AV}, T), (S_{GM}, T) , (S_{BM}, T) , (S_{BBC}, T) , (S_{WHT}, T)
	Join	Order: S_{XP}, S_{PL}, S_{AV} (S_0, S_{XP}), (S_0, S_{PL}), (S_{XP}, S_{AV}), (S_{PL}, S_{AV}), (S_{XP}, T), (S_{PL}, T), (S_{AV}, T)
CruiseSearching	Split	Order: S_{WC}, S_{GM}, S_{BM}, S_{BBC}, S_{WHT} (S_0, S_{WC}), (S_0, S_{GM}), (S_0, S_{BM}), (S_0, S_{BBC}), (S_0, S_{WHT}), (S_{WC}, T), (S_{GM}, T) , (S_{BM}, T) , (S_{BBC}, T) , (S_{WHT}, T)
	Join	Order: S_{WC} (S_0, S_{WC}), (S_{WC}, T)
TrainSearching	Split	Order: S_{WT}, S_{ES}, S_{GM}, S_{BM}, S_{BBC}, S_{WHT} (S_0, S_{WT}), (S_0, S_{GM}), (S_0, S_{BM}), (S_0, S_{BBC}), (S_0, S_{WHT}), (S_{WT}, S_{ES}), (S_{WC}, T), (S_{ES}, T), (S_{GM}, T) , (S_{BM}, T) , (S_{BBC}, T) , (S_{WHT}, T)
	Join	Order: S_{WT}, S_{ES} (S_0, S_{WT}), (S_{WT}, S_{ES}), (S_{WT}, T), (S_{ES}, T)
Location	Split	Order: S_{XP}, S_{PL}, S_{EB}, S_{EZ}, S_{RA}, S_{AG}, S_{BO}, S_{AV}, S_{WC}, S_{WT}, S_{ES} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{EZ}), (S_0, S_{RA}), (S_0, S_{AG}), (S_0, S_{BO}), (S_0, S_{AV}), (S_0, S_{WC}), (S_0, S_{WT}), (S_0, S_{ES}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{EZ}, T), (S_{RA}, T), (S_{AG}, T), (S_{BO}, T), (S_{AV}, T), (S_{WC}, T), (S_{WT}, T), (S_{ES}, T)
	Join	Order: S_{GM}, S_{BM} (S_0, S_{GM}), (S_0, S_{BM}), (S_{GM}, T), (S_{BM}, T)
WeatherForecast	Split	Order: S_{XP}, S_{PL}, S_{EB}, S_{EZ}, S_{RA}, S_{AG}, S_{BO}, S_{AV}, S_{WC}, S_{WT}, S_{ES} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{EZ}), (S_0, S_{RA}), (S_0, S_{AG}), (S_0, S_{BO}), (S_0, S_{AV}), (S_0, S_{WC}), (S_0, S_{WT}), (S_0, S_{ES}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{EZ}, T), (S_{RA}, T), (S_{AG}, T), (S_{BO}, T), (S_{AV}, T), (S_{WC}, T), (S_{WT}, T), (S_{ES}, T)
	Join	Order S_{BBC}, S_{WHT} (S_0, S_{BBC}), (S_0, S_{WHT}), (S_{BBC}, T), (S_{WHT}, T)
HotelRating	Split	Order: S_{XP}, S_{PL}, S_{EB}, S_{AG}, S_{BO} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{AG}), (S_0, S_{BO}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{AG}, T), (S_{BO}, T)
	Join	Order: S_{AG}, S_{BO} (S_0, S_{AG}), (S_0, S_{BO}), (S_{AG}, S_{BO}), (S_{AG}, T), (S_{BO}, T)
FlightReservation	Split	Order: S_{XP}, S_{PL}, S_{EB}, S_{EZ}, S_{RA} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{EZ}), (S_0, S_{RA}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{EZ}, T), (S_{RA}, T)
	Join	Order: S_{XP}, S_{PL}, S_{EB}, S_{EZ}, S_{RA} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{EZ}), (S_0, S_{RA}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{EZ}, T), (S_{RA}, T)
HotelReservation	Split	Order: S_{XP}, S_{PL}, S_{EB}, S_{AG}, S_{BO} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{AG}), (S_0, S_{BO}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{AG}, T), (S_{BO}, T)
	Join	Order: S_{XP}, S_{PL}, S_{EB}, S_{AG}, S_{BO} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{AG}), (S_0, S_{BO}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{AG}, T), (S_{BO}, T)
CarReservation	Split	Order: S_{XP}, S_{PL}, S_{AV} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{AV}), (S_{XP}, T), (S_{PL}, T), (S_{AV}, T)
	Join	Order: S_{XP}, S_{PL}, S_{AV} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{AV}), (S_{XP}, T), (S_{PL}, T), (S_{AV}, T)
CruiseReservation	Split	Order: S_{WC} (S_0, S_{WC}), (S_{WC}, T)

	Join	Order: S_{WC} (S_0, S_{WC}), (S_{WC}, T)
TrainReservation	Split	Order: S_{WT}, S_{ES} (S_0, S_{WT}), (S_0, S_{ES}), (S_{WT}, T), (S_{ES}, T)
	Join	Order: S_{WT}, S_{ES} (S_0, S_{WT}), (S_0, S_{ES}), (S_{WT}, T), (S_{ES}, T)
FlightBooking	Split	Order: S_{VS}, S_{MS}, S_{PP}, S_{WE}, S_{E1}, S_{E2}, S_{WR}, S_{R1}, S_{R2}, S_{WG}, S_{G1}, S_{WI}, S_{EZ} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_0, S_{WE}), (S_0, S_{E1}), (S_0, S_{E2}), (S_0, S_{WR}), (S_0, S_{R1}), (S_0, S_{R2}), (S_0, S_{WG}), (S_0, S_{G1}), (S_0, S_{WI}), (S_0, S_{EZ}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T), (S_{WE}, T), (S_{E1}, T), (S_{E2}, T), (S_{WR}, T), (S_{R1}, T), (S_{R2}, T), (S_{WG}, T), (S_{G1}, T), (S_{WI}, T), (S_{EZ}, T)
	Join	Order: S_{XP}, S_{PL}, S_{EB}, S_{EZ}, S_{RA} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{EZ}), (S_0, S_{RA}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{EZ}, T), (S_{RA}, T)
HotelBooking	Split	Order: S_{VS}, S_{MS}, S_{PP}, S_{WE}, S_{E1}, S_{E2}, S_{WR}, S_{R1}, S_{R2}, S_{WG}, S_{G1}, S_{WI} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_0, S_{WE}), (S_0, S_{E1}), (S_0, S_{E2}), (S_0, S_{WR}), (S_0, S_{R1}), (S_0, S_{R2}), (S_0, S_{WG}), (S_0, S_{G1}), (S_0, S_{WI}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T), (S_{WE}, T), (S_{E1}, T), (S_{E2}, T), (S_{WR}, T), (S_{R1}, T), (S_{R2}, T), (S_{WG}, T), (S_{G1}, T), (S_{WI}, T)
	Join	Order: S_{XP}, S_{PL}, S_{EB}, S_{AG}, S_{BO} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{EB}), (S_0, S_{AG}), (S_0, S_{BO}), (S_{XP}, T), (S_{PL}, T), (S_{EB}, T), (S_{AG}, T), (S_{BO}, T)
CarBooking	Split	Order: S_{VS}, S_{MS}, S_{PP}, S_{WE}, S_{E1}, S_{E2}, S_{WR}, S_{R1}, S_{R2}, S_{WG}, S_{G1}, S_{WI}, S_{AV} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_0, S_{WE}), (S_0, S_{E1}), (S_0, S_{E2}), (S_0, S_{WR}), (S_0, S_{R1}), (S_0, S_{R2}), (S_0, S_{WG}), (S_0, S_{G1}), (S_0, S_{WI}), (S_0, S_{AV}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T), (S_{WE}, T), (S_{E1}, T), (S_{E2}, T), (S_{WR}, T), (S_{R1}, T), (S_{R2}, T), (S_{WG}, T), (S_{G1}, T), (S_{WI}, T), (S_{AV}, T)
	Join	Order: S_{XP}, S_{PL}, S_{AV} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{AV}), (S_{XP}, T), (S_{PL}, T), (S_{AV}, T)
CruiseBooking	Split	Order: S_{VS}, S_{MS}, S_{PP}, S_{WE}, S_{E1}, S_{E2}, S_{WR}, S_{R1}, S_{R2}, S_{WG}, S_{G1}, S_{WI} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_0, S_{WE}), (S_0, S_{E1}), (S_0, S_{E2}), (S_0, S_{WR}), (S_0, S_{R1}), (S_0, S_{R2}), (S_0, S_{WG}), (S_0, S_{G1}), (S_0, S_{WI}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T), (S_{WE}, T), (S_{E1}, T), (S_{E2}, T), (S_{WR}, T), (S_{R1}, T), (S_{R2}, T), (S_{WG}, T), (S_{G1}, T), (S_{WI}, T)
	Join	Order: S_{WC} (S_0, S_{WC}), (S_{WC}, T)
TrainBooking	Split	Order: S_{VS}, S_{MS}, S_{PP}, S_{WE}, S_{E1}, S_{E2}, S_{WR}, S_{R1}, S_{R2}, S_{WG}, S_{G1}, S_{WI} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_0, S_{WE}), (S_0, S_{E1}), (S_0, S_{E2}), (S_0, S_{WR}), (S_0, S_{R1}), (S_0, S_{R2}), (S_0, S_{WG}), (S_0, S_{G1}), (S_0, S_{WI}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T), (S_{WE}, T), (S_{E1}, T), (S_{E2}, T), (S_{WR}, T), (S_{R1}, T), (S_{R2}, T), (S_{WG}, T), (S_{G1}, T), (S_{WI}, T)
	Join	Order: S_{WT}, S_{ES} (S_0, S_{WT}), (S_0, S_{ES}), (S_{WT}, T), (S_{ES}, T)
Events	Split	Order: S_{VS}, S_{MS}, S_{PP} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T)
	Join	Order: S_{WE}, S_{E1}, S_{E2} (S_0, S_{WE}), (S_{WE}, S_{E1}), (S_{WE}, S_{E2}), (S_{E1}, S_{E2}), (S_{WE}, T), (S_{E1}, T), (S_{E2}, T)
Restaurants	Split	Order: S_{VS}, S_{MS}, S_{PP} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T)

	Join	Order: S_{WR}, S_{R1}, S_{R2} (S_0, S_{WR}), (S_{WR}, S_{R1}), (S_{WR}, S_{R2}), (S_{R1}, S_{R2}), (S_{WR}, T), (S_{R1}, T), (S_{R2}, T)
GuideandLocalTour	Split	Order: S_{VS}, S_{MS}, S_{PP} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T)
	Join	Order: S_{WG}, S_{G1} (S_0, S_{WG}), (S_{WG}, S_{G1}), (S_{WG}, T), (S_{G1}, T)
Insurance	Split	Order: S_{VS}, S_{MS}, S_{PP} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T)
	Join	Order: S_{WI}, S_{AV}, S_{EZ} (S_0, S_{WI}), (S_0, S_{AV}), (S_0, S_{EZ}), (S_{WI}, S_{AV}), (S_{WI}, S_{EZ}), (S_{AV}, S_{EZ}), (S_{WI}, T), (S_{AV}, T), (S_{EZ}, T)
PromotionPack	Split	Order: S_{VS}, S_{MS}, S_{PP} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T)
	Join	Order: S_{XP}, S_{PL}, S_{TP} (S_0, S_{XP}), (S_0, S_{PL}), (S_0, S_{TP}), (S_{XP}, T), (S_{PL}, T), (S_{TP}, T)
Payment	Split	Order: S_{TD1}, S_{TD2} (S_0, S_{TD1}), (S_0, S_{TD2}), (S_{TD1}, T), (S_{TD2}, T)
	Join	Order: S_{VS}, S_{MS}, S_{PP} (S_0, S_{VS}), (S_0, S_{MS}), (S_0, S_{PP}), (S_{VS}, T), (S_{MS}, T), (S_{PP}, T)
TravellingDocs	Split	Order: S_D, S_{TI}, S_{DM}, S_{BI} (S_0, S_{D2}), (S_0, S_{TI}), (S_{DM}, S_{TI}), (S_0, S_{BI}), (S_{D2}, T), (S_{TI}, T), (S_{DM}, T), (S_{BI}, T)
	Join	Order: S_{TD1}, S_{TD2} (S_0, S_{TD1}), (S_0, S_{TD2}), (S_{TD1}, T), (S_{TD2}, T)
TouristInfo	Split	Order: S_D (S_0, S_{D2}), (S_{D2}, T)
	Join	Order: S_{TI} (S_0, S_{TI}), (S_{TI}, T)
Statistics	Split	Order: S_D (S_0, S_{D2}), (S_{D2}, T)
	Join	Order: S_{DM}, S_{BI} (S_0, S_{DM}), (S_0, S_{BI}), (S_{DM}, T), (S_{BI}, T)
End	Split	N/A
	Join	Order: S_D (S_0, S_{D2}), (S_{D2}, T)

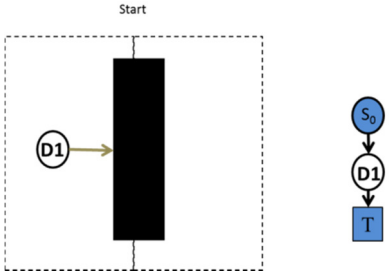
Table E.4: Services Glossary

Service	Associated Tasks	
Agoda	HotelSeaching, HotelRating, HoteReservation, HotelBooking	S_{AG}
Avis	CarSearching, CarReservation, CarBooking, Insurance	S_{AV}
BBCWeather	WeatherForecast	S_{BBC}
BingMap	Location	S_{BM}
Booking	HotelSeaching, HotelRating, HoteReservation, HotelBooking	S_{BO}
BusinessIntel	Statistics	S_{BI}
Datamanagement	Statistics	S_{DM}
EazyJet	FlightSearching, FlightReservation, FlightBooking, Insurance	S_{EZ}

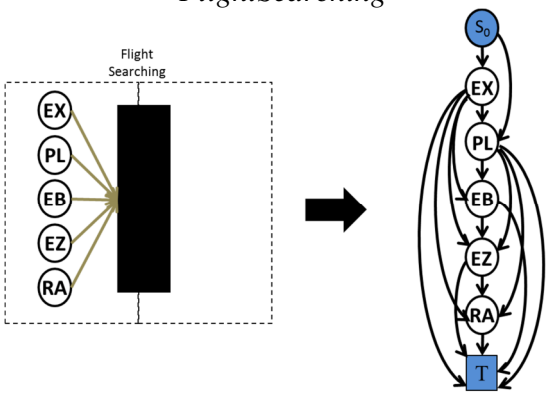
Ebookers	FlightSearching, FlightReservation, FlightBooking, HotelSearching, HotelReservation, HotelBooking,	S_{EB}
EuroStar	TrainSearching, TrainReservation, TrainBooking,	S_{ES}
Expedia	FlightSearching, FlightReservation, FlightBooking, HotelSearching, HotelReservation, HotelBooking, CarSearching, CarReservation, CarBooking, PromotionPack	S_{XP}
GoogleMap	Location	S_{GM}
Master	Payment	S_{MS}
Paypal	Payment	S_{PP}
Priceline	FlightSearching, FlightReservation, FlightBooking, HotelSearching, HotelReservation, HotelBooking, CarSearching, CarReservation, CarBooking, PromotionPack	S_{PL}
RyanAir	FlightSearching, FlightReservation, FlightBooking	S_{RA}
TourisInfo	TouristInfo	S_{TI}
TravelDocs	TravellingDocs	S_{TD}
TravelPro	PromotionPack	S_{TP}
Visa	Payment	S_{VS}
Weather	WeatherForecast	S_{WHT}
WorldCruise	CruiseSearching, CruiseReservation, CruiseBooking	S_{WC}
WorldEvents	Events	S_{WE}
WorldInsurance	Insurance	S_{WI}
WorldLocalTour	GuideandLocalTour	S_{WG}
WorldRestaurant	Restaurant	S_{WR}
WorldTrain	TrainSearching, TrainReservation, TrainBooking	S_{WT}

E.2 Service Composition Formulas (Service-join Types)

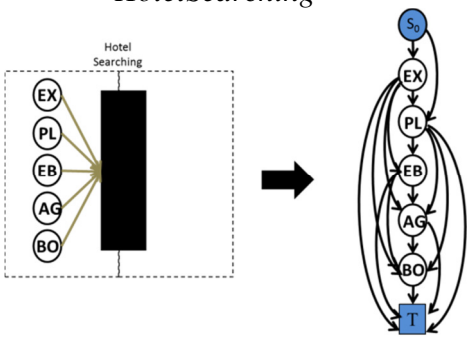
Start



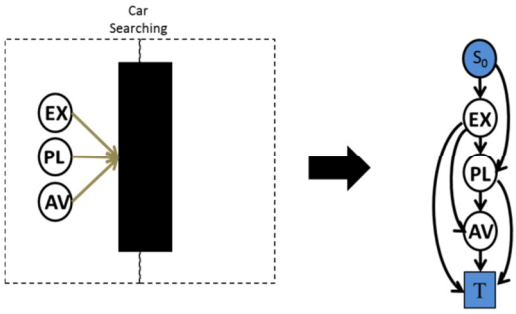
FlightSearching



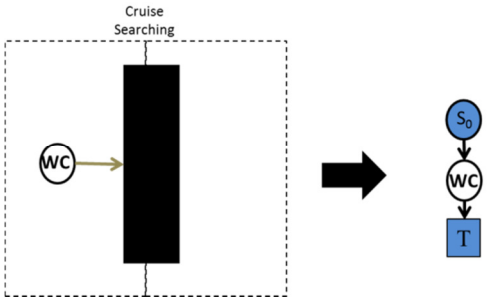
HotelSearching



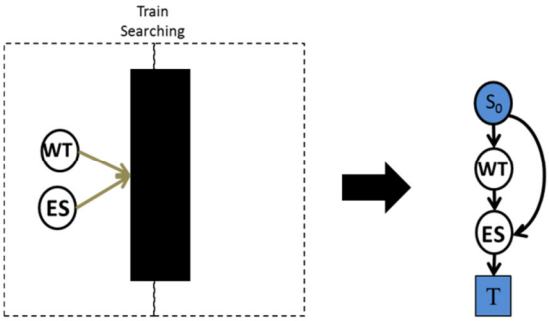
CarSearching



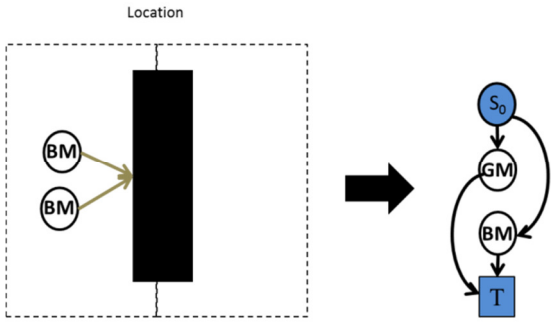
CruiseSearching



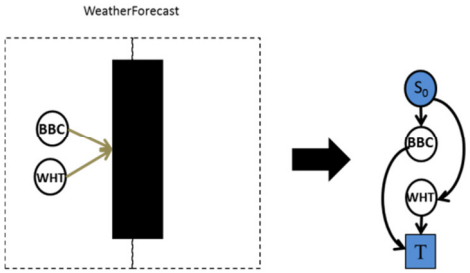
TrainSearching



Location



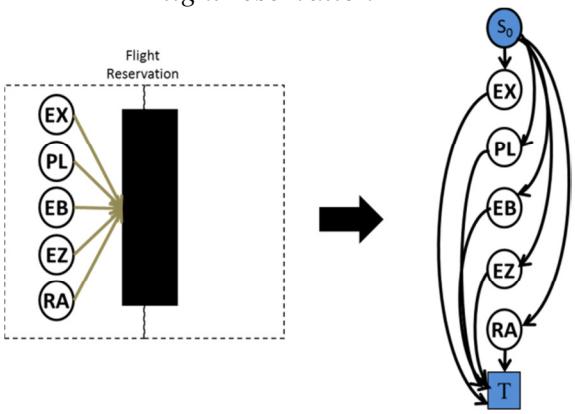
WeatherForecase



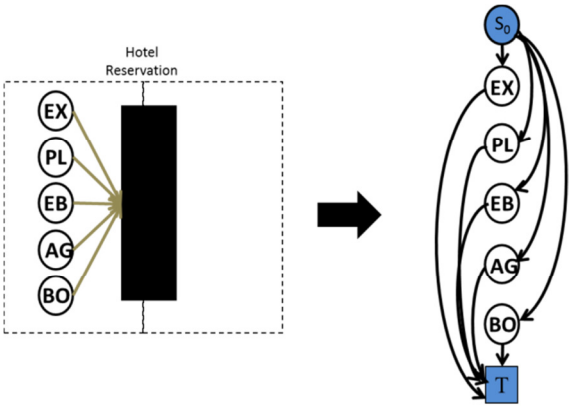
HotelRating



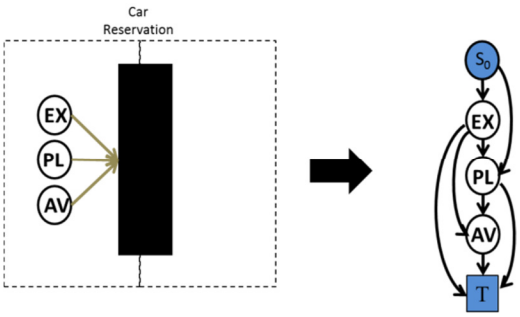
FlightReservation



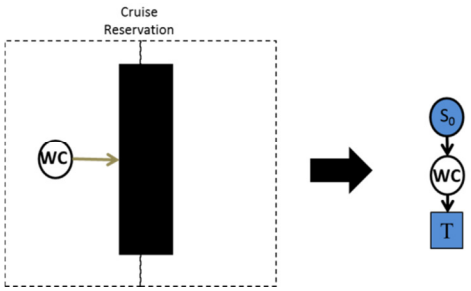
HotelReservation



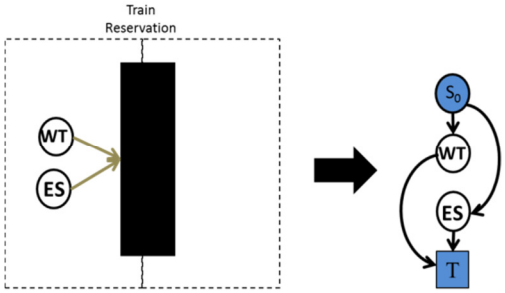
CarReservation



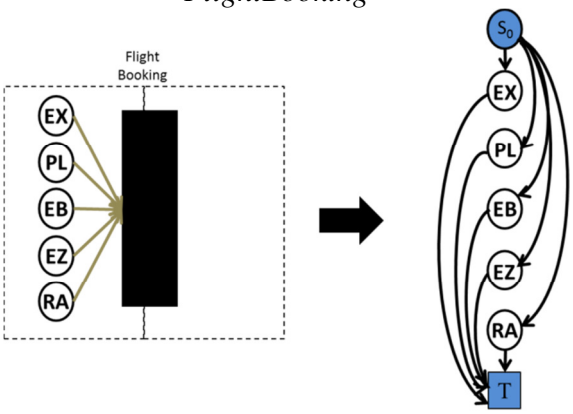
CruiseReservation



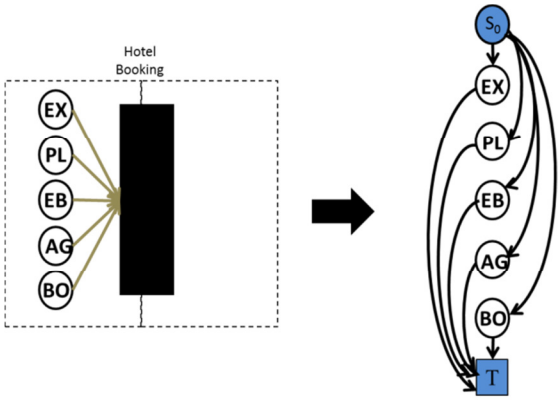
TrainReservation



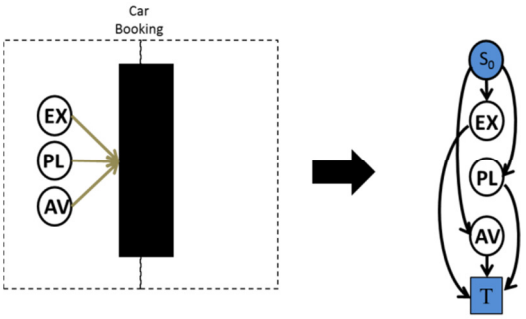
FlightBooking



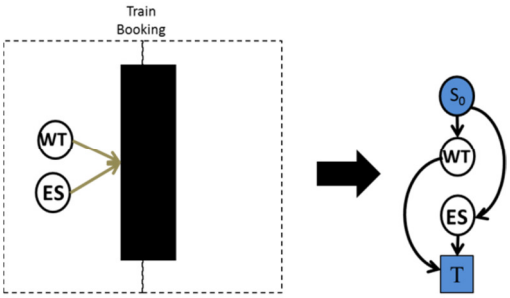
HotelBooking



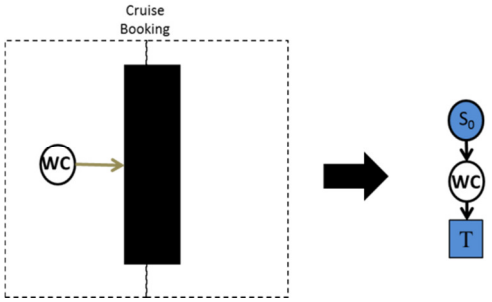
CarBooking



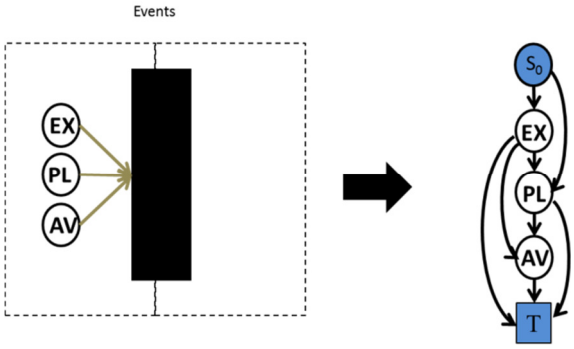
TrainBooking



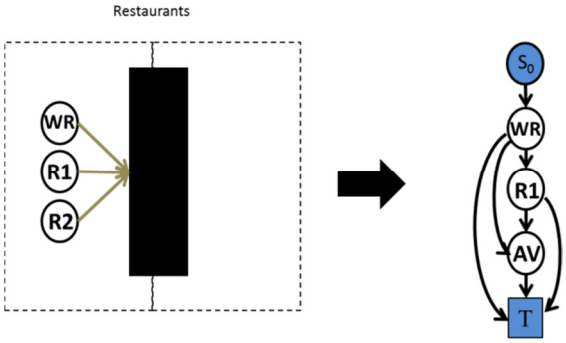
CruiseBooking



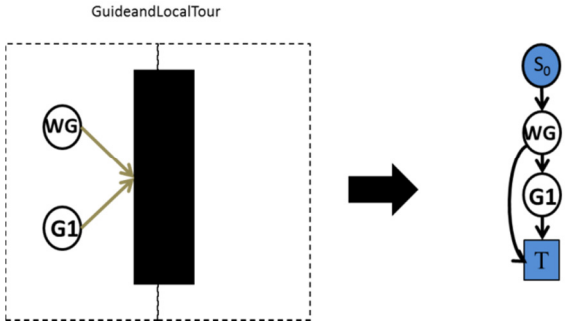
Events



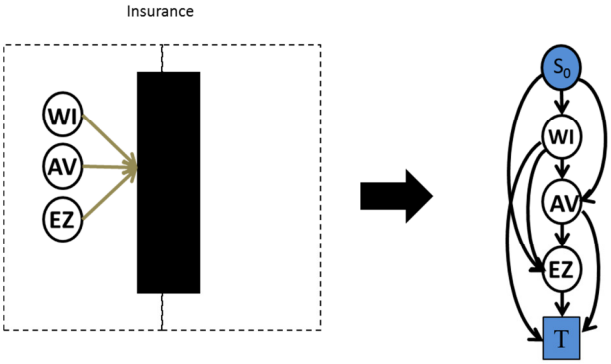
Restaurants



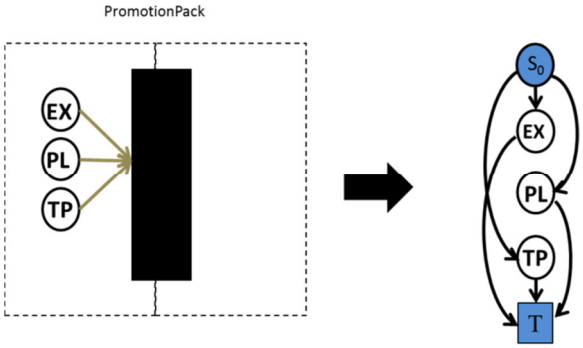
GuideandLocalTour



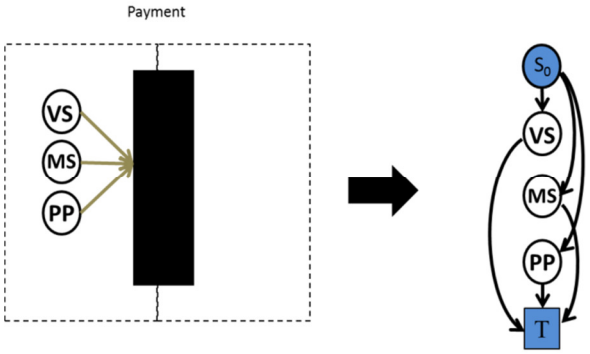
Insurance



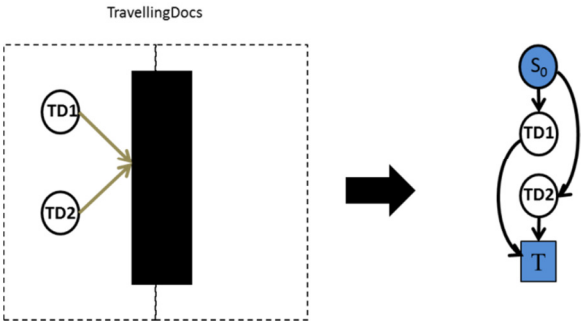
PromotionPack



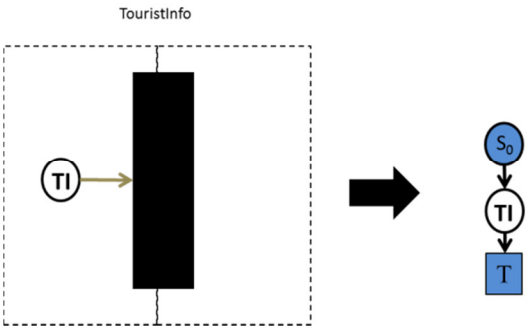
Payment



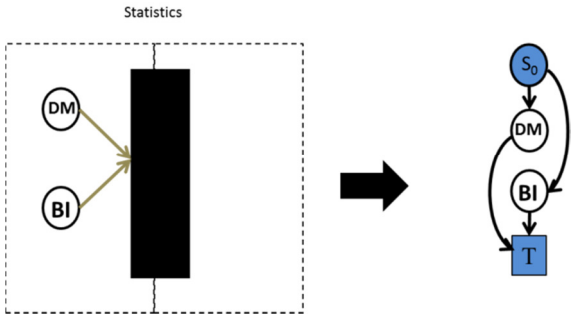
TravellingDocs



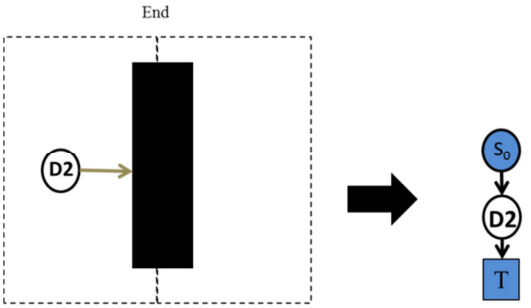
TouristInfo



Statistics

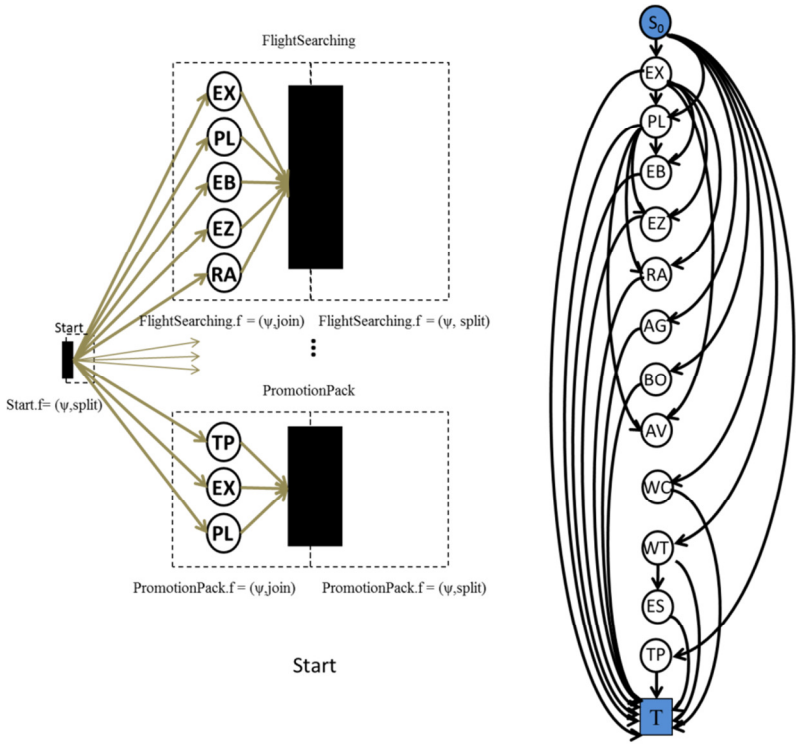


End

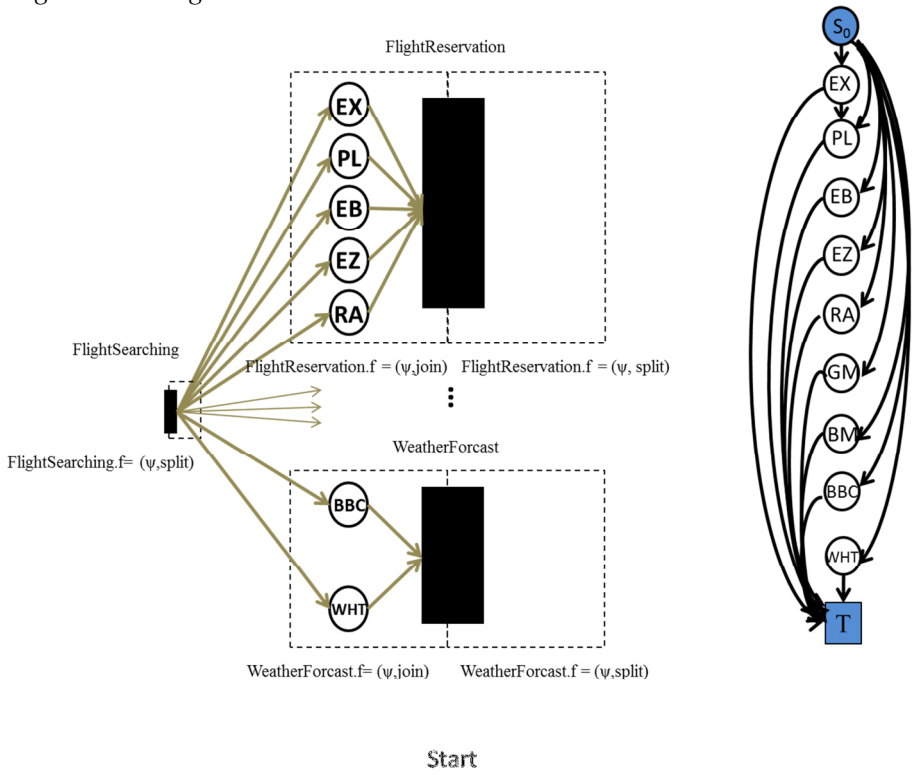


E.3 Service Separation Formulas (Service-split Types)

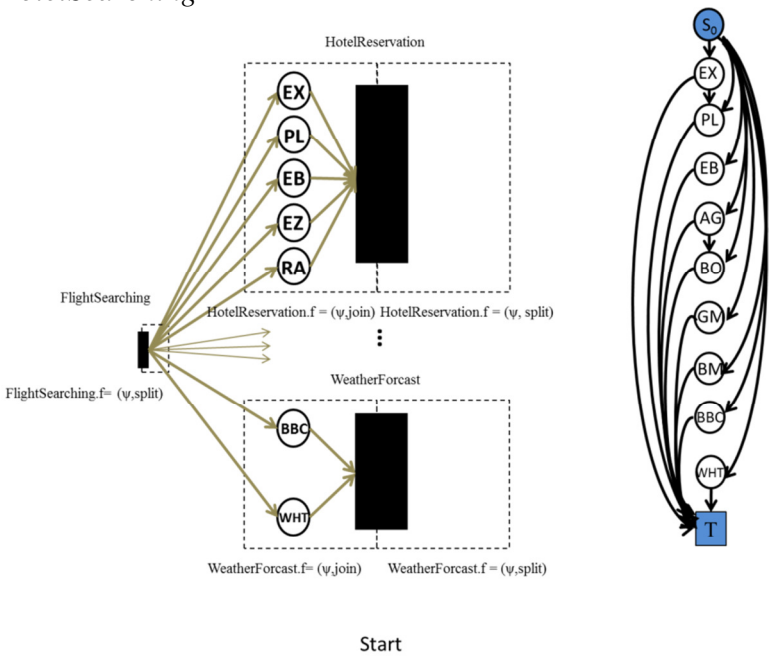
Start



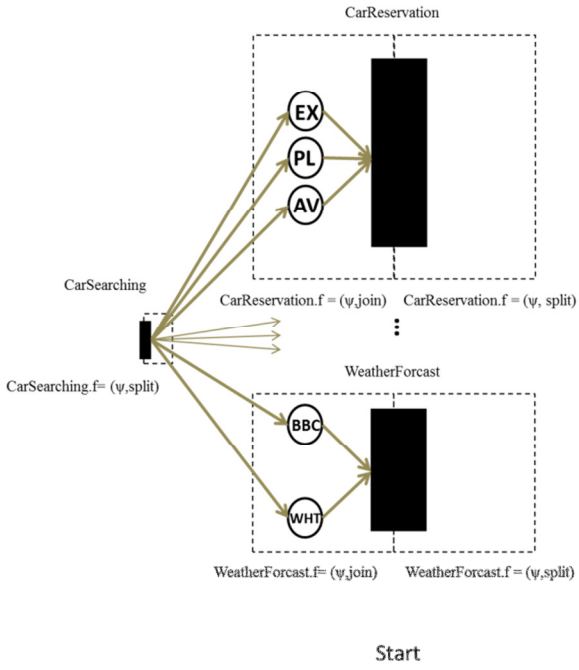
FlightSearching

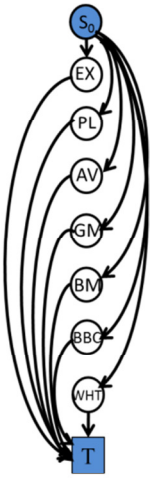


HotelSearching

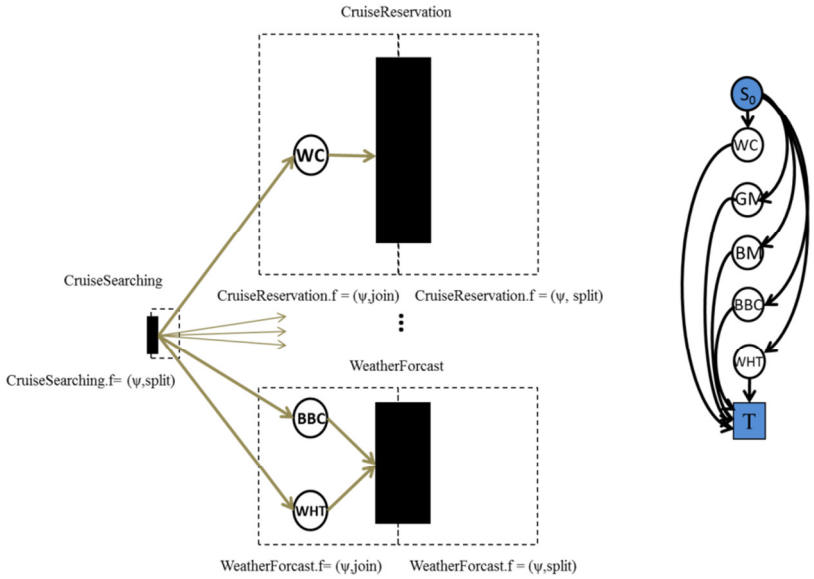


CarSearching

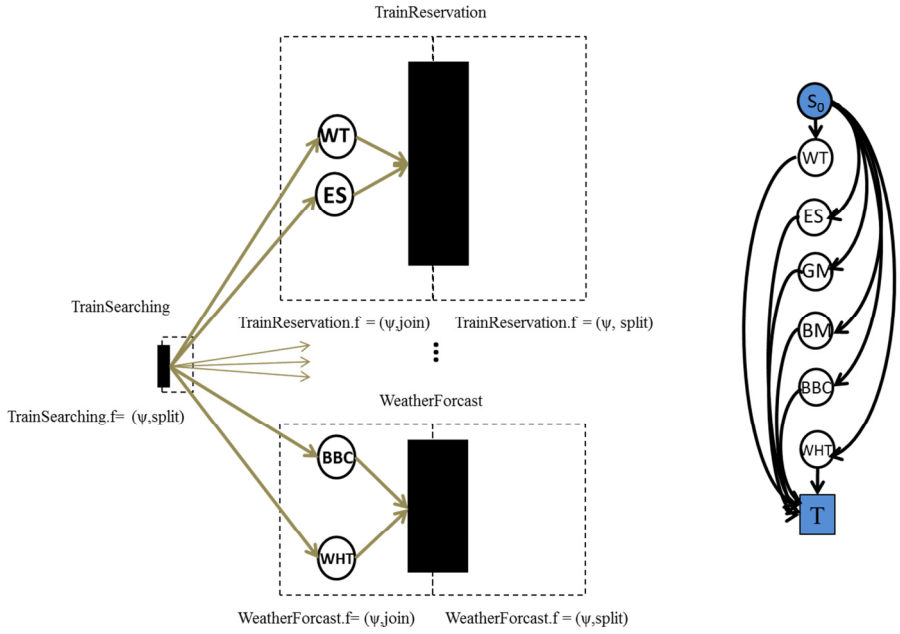




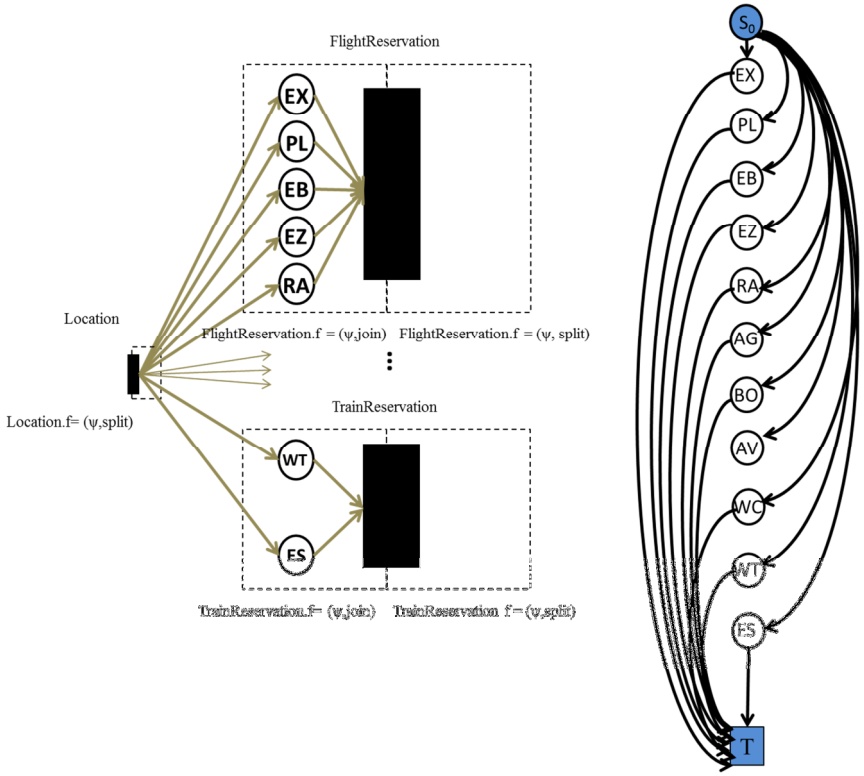
CruiseSearching



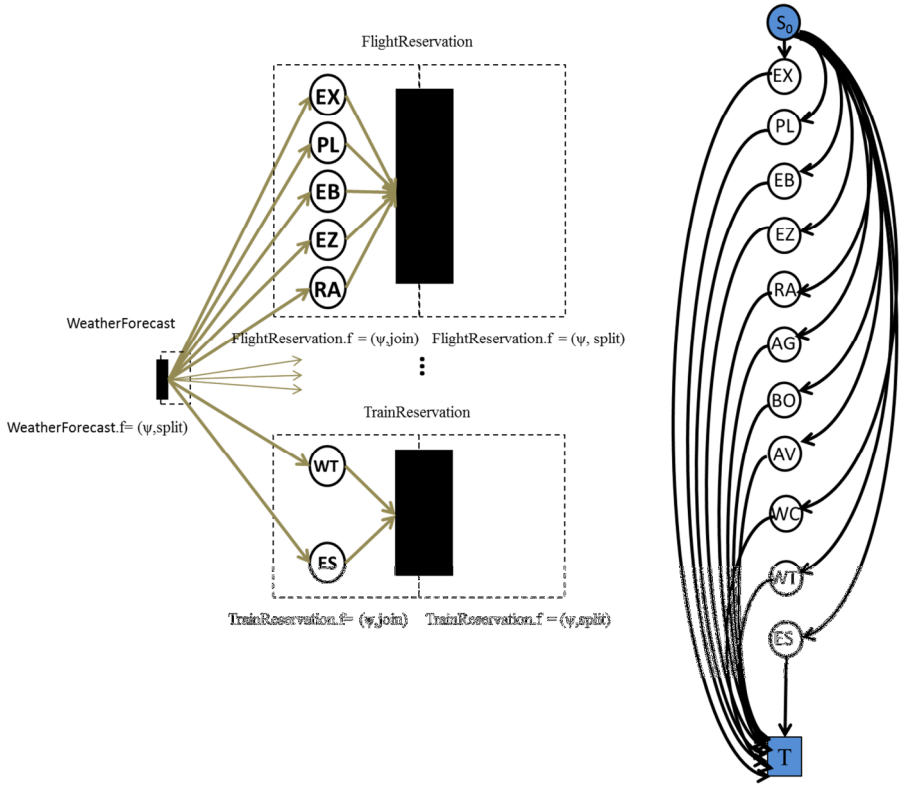
TrainSearching



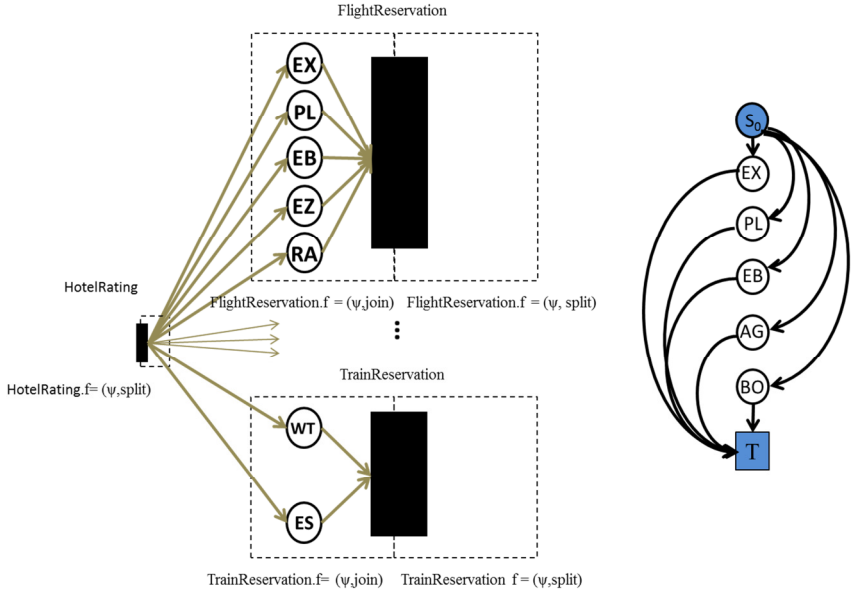
Location



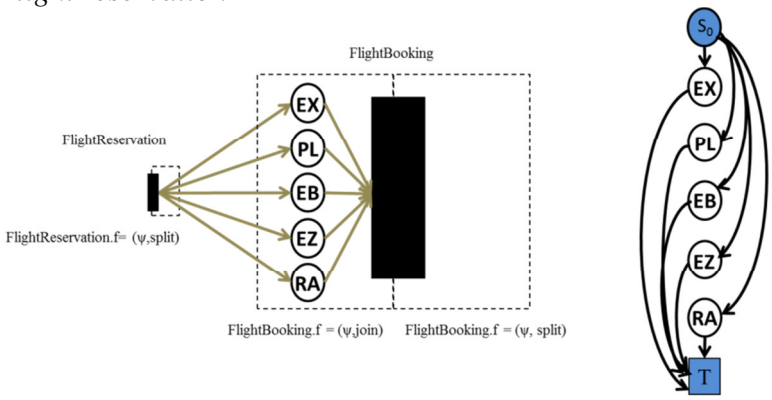
WeatherForecase



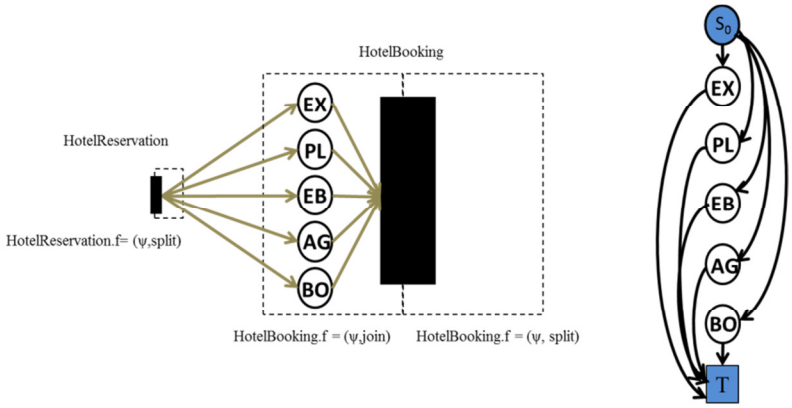
HotelRating



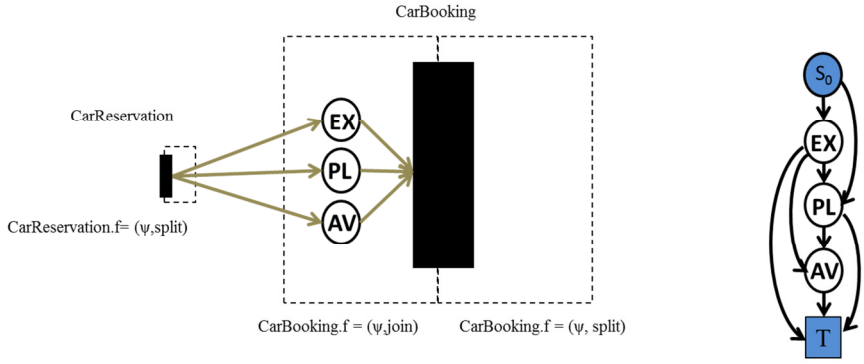
FlightReservation



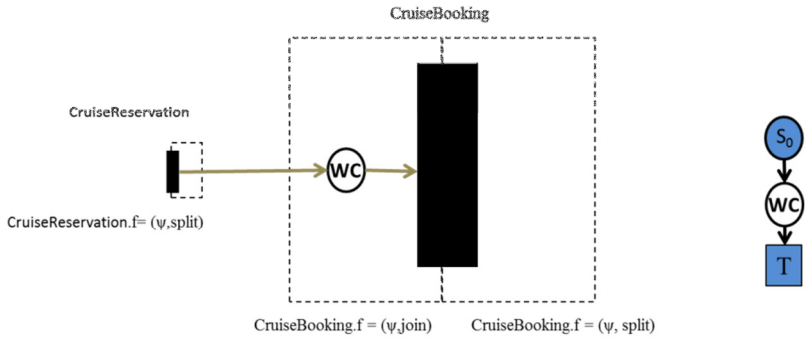
HotelReservation



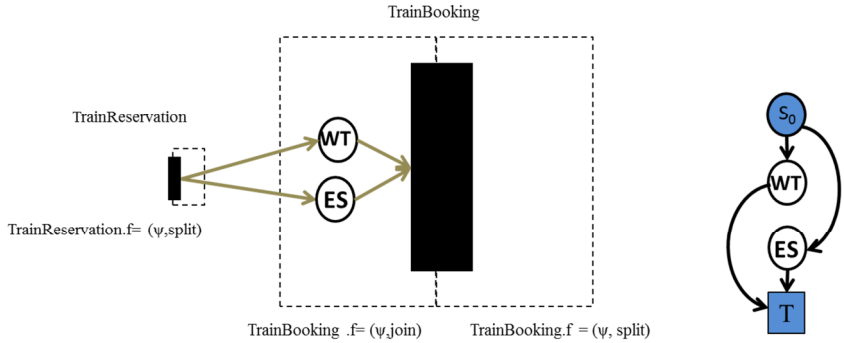
CarReservation



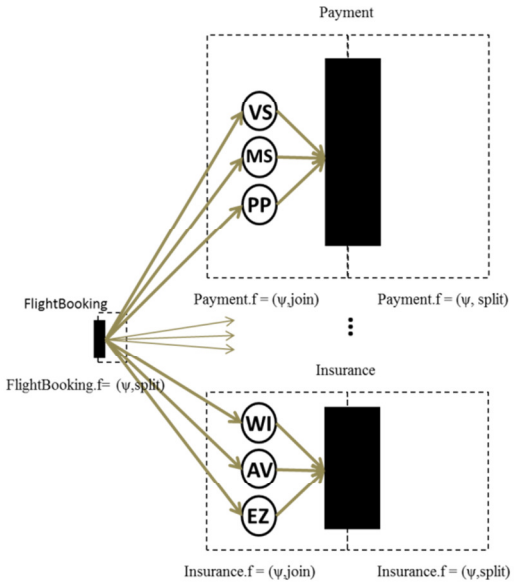
CruiseReservation

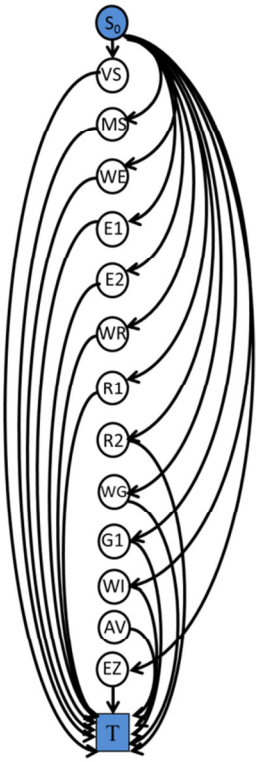


TrainReservation

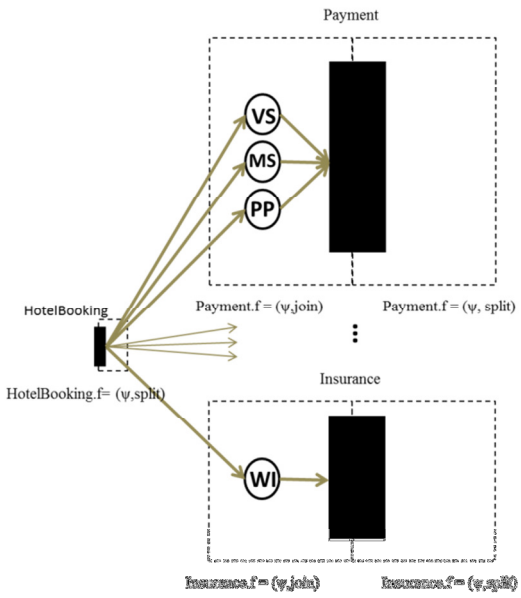


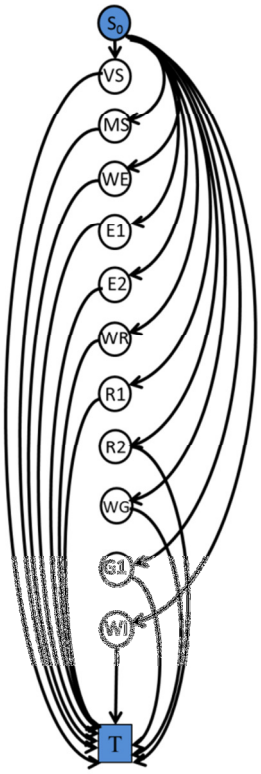
FlightBooking



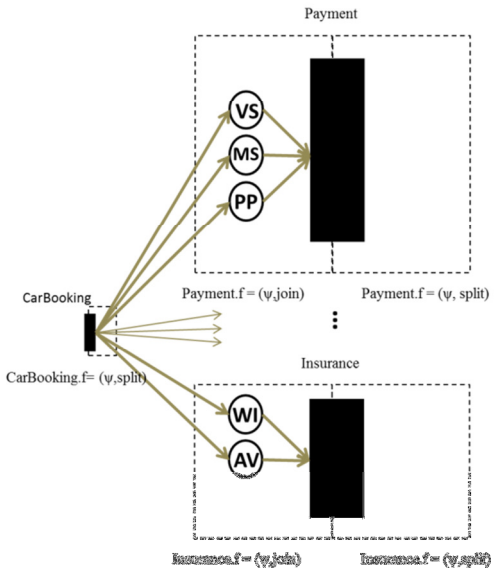


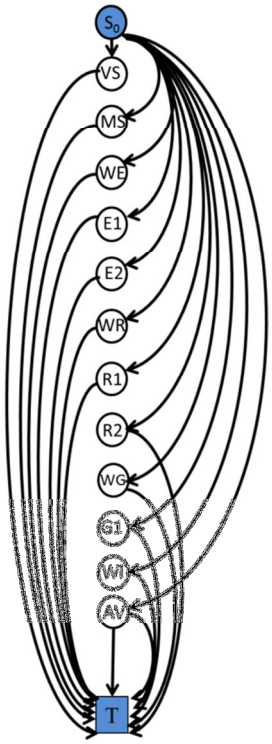
HotelBooking



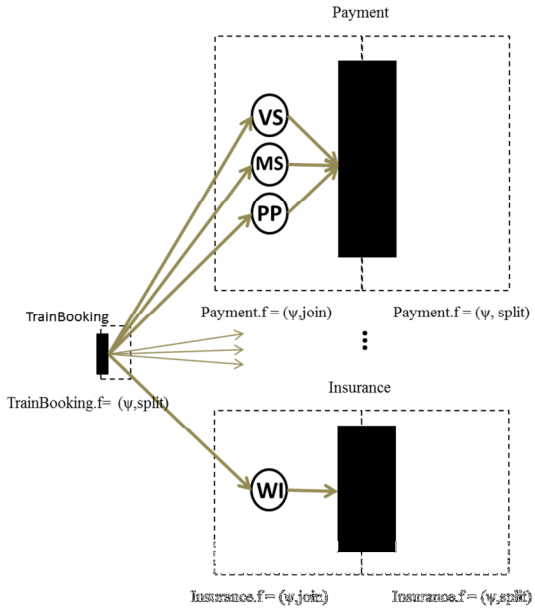


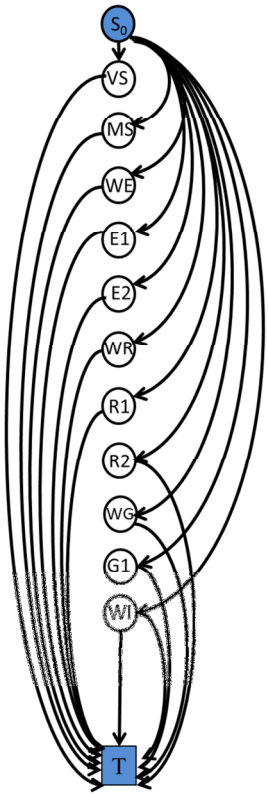
CarBooking



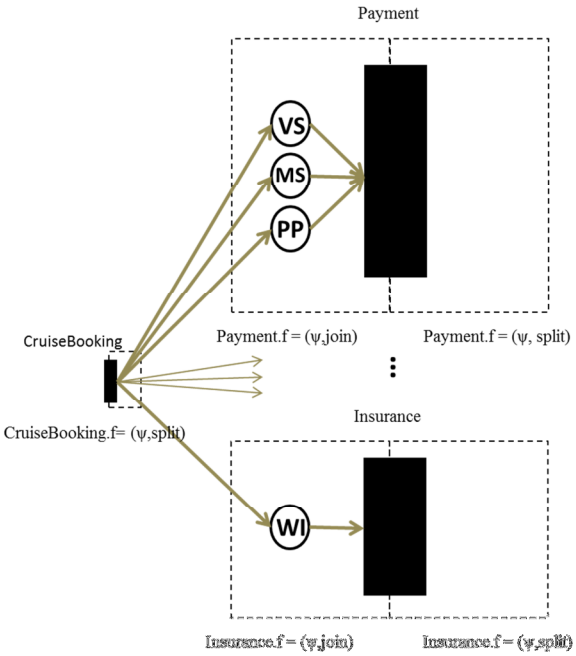


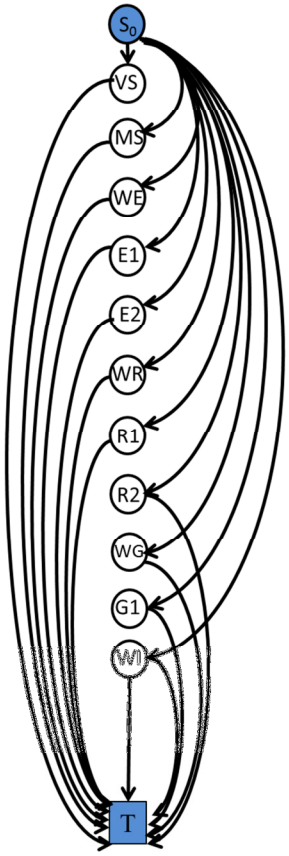
TrainBooking



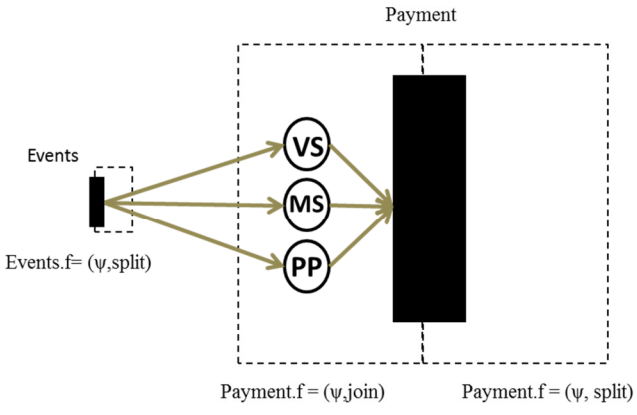


CruiseBooking



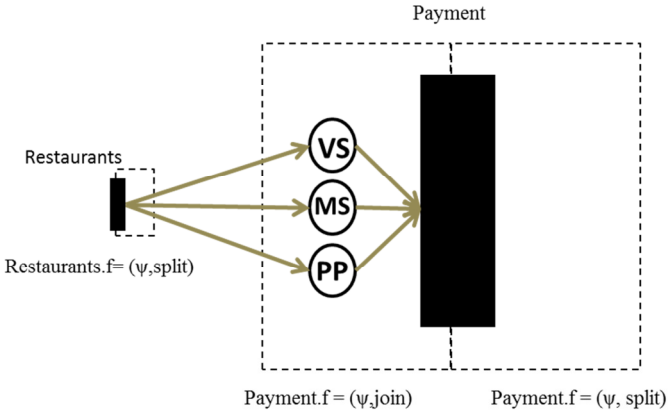


Events



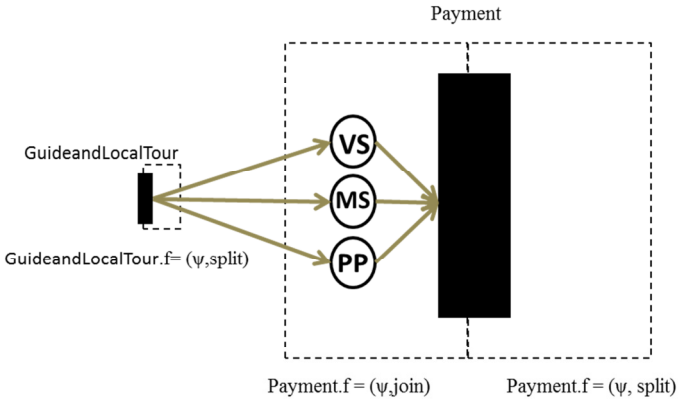


Restaurants



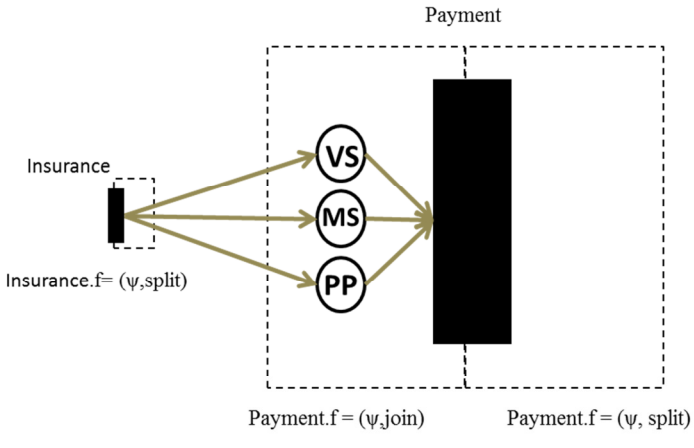


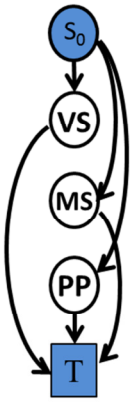
GuideandLocalTour



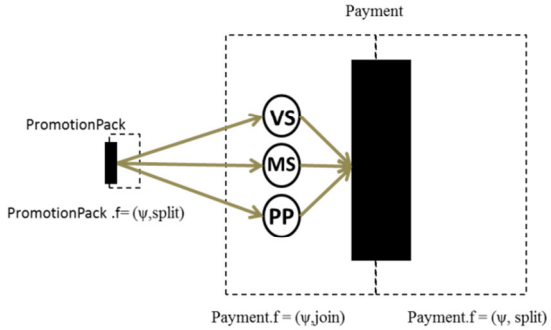


Insurance



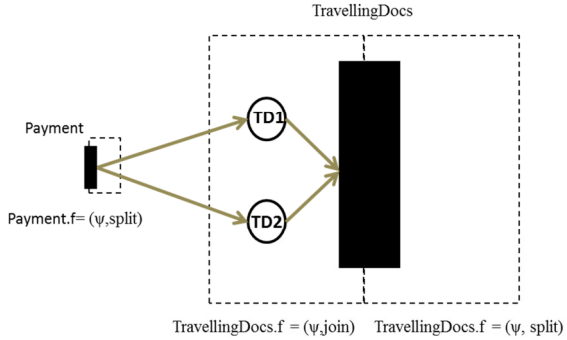


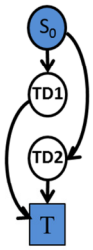
PromotionPack



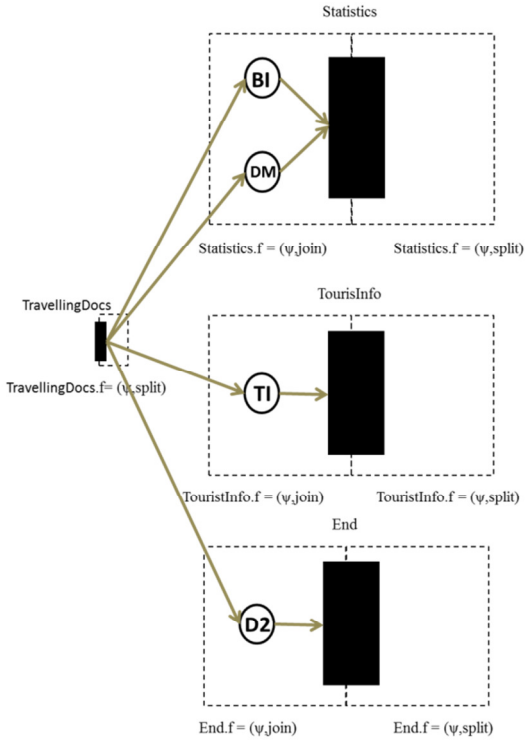


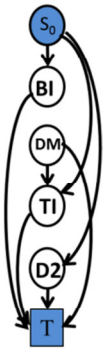
Payment



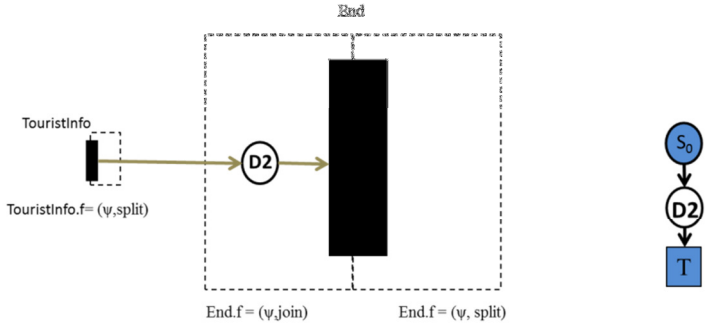


TravellingDocs

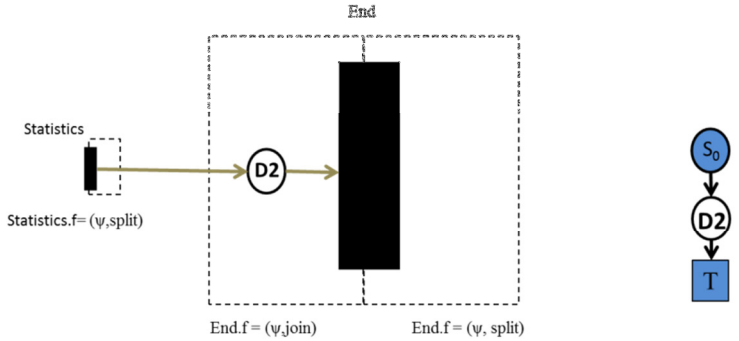




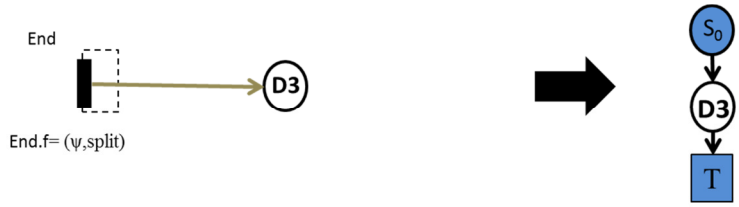
TouristInfo



Statistics



End



Appendix F: Programme Codes

F.1 Basic Classes

Atomic Proposition

```
classdef AP
    properties
        Name = '';
        Result = '';
    end
    methods
        function obj = AP(name,result)
            obj.Name = name;
            obj.Result = result;
        end
    end
end
```

Composite Formula

```
classdef Composite < handle
    properties
        Name = '';
        Element = '';
        D = [];
        Q = [];
        Lnode = [];
        Rnode = [];
        Task = [];
        Cloves = {};
        Ncloves = {};
        Parent = {};
    end
    methods
        function obj = Composite(name,element,d,q,task,cloves,ncloves)
           obj.Name = name;
           obj.Element = element;
           obj.D = d;
           obj.Q = q;
           obj.Cloves = cloves;
           obj.Task = task;
           obj.Ncloves = ncloves;
        end
        function output = isAtom(this)
            if isempty(this.Lnode) && isempty(this.Rnode)
                output = 1;
                return;
            end
                output = 0;
        end
        function addleft(this,node)
            ThisNode = this;
            while ~isempty(ThisNode.Lnode)
                ThisNode = ThisNode.Lnode;
```

```
            end
            ThisNode.Lnode = node;
        end
        function addright(this,node)
            ThisNode = this;
            while ~isempty(ThisNode.Rnode)
                ThisNode = ThisNode.Rnode;
            end
            ThisNode.Rnode = node;
        end
    end
end
```

Path Formulas

```
classdef Path < handle
    properties
        Element = '';
        Task = [];
        Lnode = [];
        Rnode = [];
        Parent = {};
        D = '';
    end
    methods
        function obj = Path(element,task,d)
            obj.Task = task;
            obj.Element = element;
            obj.D = d;
        end
        function addleft(this,node)
            ThisNode = this;
            while ~isempty(ThisNode.Lnode)
                ThisNode = ThisNode.Lnode;
                if strcmp(class(ThisNode),'Composite')
                    error('This leaf node is composite type, cannot add
!');
                end
            end
            ThisNode.Lnode = node;

            if isempty(node.Parent)
                node.Parent = {ThisNode};
            else
                for i=1:length(node.Parent)
                    if node.Parent{i} == ThisNode;
                        return;
                    end
                end
                node.Parent = horzcat(node.Parent,{ThisNode});
            end
        end
        function addright(this,node)
            ThisNode = this;
            k = ismember({'EX' 'AX' 'EG' 'AG' 'EF' 'AF'
'!'},this.Element);
            if any(k==1) || isempty(this.Element)
```

```
                error('Cannot add to the right node to EX AX EG AG EF AF
or
                !(not).');
            end
            while ~isempty(ThisNode.Rnode)
                ThisNode = ThisNode.Rnode;
                if strcmp(class(ThisNode),'Composite')
                    error('This leaf node is composite type, cannot add
!');
                end
            end
            ThisNode.Rnode = node;
            if isempty(node.Parent)
                node.Parent = {ThisNode};
            else
                for i=1:length(node.Parent)
                    if node.Parent{i} == ThisNode;
                        return;
                    end
                end
                node.Parent = horzcat(node.Parent,{ThisNode});
            end
        end
    end
end
```

Direction Formula

```
classdef Direction < handle
    properties
        Element = '';
        Dlnode = [];
        Drnode = [];
        Parent = {};
    end
    methods
        function obj = Direction(element)
            obj.Element = element;
        end
        function addleft(this,node)
            ThisNode = this;
            while ~isempty(ThisNode.Dlnode)
                ThisNode = ThisNode.Dlnode;
                if strcmp(class(ThisNode),'Path')
                    error('This leaf node is path type, cannot add !');
                end
            end
            ThisNode.Dlnode = node;
            if isempty(node.Parent)
                node.Parent = {ThisNode};
            else
                for i=1:length(node.Parent)
                    if node.Parent{i} == ThisNode;
                        return;
                    end
                end
                node.Parent = horzcat(node.Parent,{ThisNode});
```

```
            end
        end
        function addright(this,node)
            ThisNode = this;
            if strcmp('!',this.Element)
                error('Cannot add to the right node to ! (not).');
            end
            while ~isempty(ThisNode.Drnode)
                ThisNode = ThisNode.Drnode;
                if strcmp(class(ThisNode),'Path')
                    error('This leaf node is path type, cannot add !');
                end
            end
            ThisNode.Drnode = node;
            if isempty(node.Parent)
                node.Parent = {ThisNode};
            else
                for i=1:length(node.Parent)
                    if node.Parent{i} == ThisNode;
                        return;
                    end
                end
                node.Parent = horzcat(node.Parent,{ThisNode});
            end
        end
    end
end
```

RCOBDD Nodes

```
classdef Node < handle
    properties
        Name = '';
        ToNode = {};
        FromNode = {};
        Prop = {};
        mark;
    end
    methods
        function obj = Node(name,prop)
           obj.Name = name;
           obj.Prop = prop;
           obj.mark = 1;
        end
    end
end
```

Services

```
classdef Service < handle
    properties
      Name = '';
      Prop = {};
      Spec = {};
      TaskSplit = {};
      TaskJoin = {};
      Occur;
```

```
    end
    methods
        function obj = Service(name,prop)
            obj.Name = name;
            obj.Occur = 0;
            obj.Prop = prop;
        end
    end
end
```

Tasks

```
classdef Task < handle
    properties
        Name = '';
        ServiceJoin = {};
        si = {};
        Ti = {};
        ServiceJoinF = {};
        MatrixJoinF;
        ServiceSplit = {};
        so = {};
        To = {};
        ServiceSplitF = {};
        MatrixSplitF;
        TaskJoin = {};
        TaskSplit = {};
        Atoms = {};
        Visit = 0;
    end
    methods
        function obj = Task(name)
            obj.Name = name;
            obj.si = Node('si',{});
            obj.Ti = Node('Ti',{});
            obj.so = Node('so',{});
            obj.To = Node('To',{});
        end
        function disptask(this)
            disp('Task Join:');
            if ~isempty(this.TaskJoin)
                for i=1:length(this.TaskJoin)
                    disp(this.TaskJoin{i}.Name);
                end
            end
            disp('Task Split:');
            if ~isempty(this.TaskSplit)
                for i=1:length(this.TaskSplit)
                    disp(this.TaskSplit{i}.Name);
                end
            end
        end
        function dispservice(this)
            disp('Service Join:');
            if ~isempty(this.ServiceJoin)
                for i=1:length(this.ServiceJoin)
                    disp(this.ServiceJoin{i}.Name);
```

```
                end
            end
            disp('Service Split:');
            if ~isempty(this.ServiceSplit)
                for i=1:length(this.ServiceSplit)
                    disp(this.ServiceSplit{i}.Name);
                end
            end
        end
    end
end
```

Workflow

```
classdef Workflow < handle
    properties
        Services = {};
        Tasks = {};
        Start;
        End;
    end
    methods
        function obj = Workflow(start,End)
            obj.Start = start;
            obj.End = End;
        end
    end
end
```

F.2 Set Operations

Intersect

```
function output = intersectc(a,b)
    output = {};
    if ~isempty(a)
        if ~isempty(b)
           if ischar(a{1})
                a = unique(a);
                b = unique(b);
                for i = 1:length(a)
                   for j = 1:length(b)
                      if (strcmp(a{i},b{j}))
                         output = unionc(output,b(j));
                      end
                   end
                end
                return;
           end
           if strcmp(class(a{1}),'Task') || strcmp(class(a{1}),'Service')
||
            strcmp(class(a{1}),'Node') || strcmp(class(a{1}),'AP') ||
            strcmp(class(a{1}),'Composite')
                 for i = 1:length(a)
                   for j = 1:length(b)
                       if strcmp(b{j}.Name,a{i}.Name)
```

```
                            output = unionc(output,b(j));
                        end
                    end
                end
                return;
            end
        end
    end
end
```

Isequal

```
function output = isequalc(a,b)
    output = 0;
    for i=1:length(a)
        if ~ismemberc(b,a(i))
            output = 0;
            return;
        end
        output = 1;
    end
end
```

Ismember

```
function [output, loc] = ismemberc(a,b)
    output = 0;
    loc = 0;
     if ~isempty(a)
         if ischar(a{1})
             o = ismember(a,b);
             if isempty(find(o==1))
                output = 0;
                loc = 0;
             else
                output = 1;
                loc = find(o==1);
             end
             return;
         end
         if strcmp(class(a{1}),'Task') || strcmp(class(a{1}),'Service')
||
            strcmp(class(a{1}),'Node') || strcmp(class(a{1}),'AP') ||
            strcmp(class(a{1}),'Composite')
             f=0;
             for i=1:length(a)
                 if strcmp(a{i}.Name,b{1}.Name)
                     f=1;
                     loc=i;
                     output = 1;
                     return;
                 end
             end
         end
     end
end
```

Setdiff

```
function output = setdiffc(a,b)

    output = a;
    c = intersectc(a,b);

    for i = 1:length(c)
       [o , loc] = ismemberc(a,c(i));
       if o
           if length(a) == 1
                a = {};
           else
                a(loc) = [];
           end
       end
    end
    output = a;

end
```

SetXor

```
function output = setxorc(a,b)

    output = setxor(a,b);

end
```

Union

```
function output = unionc(a,b)

    output = {};

    if ~isempty(a)
        if ~isempty(b)
           % unionc type char
           if ischar(a{1})
               output = union(a,b);
               return;
           end

           % unionc type Task
           if strcmp(class(a{1}),'Task') || strcmp(class(a{1}),'Service')
             || strcmp(class(a{1}),'Node') || strcmp(class(a{1}),'AP')
             || strcmp(class(a{1}),'Composite')
               f=0;
               output = a;
               for i=1:length(b)
                   for j=1:length(a)
                       if strcmp(a{j}.Name ,b{i}.Name)
                          f=1;
                       end
                   end
                   if ~f
```

```
                        output = horzcat(output, b(i));
                    end
                    f=0;
                end
                return;
            end

        else
            output = a;
            return;
        end
    else
        output = b;
        return;
    end
end
```

F.3 Workflow Operations

AddServiceSplit

```
function addservicesplit(services, task, swn)
    new = setdiffc(services, task.ServiceSplit);
    if ~isempty(new)
        task.ServiceSplit = unionc(task.ServiceSplit, new);
        for i=1:length(new)
            s1 = new{i};
            task.ServiceSplitF =
unionc(task.ServiceSplitF,{Node(s1.Name,@s1.Prop)});
        end
        task.ServiceSplit = servicerelocation(task.ServiceSplit,swn);
        task.ServiceSplitF =
noderelocation(task.ServiceSplit,task.ServiceSplitF,swn);
    end
    sort = [];
    for i=1:length(services)
       [o,loc] = ismemberc(task.ServiceSplit,services(i));
       sort = union(sort,loc);
    end
    task.so.ToNode = unionc(task.so.ToNode,task.ServiceSplitF(sort(1)));
%
    le = length(sort); % Create new links of the rest
    if le > 1
        for i=1:le-1
            task.ServiceSplitF{sort(i)}.ToNode =
unionc(task.ServiceSplitF{sort(i)}.ToNode,task.ServiceSplitF(sort(i+1)));
        end
    end
    task.ServiceSplitF{sort(le)}.ToNode =
unionc(task.ServiceSplitF{sort(le)}.ToNode,{task.To}); % Add the last
link
    for i=1:length(services)
        services{i}.TaskJoin = unionc(services{i}.TaskJoin, {task});
    end
end
```

Check

```
function output = check(task)
    if ~strcmp(class(task),'Task')
        error('Input must be type task');
    end
    if length(task.ServiceJoin) ~= length(task.ServiceJoinF)
        error('ServiceJoin and ServiceJoinF are not matched');
    end
    for i=1:length(task.ServiceJoin)
        if ~strcmp(task.ServiceJoin{i}.Name, task.ServiceJoinF{i}.Name)
            error('ServiceJoin and ServiceJoinF are not matched');
        end
    end
    if length(task.ServiceSplit) ~= length(task.ServiceSplitF)
        error('ServiceSplit and ServiceSplitF numbers are not matched');
    end
    for i=1:length(task.ServiceSplit)
        if ~strcmp(task.ServiceSplit{i}.Name, task.ServiceSplitF{i}.Name)
            error('ServiceSplit and ServiceSplitF elements are not
matched');
        end
    end
    for i=1:length(task.TaskSplit)
        check(task.TaskSplit{i});
    end
    output = 1;
    return;
end
```

Delete Service

```
function deleteservice(service,task,swn)
    if ~strcmp(class(service),'Service')
        error('The first augument must be of type service');
    end
    if ~strcmp(class(task),'Task')
        error('The second augument must be of type task');
    end
    if ~strcmp(class(swn),'Workflow')
        error('The third argument must be of type workflow');
    end
    [o, loc] = ismemberc(task.ServiceJoin,{service});
    if ~o
        error('This Service does not exist');
    end
    tnode = task.ServiceJoinF{loc};
    tnode.mark = 0;
    if ~reachable(task.si,task.Ti,task.ServiceJoinF)
        reply = [];
        while ~strcmpi(reply,'Y')
           reply = input('Deleting this service will make the workflow
execution invalid.\nDo you want to execute? [Y]/[N]: ', 's');
           if strcmpi(reply,'N')
                task.si.ToNode = unionc(task.si.ToNode,{tnode});
                error('This service is not deleted');
           end
           if ~(strcmpi(reply,'N') || strcmpi(reply,'Y'))
```

```
                disp('Not recognize');
            end
        end
    end
    tnode.mark = 1;
    if length(task.ServiceJoin) == 1
        task.ServiceJoin = {};
    else
        task.ServiceJoin(loc) = [];
    end
    updatejoinlink(task,loc);
    t = intersectc(service.TaskJoin,task.TaskJoin);
    for i=1:length(t)
        updateservicesplit(t{i},swn);
    end
    service.Occur = service.Occur-1;
    [o,loc] = ismemberc(service.TaskSplit,{task});
    if length(service.TaskSplit) == 1
        service.TaskSplit = {};
    else
        service.TaskSplit(loc) = [];
    end
    for i=1:length(t)
        if ~ismemberc(t{i}.ServiceSplit,{service})
            [x,y] = ismemberc(service.TaskJoin,{t{i}});
            if length(service.TaskJoin) == 1
                service.TaskJoin = {};
            else
                service.TaskJoin(y) = [];
            end
        end
    end
    if service.Occur == 0
        [o,loc] = ismemberc(swn.Services,{service});
        if length(swn.Services) == 1
            swn.Services = {};
        else
            swn.Services(loc)=[];
        end
    end
end
```

Service Joined to Tasks

```
function servicejointask(services, task, ptasks, swn)
    % Services Join in a task
    if ~(iscell(services) && iscell(ptasks))
       error('The first and third arguments must be type cell');
    end
    if isempty(services)
       error('The first argument must have element');
    end
    if ~strcmp(class(services{1}),'Service')
       error('The first argument must be a cell of services');
    end
    if isempty(task)
       error('The second argument must be length 1');
```

```
    end
    if ~(strcmp(class(task),'Task'))
       error('The second agument must be type Task');
    end
    if ~strcmp(class(swn),'Workflow')
        error('The forth argument must be type workflow');
    end
    %  Add new services into workflow swn and update task
    new = setdiffc(services, swn.Services);
    if ~isempty(new)
        swn.Services = unionc(swn.Services,new);
    end
    new = setdiffc(services,task.ServiceJoin);
    if ~isempty(new)
        task.ServiceJoin = unionc(task.ServiceJoin, new);
            for i=1:length(new)
                new{i}.Occur = new{i}.Occur+1;  %  Keep track the number
of occurence of this service in a workflow
                s1 = new{i};
                task.ServiceJoinF =
unionc(task.ServiceJoinF,{Node(s1.Name,@s1.Prop)});
            end
            task.ServiceJoin = servicerelocation(task.ServiceJoin,swn);
            task.ServiceJoinF =
noderelocation(task.ServiceJoin,task.ServiceJoinF,swn);
    end
    sort = [];
    for i=1:length(services)
       [o,loc] = ismemberc(task.ServiceJoin,services(i));
       sort = union(sort,loc);
    end
    task.si.ToNode = unionc(task.si.ToNode,task.ServiceJoinF(sort(1)));
    le =length(sort);  %  Create links for the rest
    if le > 1
        for i=1:le-1
            task.ServiceJoinF{sort(i)}.ToNode =
unionc(task.ServiceJoinF{sort(i)}.ToNode,task.ServiceJoinF(sort(i+1)));
        end
    end
    task.ServiceJoinF{sort(le)}.ToNode =
unionc(task.ServiceJoinF{sort(le)}.ToNode,{task.Ti});  %  Add the last
link
    for i=1:length(new)
       new{i}.TaskSplit = unionc(new{i}.TaskSplit, {task});
    end
    for i=1:length(ptasks)
        addservicesplit(services,ptasks{i},swn);
    end
end
```

Tasks Joined to Tasks

```
function taskjointask(task1, task2, swn)
%   Task Join
    if ~(strcmp(class(task1), 'Task') && strcmp(class(task2),'Task'))
       error('Both arguments must be type Task');
    end
```

```
    % update workflow
    if ~ismemberc(swn.Tasks,{task1})
        swn.Tasks = unionc(swn.Tasks,{task1});
    end
    if ~ismemberc(swn.Tasks,{task2})
        swn.Tasks = unionc(swn.Tasks,{task2});
    end
    % update task
    if ~strcmp (task1.Name,task2.Name)
        task1.TaskJoin = unionc(task1.TaskJoin, {task2});
    else
        error('Cannot join itself');
    end
end
```

Tasks Split to Tasks

```
function tasksplittask(task1, task2, swn)
%   Task Split
    if ~((strcmp(class(task1), 'Task')) && (strcmp(class(task2),'Task')))
       error('Both arguments must be type Task');
    end
    if ~ismemberc(swn.Tasks,{task1})
        swn.Tasks = unionc(swn.Tasks,{task1});
    end
    if ~ismemberc(swn.Tasks,{task2})
        swn.Tasks = unionc(swn.Tasks,{task2});
    end
    if ~strcmp(task1.Name,task2.Name)
        task1.TaskSplit = unionc(task1.TaskSplit, {task2});
    else
        error('Cannot split to itself');
    end
end
```

Update Nodes

```
function updatefromnode(swn)
%UPDATEFROMNODE Summary of this function goes here
%   Detailed explanation goes here
     for i=1:length(swn.Tasks)
        task = swn.Tasks{i};
        % ServiceJoin
        % ServiceJoin starting at si
        for j=1:length(task.si.ToNode)
            tnode = task.si.ToNode{j};
            tnode.FromNode = unionc(tnode.FromNode,{task.si});
        end
        for j=1:length(task.ServiceJoinF)
            cnode = task.ServiceJoinF{j};
            for k=1:length(cnode.ToNode)
                tnode = cnode.ToNode{k};
                tnode.FromNode = unionc(tnode.FromNode,{cnode});
            end
        end
        %ServiceSplit
        for j=1:length(task.so.ToNode)
```

```
            tnode = task.so.ToNode{j};
            tnode.FromNode = unionc(tnode.FromNode,{task.so});
        end
        for j=1:length(task.ServiceSplitF)
            cnode = task.ServiceSplitF{j};
            for k=1:length(cnode.ToNode)
                tnode = cnode.ToNode{k};
                tnode.FromNode = unionc(tnode.FromNode,{cnode});
            end
        end
    end
end
```

Update Links

```
function updatejoinlink(task,loc)
%   Update link of the deleted service indexed by loc
    if ~strcmp(class(task),'Task')
        error('The first argument must be type task');
    end
    if isempty(task.ServiceJoinF)
        return;
    end
    % Start delete operation
    s = task.ServiceJoinF{loc}; % Indexing the to be deleted node by s
    % Delete links to the loc of si.ToNode
    [x,y] = ismemberc(task.si.ToNode,{s});
    if x
        if length(task.si.ToNode) == 1
            task.si.ToNode = {};
        else
            task.si.ToNode(y) = [];
        end
    end
    % The remaining element
    for i=1:loc-1
        [x,y] = ismemberc(task.ServiceJoinF{i}.ToNode,{s});
        if x
            if length(task.ServiceJoinF{i}.ToNode) == 1
                task.ServiceJoinF{i}.ToNode = {};
            else
                task.ServiceJoinF{i}.ToNode(y) = [];
            end
        end
    end
    % Delete the node from task.ServiceJoinF, referred by loc
    if length(task.ServiceJoinF) == 1
        task.ServiceJoinF = {};
    else
        task.ServiceJoinF(loc) = [];
    end
end
```

Update Service Split

```
function  updateservicesplit(task,swn)
%   Update link of service split of a task in the swn
    if ~strcmp(class(task),'Task')
        error('The input must be type task');
    end
    if ~strcmp(class(swn),'Workflow')
        error('The forth argument must be type workflow');
    end
    task.ServiceSplit = {};     % Destroy ServiceSplit
    % Collecting all connected spliting services from the task
    for i=1:length(task.TaskSplit)
        task.ServiceSplit =
unionc(task.ServiceSplit,task.TaskSplit{i}.ServiceJoin);
    end
    % Relocating services
    task.ServiceSplit = servicerelocation(task.ServiceSplit,swn);
    % Destroy current elements of nodes in ServiceSplitF
    for i=1:length(task.ServiceSplitF)
        delete(task.ServiceSplitF{i});
    end
    task.ServiceSplitF = {};
    % Create a new set of nodes according to task.ServiceSplit
    for i=1:length(task.ServiceSplit)
        s1 = task.ServiceSplit{i};
        task.ServiceSplitF =
unionc(task.ServiceSplitF,{Node(s1.Name,@s1.Prop)});
    end
    % Destroy the first element's links so
    task.so.ToNode = {};
    %Bulid all links
    for i=1:length(task.TaskSplit)
        % Build links of the first node so
        for j=1:length(task.TaskSplit{i}.si.ToNode)
            [o,loc] =
ismemberc(task.ServiceSplitF,task.TaskSplit{i}.si.ToNode(j));
            task.so.ToNode =
unionc(task.so.ToNode,task.ServiceSplitF(loc));
        end
        % Build links of the rest
        t1 = task.TaskSplit{i};
        for j=1:length(t1.ServiceJoinF)
            n1 = t1.ServiceJoinF{j};
            [o,loc] = ismemberc(task.ServiceSplitF,{n1});
            cn = task.ServiceSplitF{loc};
            for k=1:length(n1.ToNode)
                if strcmp(n1.ToNode{k}.Name,'Ti')
                    cn.ToNode = unionc(cn.ToNode,{task.To});
                else
                    [o,loc] = ismemberc(task.ServiceSplitF,n1.ToNode(k));
                    cn.ToNode =
unionc(cn.ToNode,task.ServiceSplitF(loc));
                end
            end
        end
end
```

F.4 Compliance Checking

TS Satisfiability

```
function result = tssat(dformula,swn,origin,srv)
    if ~strcmp(class(dformula),'Direction')
        error('The first arguement must be type direction');
    end
    if ~strcmp(class(origin),'Task')
        error('The second arguement must be type service');
    end
    if ~ismemberc(swn.Tasks,{origin})
        error('Task (third arguement) must be in the workflow');
    end
    if ~isempty(srv)
        if ~strcmp(class(srv),'Service')
            error('The forth arguement must be type service');
        end
        if ~ismemberc(origin.ServiceJoin,{srv})
            error('Service is not part of the task');
        end
    end
    %  Create swn according to originating service
    ttskjoin = origin.TaskJoin;
    tsktemp = {};
    for i=1:length(origin.TaskJoin)
        tsktemp = horzcat(tsktemp,{origin.TaskJoin{i}.TaskSplit});
        origin.TaskJoin{i}.TaskSplit = {};
    end
    if ~isempty(srv)
        tskjoin = intersectc(srv.TaskJoin,ttskjoin);
    else
        tskjoin = ttskjoin;
    end
    for i=1:length(tskjoin)
        tskjoin{i}.TaskSplit = {origin};
    end
    origin.TaskJoin = tskjoin;
    result = directionsat(dformula,swn,origin,srv,1); %  Round 1
    if result == 's'    %  Round 2
        result = directionsat(dformula,swn,origin,srv,2);
        if  result =='u'
            result = 'p';
        end
    else  %  Round 3 when result is unsat
        result = directionsat(dformula,swn,origin,srv,3);
        if result == 's'
            result = 'p';
        end
    end
    %  Restore swn
    origin.TaskJoin = ttskjoin;
    for i=1:length(origin.TaskJoin)
        origin.TaskJoin{i}.TaskSplit = tsktemp{i};
    end
end
```

F.4.1 Composite Algorithms

Composite Satisfiability

```
function states = compositesat(cformula,swn,r)
    if ~strcmp(class(cformula),'Composite')
        error('There is no Composite formula');
    end
    % Clear exisitng cformula
    current = AP(cformula.Name,'');
    for i=1:length(swn.Tasks)
        swn.Tasks{i}.Atoms = setdiffc(swn.Tasks{i}.Atoms,{current});
    end
    %  Call composite engine
    states = compositeengine(cformula,swn,r);
end
```

Composite Engine

```
function states = compositeengine(cformula,swn,r)
%compositeengine Summary of this function goes here
    states = {};
    % If ceiling
    if strcmp(cformula.Name,'true')
        if isempty(cformula.Task)
            states = swn.Tasks;
        else
            states = {cformula.Task};
        end
        return;
    end
     %  If flooring
    if strcmp(cformula.Name, 'false')
        states = {};
        return;
    end
    %   Check of cformula is atom or not and get result
    if ~isempty(cformula.Task)
        if cformula.isAtom()
            result = compositecheck1(cformula.Task,cformula);
        else  %  If cformula is not Atom
            result = compositecheck2(cformula.Task,cformula);
        end
        if r == 2
            if result == 's'
                states = {cformula.Task};
                return;
            else
                states = {};
                return;
            end
        else  %  r == 1 or 3
            % Do not mark any state when result is unsatisfied
            if result =='u'
                states = {};
                return;
            else
```

```
                states = {cformula.Task};
                return;
            end
        end
    else
        for i=1:length(swn.Tasks)
            cformula.Task = swn.Tasks{i};
            tstates = compositeengine(cformula,swn,r);
            states = unionc(states,tstates);
        end
        cformula.Task = [];
    end
end
```

1st Class Algorithm

```
function result = compositecheck1(task,cformula)
    if cformula.Q =='E'
        result = forsomesat(task,cformula);
    else    %  In the case of cformula.Q = A
        result = forallsat(task,cformula);
    end

end
```

1st Class For Some Satisfiability

```
function  result = forsomesat(task, cformula)
    V = {}; % A set of nodes that statisfy any of cformula.cloves for
this task
    if isempty(cformula.Cloves) && isempty(cformula.Ncloves)
        result = 's';
        return;
    end
    % Check for service join
     if cformula.D =='P'
        V = task.ServiceJoinF;
        c = 0;
        dummy = 0;
        ndummy = 0;

        if ~isempty(cformula.Cloves)
           if cformula.Cloves{1} =='d'
               dummy = 1;
               cformula.Cloves = cformula.Task.ServiceJoin{1}.Prop(1);
           end
        end
        if ~isempty(cformula.Ncloves)
            if cformula.Ncloves{1} == 'd'
                ndummy = 1;
                cformula.Ncloves = cformula.Task.ServiceJoin{1}.Prop(1);
            end
        end
        for i=1:length(V)
            x = ismember(V{i}.Prop(),cformula.Cloves);
            if any(x==1)
                V{i}.mark = 0;
```

```
                c = c+1;
            else
                y = ismember(cformula.Ncloves,V{i}.Prop());
                if any(y==0)
                    V{i}.mark = 0;
                    c = c+1;
                end
            end
        end
         % Non reachable paths from unsat or all are marked with sat
        if (length(V) == c) || (~reachable(task.si,task.Ti,V))
            result = 's';
        elseif c > 0
             result = 'p';
        else
             result = 'u';
        end
         if dummy
            cformula.Cloves = {'d'};
        end
        if ndummy
            cformula.Ncloves = {'d'};
        end
        % Set mark
        for i=1:length(V)
            V{i}.mark = 1;
        end
    % Check for service split
    elseif cformula.D =='F'
        V = task.ServiceSplitF;
        c = 0;
        dummy = 0;
        ndummy = 0;
        if ~isempty(cformula.Cloves)
           if cformula.Cloves{1} == 'd'
               dummy = 1;
               cformula.Cloves = cformula.Task.ServiceSplit{1}.Prop(1);
           end
        end
        if ~isempty(cformula.Ncloves)
            if cformula.Nclves{1} == 'd'
                ndummy = 1;
                cformula.Ncloves = cformula.Task.ServiceSplit{1}.Prop(1);
            end
        end
        for i=1:length(V)
            x = ismember(V{i}.Prop(),cformula.Cloves);
            if any(x==1)
                V{i}.mark = 0;
                c = c+1;
            else
                y = ismember(cformula.Ncloves,V{i}.Prop());
                if any(y==0)
                    V{i}.mark = 0;
                    c = c+1;
                end
```

```
            end
        end
        % Non reachable paths from unsat or all are marked with sat
        if (length(V) == c) || (~reachable(task.so,task.To,V))
             result = 's';
        elseif c > 0
             result = 'p';
        else
             result = 'u';
        end
        if dummy
            cformula.Cloves = {'d'};
        end
        if ndummy
            cformula.Ncloves = {'d'};
        end
                % Set mark
        for i=1:length(V)
            V{i}.mark = 1;
        end
     end
end
```

1st Class For All Satisfiability

```
function  result = forallsat(task, cformula)
    V = {}; % A set of nodes that statisfy any of cformula.cloves for
this task
        if isempty(cformula.Cloves) && isempty(cformula.Ncloves)
        result = 's';
        return;
    end
    % Check for service join
    if cformula.D =='P'
        V = task.ServiceJoinF;
        c = 0;
        dummy = 0;
        ndummy = 0;

        if ~isempty(cformula.Cloves)
           if cformula.Cloves{1} == 'd'
               dummy = 1;
               cformula.Cloves = cformula.Task.ServiceJoin{1}.Prop(1);
           end
        end
        if ~isempty(cformula.Ncloves)
            if cformula.Ncloves{1} == 'd'
                cformula.Ncloves = cformula.Task.ServiceJoin{1}.Prop(1);
                ndummy = 1;
            end
        end
        % Clear all mark
        for i=1:length(V)
            V{i}.mark = 0;
        end
        for i=1:length(V)
            x = ismember(V{i}.Prop(),cformula.Cloves);
```

```
            if any(x==1)
                V{i}.mark = 1;
                c = c+1;
            else
                y = ismember(cformula.Ncloves,V{i}.Prop());
                if any(y==0)
                    V{i}.mark = 1;
                    c = c+1;
                end
            end
        end
        % Non reachable paths from unsat or all are marked with sat
        if length(V) == c
             result = 's';
        elseif reachable(task.si,task.Ti,V)
             result = 'p';
        else
             result = 'u';
        end
        % Reset mark
        for i=1:length(V)
            V{i}.mark = 1;
        end
        if dummy
            cformula.Cloves = {'d'};
        end
        if ndummy
            cformula.Ncloves = {'d'};
        end
    % Check for service split
    elseif cformula.D =='F'
        V = task.ServiceSplitF;
        c = 0;
        dummy = 0;
        ndummy = 0;
        if ~isempty(cformula.Cloves)
           if cformula.Cloves{1} == 'd'
               dummy = 1;
               cformula.Cloves = cformula.Task.ServiceSplit{1}.Prop(1);
           end
        end
        if ~isempty(cformula.Ncloves)
            if cformula.Nclves{1} == 'd'
                ndummy = 1;
                cformula.Ncloves = cformula.Task.ServiceSplit{1}.Prop(1);
            end
        end
        % Clear all mark
        for i=1:length(V)
            V{i}.mark = 0;
        end
        for i=1:length(V)
            x = ismember(V{i}.Prop(),cformula.Cloves);
            if any(x==1)
                V{i}.mark = 1;
                c = c+1;
            else
```

```
                y = ismember(cformula.Ncloves,V{i}.Prop());
                if any(y==0)
                    V{i}.mark = 1;
                    c = c+1;
                end
            end
        end
        % Non reachable paths from unsat or all are marked with sat
        if length(V) == c
             result = 's';
        elseif reachable(task.si,task.Ti,V)
             result = 'p';
        else
             result = 'u';
        end
        if dummy
            cformula.Cloves = {'d'};
        end
        if ndummy
            cformula.Ncloves = {'d'};
        end
        % Reset mark
        for i=1:length(V)
            V{i}.mark = 1;
        end
    end
end
```

Reachable

```
function output = reachable(start,T,nodes)
    output = 0;
    if ~(strcmp(class(start),'Node') && (strcmp(class(T),'Node')))
        error('Both first and second arguments must be type node');
    end
    if ~iscell(nodes)
        error('The third argument must be type cell of nodes');
    end
    if isempty(start.ToNode) || isempty(nodes)
        output = 0;
        return;
    end
    if ~isempty(nodes)
        if ~strcmp(class(nodes{1}), 'Node')
            error('The third argument must be type cell of nodes');
        end
    end
    % rn is immediate reachable nodes in each iteration, VV is a set of
    % already reachable nodes
    VV = {};
    rn = {};
    V = {};
    % Extract only nodes with mark 1
    for i=1:length(start.ToNode)
        if start.ToNode{i}.mark
            rn = unionc(rn,{start.ToNode{i}});
        end
```

```
    end
    for i=1:length(nodes)
        if nodes{i}.mark
            V = unionc(V,nodes(i));
        end
    end
    while ~isempty(V)
        if isempty(rn) % If there are no other reachable nodes
            output = 0;
            return;
        end
        VV = rn; % Swallow reachable nodes
        if ismemberc(VV,{T}) % Check if it reaches to T
            output = 1;
            return;
        end
        rn = {};
        % Update rn
        for i=1:length(VV)
            for j=1:length(VV{i}.ToNode)
                if VV{i}.ToNode{j}.mark
                    rn = unionc(rn,{VV{i}.ToNode{j}});
                end
            end
        end
        if ismemberc(rn,{T}) % Check if it reaches to T
            output = 1;
            return;
        end
        rn = setdiffc(rn,VV);
        V = setdiffc(V,VV);
    end
end
```

2nd Class Algorithm

```
function result = compositecheck2(task,cformula)
    if cformula.D =='P'
        result =
extractpathcheck(task.si,cformula,cell(1,length(task.ServiceJoin)+1),0);
    else    %  In the case of cformula.D = F
        result =
extractpathcheck(task.so,cformula,cell(1,length(task.ServiceSplit)+1),0);
    end
end
```

Path Extraction

```
function result = extractpathcheck(s,cformula,path,inx)
    result = '';
    if isempty(s.ToNode)
        result = logicalcheck(cformula,path,inx);
        return;
    end
    inx = inx+1;
    path{inx} = s;
    for i=1:length(s.ToNode)
```

```
        result1 = extractpathcheck(s.ToNode{i},cformula,path,inx);
        result = comresult(result1,result);
        %new
        if result == 'p'
            return;
        end
    end
end
```

Logical Check

```
function result = logicalcheck(cformula,path,inx)
    result = '';
    if isempty(cformula)
        return;
    end
    if cformula.isAtom()
        if cformula.Q =='E'
            result = forsomesat2(cformula,path,inx);
            return;
        else
            result = forallsat2(cformula,path,inx);
            return;
        end
    end
    result1 = logicalcheck(cformula.Lnode,path,inx);
    result2 = logicalcheck(cformula.Rnode,path,inx);
    %  Check for each operator encountering
    switch cformula.Element
        case '&'
            if (result1 == 's') && (result2 == 's');
                result = 's';
            else
                result = 'u';
            end
        case '||'
            if (result1 == 'u') && (result2 == 'u');
                result = 'u';
            else
                result = 's';
            end
        case '!'
            if (result1 == 'u')
                result = 's';
            else
                result = 'u';
            end
        otherwise
            error('Composite formula is invalid');
    end
end
```

2nd Class For Some Satisfiability

```
function result = forsomesat2(cformula,path,inx)
    path(1) = [];
    inx = inx-1;
    V = path;
    if isempty(cformula.Cloves) && isempty(cformula.Ncloves)
        result = 's';
        return;
    end
    if cformula.D == 'P'
        if ~isempty(cformula.Cloves)
           if cformula.Cloves{1} == 'd'
               cformula.Cloves = cformula.Task.ServiceJoin{1}.Prop(1);
           end
        end
        if ~isempty(cformula.Ncloves)
            if cformula.Ncloves{1} == 'd'
                cformula.Ncloves = cformula.Task.ServiceJoin{1}.Prop(1);
            end
        end
    end
    if cformula.D == 'F'
        if ~isempty(cformula.Cloves)
           if cformula.Cloves{1} == 'd'
               if inx == 1
                   result = 'u';
                   return;
               end
               cformula.Cloves = cformula.Task.ServiceSplit{1}.Prop(1);
           end
        end
        if ~isempty(cformula.Ncloves)
            if cformula.Nclves{1} == 'd'
                if inx == 1
                   result = 'u';
                   return;
               end
                cformula.Ncloves = cformula.Task.ServiceSplit{1}.Prop(1);
            end
        end
    end
    if ~isempty(cformula.Cloves)
        for i=1:inx
            vex = V{i}.Prop();
            for j=1:length(vex)
                for k=1:length(cformula.Cloves)
                    if strcmp(vex{j},cformula.Cloves{k})
                        result = 's';
                        return;
                    end
                end
            end
            if ~isempty(cformula.Ncloves)
                for j=1:length(cformula.Ncloves)
                    hav = 0;
                    vex = V{i}.Prop();
                    for k=1:vex
                        if strcmp(vex{j},cformula.Ncloves{j});
```

```
                            hav = 1;
                            break;
                        end
                    end
                    if hav == 0
                        result = 's';
                        return;
                    end
                end
            end
        end
    end
    result = 'u';
end
```

2nd Class For All Satisfiability

```
function result = forallsat2(cformula,path,inx)
    path(1) = [];
    inx = inx-1;
    V = path;
    if isempty(cformula.Cloves) && isempty(cformula.Ncloves)
        result = 's';
        return;
    end
    if cformula.D == 'P'
        if ~isempty(cformula.Cloves)
           if cformula.Cloves{1} == 'd'
               if inx == 1
                   result = 'u';
                   return;
               end
               cformula.Cloves = cformula.Task.ServiceJoin{1}.Prop(1);
           end
        end
        if ~isempty(cformula.Ncloves)
            if cformula.Ncloves{1} == 'd'
                if inx == 1
                   result = 'u';
                   return;
               end
                cformula.Ncloves = cformula.Task.ServiceJoin{1}.Prop(1);
            end
        end
    end
    if cformula.D == 'F'
        if ~isempty(cformula.Cloves)
           if cformula.Cloves{1} == 'd'
               cformula.Cloves = cformula.Task.ServiceSplit{1}.Prop(1);
           end
        end
        if ~isempty(cformula.Ncloves)
            if cformula.Ncloves{1} == 'd'
                cformula.Ncloves = cformula.Task.ServiceSplit{1}.Prop(1);
            end
        end
    end
```

```
     if ~isempty(cformula.Cloves)
        for i=1:inx
            vex = V{i}.Prop();
            hav = 0;
            for j=1:length(vex)
                for k=1:length(cformula.Cloves)
                    if strcmp(vex{j},cformula.Cloves{k})
                        hav = 1;
                        break;
                    end
                end
                if hav == 1
                    break;
                end
            end
            if hav == 0
                %  do it for Ncloves
                if ~isempty(cformula.Ncloves)
                    pass = 0;
                    for j=1:length(cformula.Ncloves)
                        vex = V{i}.Prop();
                        havn = 0;
                        for k=1:vex
                            if strcmp(vex{k},cformula.Nclove{j})
                                havn = 1;
                                break;
                            end
                        end
                        if havn == 0
                            pass = 1;
                            break;
                        end
                    end
                    if pass == 0
                        result = 'u';
                        return;
                    end
                else
                    result = 'u';
                    return;
                end
            end
        end
    end
    result = 's';
end
```

Composite Result

```
function result = comresult(r1,r2)
    if isempty(r1)
        result = r2;
        return;
    end
    if isempty(r2)
        result = r1;
        return;
```

```
    end
    if r1 == 'u'
        if r2 ~= 'u'
            result = 'p';
        else
            result = 'u';
        end
    elseif r1 == 's'
        if r2 == 's'
            result = 's';
        else
            result = 'p';
        end
    elseif r1 == 'p'
        result = 'p';
    else
        result = r2;
    end
end
```

F.4.2 Path Algorithms

Path Satisfiability

```
function states = pathsat(pformula,d,swn,r)
    % Start Operation
    states = {};
    if isempty(pformula)
        return;
    end
    %  Reaching a leaf node (composite node)
    if strcmp(class(pformula),'Composite')
        states = compositesat(pformula,swn,r);
        return;
    end
    %  Recursively traversing until checking all path nodes
    states1 = pathsat(pformula.Lnode,d,swn,r);
    states2 = pathsat(pformula.Rnode,d,swn,r);
    %  Check for each operator encountering
    switch pformula.Element
        % Contain no element (use when path operator is not presented)
        case ''
            states = states1;
        %  Propositional operators
        case '&'
            states = intersectc(states1,states2);
        case '|'
            states = unionc(states1,states2);
        case '!'
            states = states1;
            states = setdiffc(swn.Tasks,states);
        case 'ift'   %  !states1 | states2
            nstates1 = setdiffc(swn.Tasks,states1);
            states = unionc(nstates1,states2);
        case 'ifnif' % (!states1 | states2) & (states1 | !states2)
            nstates1 = setdiffc(swn.tasks,states1);
```

```
            nstates2 = setdiffc(swn.tasks,states2);
            part1 = unionc(nstates1,states2);
            part2 = unionc(states1,nstates2);
            states = intersectc(part1,part2);
        %  Temporal operators
        case 'AX'
            if r == 3
                states = satex(states1,d,swn,pformula.Task);
            else
                states = satax(states1,d,swn,pformula.Task);
            end
        case 'EX'
            states = satex(states1,d,swn,pformula.Task);
        case 'AU'   %  AU(states1,states2) = ~(EU[~states2,(~states1 &
~states2)] | EG(~states2))
            if r == 3
                states = sateu(states1,states2,d,swn,pformula.Task);
            else
                nstates1 = setdiffc(swn.Tasks,states1);     %  ~states1
                nstates2 = setdiffc(swn.Tasks,states2);     %  ~states2
                cstates1 = intersectc(nstates1,nstates2);   %  (~states1
& ~states2)
                cstates1 = sateu(nstates2,cstates1,d,swn,pformula.Task);
%  EU[~states2,(~states1 & ~states2)]
                cstates2 = sataf(states2,d,swn,pformula.Task);   %
EG(~states2)
                cstates2 = setdiffc(swn.Tasks,cstates2);         %
EG(~states2)
                states = unionc(cstates1,cstates2);   %
EU[~states2,(~states1 & ~states2)] | EG(~states2)
                states = setdiffc(swn.Tasks,states);  %
~(EU[~states2,(~states1 & ~states2)] | EG(~states2))
            end
        case 'EU'
            states = sateu(states1,states2,d,swn,pformula.Task);
        case 'EF'  %  EF(states1) = EU(T,states1)
            states = sateu(swn.Tasks,states1,d,swn,pformula.Task);  %
EU(T,states1)
        case 'EG'  %  EG(states1) = ~AF(~states1)
            nstates1 = setdiffc(swn.Tasks,states1);        %  ~states1
            states = sataf(nstates1,d,swn,pformula.Task);  %
AF(~states1)
            states = setdiffc(swn.Tasks,states);          %
~AF(~states1)
        case 'AF'
            if r == 3
                states = sateu(swn.Tasks,states1,d,swn,pformula.Task);  %
EU(T,states1)
            else
                states = sataf(states1,d,swn,pformula.Task);
            end

        case 'AG'  %  AG(states1) = ~EF(~states1)
            if r == 3
                nstates1 = setdiffc(swn.Tasks,states1);  %  ~states1
```

```
                states = sataf(nstates1,d,swn,pformula.Task);   %
AF(~states1)
                states = setdiffc(swn.Tasks,states);           %
~AF(~states1)
            else
                nstates1 = setdiffc(swn.Tasks,states1);  %  ~states1
                states = sateu(swn.Tasks,nstates1,d,swn,pformula.Task);
%  EF(~states1) = EU(T,~states1)
                states = setdiffc(swn.Tasks,states);    %  ~EF(~states1)
            end
        otherwise
            error('Path formula is invalid');
    end
end
```

SATAF

```
function states = sataf(cstates,d,swn,task)
    X = swn.Tasks;
    Y = cstates;
    while ~isequalc(X,Y)
        X = Y;
        Z = prea(Y,d,swn);
        Y = unionc(Y,Z);
    end
    states = Y;
    if ~isempty(task)
        if ~ismemberc(swn.Tasks,{task})
            error('All tasks must be in a workflow');
        end
        states = intersectc(states,{task});
    end
end
```

SATAX

```
function states = satax(cstates,d,swn,task)
if ~isempty(task)
        states = prea(cstates,d,swn);
        states = intersectc(states,{task});
    else
        states = prea(cstates,d,swn);
    end
end
```

SATEU

```
function states = sateu(cstates1,cstates2,d,swn,task)
    states = {};
    W = cstates1;
    X = swn.Tasks;
    Y = cstates2;
    if isempty(Y)
        return;
    end
    while ~isequalc(X,Y)
        X = Y;
```

```
        V = pree(Y,d,swn);
        V = intersectc(V,W);
        Y = unionc(Y,V);
    end
    states = Y;
    if ~isempty(task)
         states = intersectc(states,{task});
    end
end
```

SATEX

```
function states = satex(cstates,d,swn,task)
    if ~isempty(task)
        if ~ismemberc(swn.Tasks,{task})
            error('All tasks must be in a workflow');
        end
        states = pree(cstates,d,swn);
        states = intersectc(states,{task});
    else
        states = pree(cstates,d,swn);

    end
end
```

PREE

```
function states = pree(cstates,d,swn)
    states = {};
    if d == 'H'
        for i=1:length(cstates)
            states = unionc(states,cstates{i}.TaskJoin);
        end
        return;
    end
    if d == 'B'
        for i=1:length(cstates)
            states = unionc(states,cstates{i}.TaskSplit);
        end
    end
end
```

PREA

```
function states = prea(cstates,d,swn)
    states = setdiffc(swn.Tasks,cstates);
    states = pree(states,d,swn);
    a=setdiffc(swn.Tasks,states);
    states = setdiffc(swn.Tasks,states);
    if d == 'B'
        states = setdiffc(states,{swn.Start});
    else
        states = setdiffc(states,{swn.End});
    end
end
```

F.4.1 Direction Algorithms

Direction Satisfiability

```
function result = directionsat(dformula,swn,origin,srv,r)
    if isempty(dformula)
        result = '';
        return;
    end
    %  Reaching a leaf node (composite node)
    if strcmp(class(dformula),'Path')
        states = pathsat(dformula,dformula.D,swn,r);
        if strncmp(dformula.Element,'A',1) && (r ~= 3)
            result = pathsatresult(states,dformula,origin,srv,'A');
        else
            result = pathsatresult(states,dformula,origin,srv,'E');
        end
        return;
    end
    %  Recursively traversing until checking all path nodes
    result1 = directionsat(dformula.Dlnode,swn,origin,srv,r);
    result2 = directionsat(dformula.Drnode,swn,origin,srv,r);
      %  Check for each operator encountering
    switch dformula.Element
         % Contain no element (use when path operator is not presented)
        case ''
           result = result1;
        %  Propositional operators
        case '&'
            if (result1 == 's') && (result2 == 's')
                result = 's';
            else
                result = 'u';
            end
        case '|'
            if (result1 == 'u') && (result2 == 'u')
                result = 'u';
            else
                result = 's';
            end
        case '!'
            if result1 == 'u'
                result = 's';
            else
                result = 'u';
            end
        case 'ift'   %  !result1 | result2
            if (result1 == 'u') || (result2 == 's')
                result = 's';
            end
        case 'ifnif' % (!result1 | result2) & (result1 | !result2)
            if result1 == result2
                result = 's';
            else
                result = 'u';
            end
        otherwise
```

```
            error('Direction formula is invalid');
    end
end
```

Result for Path Satifiability

```
function result = pathsatresult(states,dformula,origin,srv,t)
    if isempty(srv)
         if ismemberc(states,{origin})
                result = 's';
         else
                result = 'u';
         end
         return;
    end
    if t == 'A'
        if dformula.D == 'B'
            if isempty(dformula.Task)
               taskjoin = intersectc(srv.TaskJoin,origin.TaskJoin);
               for i=1:length(taskjoin)
                    if ~ismemberc(states,taskjoin(i))
                        result = 'u';
                        return;
                    end
                end
                    result = 's';
            else
                if ismemberc(states,{dformula.Task})
                    result = 's';
                else
                    result = 'u';
                end
            end
        else  %  In case of H
            if ismemberc(states,{origin})
                result = 's';
            else
                result = 'u';
            end
        end
    else  %  In case of E
        if dformula.D == 'B'
            if isempty(dformula.Task)
                taskjoin = intersectc(srv.TaskJoin,origin.TaskJoin);
                for i=1:length(taskjoin)
                    if ismemberc(states,taskjoin(i))
                        result = 's';
                        return;
                    end
                end
                result = 'u';
            else
                if ismemberc(states,{dformula.Task})
                    result = 's';
                else
                    result = 'u';
                end
```

```
            end
        else  %  In case of H
            if ismemberc(states,{origin})
                result = 's';
            else
                result = 'u';
            end
        end
    end
end
```

F.5 Initialisation

Initialising System

```
InitializingTask;
TravelAgent = Workflow(Start,End);
AssociatingTask;
InitializingService;
AssociatingService;
% correct the special node D3 and End
TravelAgent.Services = unionc(TravelAgent.Services,{D3});
addservicesplit({D3},End,TravelAgent);
InitializingTSFormula;
buildmatrix(TravelAgent)
```

Initialising Services

```
% Initialize Services
AG = Service('AG',{'AG' '0.8' 'BHR' 'SSL'});
AV = Service('AV',{'AV' '0.8'});
BBC = Service('BBC',{'BBC' 'SSL'});
BM = Service('BM',{'BM' 'SSL'});
BO = Service('BO',{'BO' '0.8' 'BHR' 'SSL'});
BI = Service('BI',{'BI'});
DM = Service('DM',{'DM'});
EZ = Service('EZ',{'EZ' 'BAR'});
EB = Service('EB',{'EB' '0.8' 'BAR' 'BHR'});
ES = Service('ES',{'ES'});
EX = Service('EX',{'EX' '0.8' 'BAR' 'BHR' 'USA'});
GM = Service('GM',{'GM' 'SSL'});
MS = Service('MS',{'MS' 'PAYMENT' 'SSL'});
PP = Service('PP',{'PP' 'PAYMENT' 'SSL'});
PL = Service('PL',{'PL' '0.8' 'BAR' 'BHR' 'USA'});
RA = Service('RA',{'RA' 'BAR'});
TI = Service('TI',{'TI'});
TD1 = Service('TD1',{'TD1' 'OWNER'});
TD2 = Service('TD2',{'TD2' 'OWNER'});
TP = Service('TP',{'TP'});
VS = Service('VS',{'VS' 'PAYMENT' 'SSL'});
WHT = Service('WHT',{'WHT' 'SSL'});
WC = Service('WC',{'WC'});
WE = Service('WE',{'WE' 'CLEV'});
WI = Service('WI',{'WI'});
WG = Service('WG',{'WG'});
WR = Service('WR',{'WR'});
WT = Service('WT',{'WT'});
```

```
D1 = Service('D1',{'D1' 'OWNER'});
D2 = Service('D2',{'D2' 'OWNER'});
D3 = Service('D3',{'D3' 'OWNER'});
E1 = Service('E1',{'E1' 'CLEV'});
E2 = Service('E2',{'E2'});
R1 = Service('R1',{'R1'});
R2 = Service('R2',{'R2'});
G1 = Service('G1',{'G1'});
```

Initialising Tasks

```
% Tasks Creation
Start = Task('Start');
FlightSearching = Task('FlightSearching');
HotelSearching = Task('HotelSearching');
CarSearching = Task('CarSearching');
CruiseSearching = Task('CruiseSearching');
TrainSearching = Task('TrainSearching');
PromotionPack = Task('PromotionPack');
HotelRating = Task('HotelRating');
Location = Task('Location');
WeatherForecast = Task('WeatherForecast');
FlightReservation = Task('FlightReservation');
HotelReservation = Task('HotelReservation');
CarReservation = Task('CarReservation');
CruiseReservation = Task('CruiseReservation');
TrainReservation = Task('TrainReservation');
FlightBooking = Task('FlightBooking');
HotelBooking = Task('HotelBooking');
CarBooking = Task('CarBooking');
CruiseBooking = Task('CruiseBooking');
TrainBooking = Task('TrainBooking');
Events = Task('Events');
Restaurants = Task('Restaurants');
Insurance = Task('Insurance');
GuideandLocalTour = Task('GuideandLocalTour');
Payment = Task('Payment');
TravellingDocs = Task('TravellingDocs');
TouristInfo = Task('TouristInfo');
Statistics = Task('Statistics');
End = Task('End');
```

Associating Services

```
% Associating Services
    % Start
    servicejointask({D1},Start,{},TravelAgent);
    % *Searching
    servicejointask({EX},FlightSearching,{Start},TravelAgent);
    servicejointask({PL},FlightSearching,{Start},TravelAgent);
    servicejointask({EX EB},FlightSearching,{Start},TravelAgent);
    servicejointask({EX EB RA},FlightSearching,{Start},TravelAgent);
    servicejointask({EX EB EZ},FlightSearching,{Start},TravelAgent);
    servicejointask({EX EB EZ RA},FlightSearching,{Start},TravelAgent);
    servicejointask({EX EZ},FlightSearching,{Start},TravelAgent);
    servicejointask({EX EZ RA},FlightSearching,{Start},TravelAgent);
    servicejointask({EX RA},FlightSearching,{Start},TravelAgent);
```

```
    servicejointask({PL EB},FlightSearching,{Start},TravelAgent);
    servicejointask({PL EB RA},FlightSearching,{Start},TravelAgent);
    servicejointask({PL EB EZ},FlightSearching,{Start},TravelAgent);
    servicejointask({PL EB EZ RA},FlightSearching,{Start},TravelAgent);
    servicejointask({PL EZ},FlightSearching,{Start},TravelAgent);
    servicejointask({PL EZ RA},FlightSearching,{Start},TravelAgent);
    servicejointask({PL RA},FlightSearching,{Start},TravelAgent);
    servicejointask({EX PL},FlightSearching,{Start},TravelAgent);
    servicejointask({EX PL EB},FlightSearching,{Start},TravelAgent);
    servicejointask({EX PL EB RA},FlightSearching,{Start},TravelAgent);
    servicejointask({EX PL EB EZ},FlightSearching,{Start},TravelAgent);
    servicejointask({EX PL EB EZ
RA},FlightSearching,{Start},TravelAgent);
    servicejointask({EX PL EZ},FlightSearching,{Start},TravelAgent);
    servicejointask({EX PL EZ RA},FlightSearching,{Start},TravelAgent);
    servicejointask({EX PL RA},FlightSearching,{Start},TravelAgent);
    servicejointask({EX},HotelSearching,{Start},TravelAgent);
    servicejointask({PL},HotelSearching,{Start},TravelAgent);
    servicejointask({EB},HotelSearching,{Start},TravelAgent);
    servicejointask({AG},HotelSearching,{Start},TravelAgent);
    servicejointask({BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EX PL},HotelSearching,{Start},TravelAgent);
    servicejointask({EX EB},HotelSearching,{Start},TravelAgent);
    servicejointask({EX AG},HotelSearching,{Start},TravelAgent);
    servicejointask({EX BO},HotelSearching,{Start},TravelAgent);
    servicejointask({PL EB},HotelSearching,{Start},TravelAgent);
    servicejointask({PL AG},HotelSearching,{Start},TravelAgent);
    servicejointask({PL BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EB AG},HotelSearching,{Start},TravelAgent);
    servicejointask({EB BO},HotelSearching,{Start},TravelAgent);
    servicejointask({AG BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EX PL EB},HotelSearching,{Start},TravelAgent);
    servicejointask({EX PL AG},HotelSearching,{Start},TravelAgent);
    servicejointask({EX PL BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EX EB AG},HotelSearching,{Start},TravelAgent);
    servicejointask({EX EB BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EX AG BO},HotelSearching,{Start},TravelAgent);
    servicejointask({PL EB AG},HotelSearching,{Start},TravelAgent);
    servicejointask({PL EB BO},HotelSearching,{Start},TravelAgent);
    servicejointask({PL AG BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EB AG BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EX PL EB AG},HotelSearching,{Start},TravelAgent);
    servicejointask({EX PL EB BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EX PL AG BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EX PL EB AG BO},HotelSearching,{Start},TravelAgent);
    servicejointask({EX},CarSearching,{Start},TravelAgent);
    servicejointask({PL},CarSearching,{Start},TravelAgent);
    servicejointask({EX AV},CarSearching,{Start},TravelAgent);
    servicejointask({PL AV},CarSearching,{Start},TravelAgent);
    servicejointask({EX PL AV},CarSearching,{Start},TravelAgent);
    servicejointask({WC},CruiseSearching,{Start},TravelAgent);
    servicejointask({WT},TrainSearching,{Start},TravelAgent);
    servicejointask({WT ES},TrainSearching,{Start},TravelAgent);
    % *Reservation
    servicejointask({EX},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
```

```
    servicejointask({PL},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EB},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EZ},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({RA},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EX PL},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EX EB},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EX EZ},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EX RA},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({PL EB},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({PL EZ},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({PL RA},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EB EZ},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EB RA},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EZ RA},FlightReservation,{FlightSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EX PL EB},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({EX PL EZ},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({EX PL RA},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({EX EB EZ},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({EX EB RA},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({EX EZ RA},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({PL EB EZ},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({PL EB RA},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({PL EZ RA},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({EX PL EB EZ},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({EX PL EB RA},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({EX PL EZ RA},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({PL EB EZ RA},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
    servicejointask({EX PL EB EZ RA},FlightReservation,{FlightSearching
Location WeatherForecast},TravelAgent);
```

```
    servicejointask({EX},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({PL},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EB},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({AG},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX PL},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX EB},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX AG},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({PL EB},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({PL AG},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({PL BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EB AG},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EB BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({AG BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX PL EB},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX PL AG},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX PL BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX EB AG},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX EB BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX AG BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({PL EB AG},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({PL EB BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({PL AG BO},HotelReservation,{HotelSearching Location
WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX PL EB AG},HotelReservation,{HotelSearching
Location WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX PL EB BO},HotelReservation,{HotelSearching
Location WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX PL AG BO},HotelReservation,{HotelSearching
Location WeatherForecast HotelRating},TravelAgent);
    servicejointask({PL EB AG BO},HotelReservation,{HotelSearching
Location WeatherForecast HotelRating},TravelAgent);
```

```
    servicejointask({EX PL EB AG BO},HotelReservation,{HotelSearching
Location WeatherForecast HotelRating},TravelAgent);
    servicejointask({EX},CarReservation,{CarSearching Location
WeatherForecast},TravelAgent);
    servicejointask({PL},CarReservation,{CarSearching Location
WeatherForecast},TravelAgent);
    servicejointask({AV},CarReservation,{CarSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EX PL},CarReservation,{CarSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EX AV},CarReservation,{CarSearching Location
WeatherForecast},TravelAgent);
    servicejointask({PL AV},CarReservation,{CarSearching Location
WeatherForecast},TravelAgent);
    servicejointask({EX PL AV},CarReservation,{CarSearching Location
WeatherForecast},TravelAgent);
    servicejointask({WC},CruiseReservation,{CruiseSearching Location
WeatherForecast},TravelAgent);
    servicejointask({WT},TrainReservation,{TrainSearching Location
WeatherForecast},TravelAgent);
    servicejointask({ES},TrainReservation,{TrainSearching Location
WeatherForecast},TravelAgent);
    servicejointask({WT ES},TrainReservation,{TrainSearching Location
WeatherForecast},TravelAgent);
     % Location WeatherForecast HotelRating
    servicejointask({GM},Location,{FlightSearching HotelSearching
CarSearching CruiseSearching TrainSearching},TravelAgent);
    servicejointask({BM},Location,{FlightSearching HotelSearching
CarSearching CruiseSearching TrainSearching},TravelAgent);
    servicejointask({BBC},WeatherForecast,{FlightSearching HotelSearching
CarSearching CruiseSearching TrainSearching},TravelAgent);
    servicejointask({WHT},WeatherForecast,{FlightSearching HotelSearching
CarSearching CruiseSearching TrainSearching},TravelAgent);
    servicejointask({AG},HotelRating,{HotelSearching},TravelAgent);
    servicejointask({BO},HotelRating,{HotelSearching},TravelAgent);
    servicejointask({AG BO},HotelRating,{HotelSearching},TravelAgent);
    % *Booking
    servicejointask({EX},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({PL},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EB},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EZ},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX
PL},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX
EB},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX
EZ},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({PL
EB},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({PL
EZ},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({PL
RA},FlightBooking,{FlightReservation},TravelAgent);
```

```
    servicejointask({EB
EZ},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EB
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EZ
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX PL
EB},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX PL
EZ},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX PL
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX EB
EZ},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX EB
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX EZ
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({PL EB
EZ},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({PL EB
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({PL EZ
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX PL EB
EZ},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX PL EB
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX PL EZ
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({PL EB EZ
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX PL EB EZ
RA},FlightBooking,{FlightReservation},TravelAgent);
    servicejointask({EX},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({PL},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EB},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({AG},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX PL},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX EB},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX AG},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({PL EB},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({PL AG},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({PL BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EB AG},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EB BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({AG BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX PL
EB},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX PL
AG},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX PL
BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX EB
AG},HotelBooking,{HotelReservation},TravelAgent);
```

```
    servicejointask({EX EB
BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX AG
BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({PL EB
AG},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({PL EB
BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({PL AG
BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX PL EB
AG},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX PL EB
BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX PL AG
BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({PL EB AG
BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX PL EB AG
BO},HotelBooking,{HotelReservation},TravelAgent);
    servicejointask({EX},CarBooking,{CarReservation},TravelAgent);
    servicejointask({PL},CarBooking,{CarReservation},TravelAgent);
    servicejointask({AV},CarBooking,{CarReservation},TravelAgent);
    servicejointask({EX PL},CarBooking,{CarReservation},TravelAgent);
    servicejointask({EX AV},CarBooking,{CarReservation},TravelAgent);
    servicejointask({PL AV},CarBooking,{CarReservation},TravelAgent);
    servicejointask({EX PL AV},CarBooking,{CarReservation},TravelAgent);
    servicejointask({WC},CruiseBooking,{CruiseReservation},TravelAgent);
    servicejointask({WT},TrainBooking,{TrainReservation},TravelAgent);
    servicejointask({ES},TrainBooking,{TrainReservation},TravelAgent);
    servicejointask({WT ES},TrainBooking,{TrainReservation},TravelAgent);
    % Payment
    servicejointask({VS},Payment,{FlightBooking HotelBooking CarBooking
CruiseBooking TrainBooking Events Restaurants GuideandLocalTour Insurance
PromotionPack},TravelAgent);
    servicejointask({MS},Payment,{FlightBooking HotelBooking CarBooking
CruiseBooking TrainBooking Events Restaurants GuideandLocalTour Insurance
PromotionPack},TravelAgent);
    servicejointask({PP},Payment,{FlightBooking HotelBooking CarBooking
CruiseBooking TrainBooking Events Restaurants GuideandLocalTour Insurance
PromotionPack},TravelAgent);
        % Events Restaurants GuideandLocalTour Insurance
    servicejointask({WE},Events,{FlightBooking HotelBooking CarBooking
CruiseBooking TrainBooking},TravelAgent);
    servicejointask({E1},Events,{FlightBooking HotelBooking CarBooking
CruiseBooking TrainBooking},TravelAgent);
    servicejointask({E2},Events,{FlightBooking HotelBooking CarBooking
CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WE E1},Events,{FlightBooking HotelBooking CarBooking
CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WE E2},Events,{FlightBooking HotelBooking CarBooking
CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WE E1 E2},Events,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WR},Restaurants,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
```

```
    servicejointask({R1},Restaurants,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
    servicejointask({R2},Restaurants,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WR R1},Restaurants,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WR R2},Restaurants,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WR R1 R2},Restaurants,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WG},GuideandLocalTour,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
    servicejointask({G1},GuideandLocalTour,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WG G1},GuideandLocalTour,{FlightBooking HotelBooking
CarBooking CruiseBooking TrainBooking},TravelAgent);
    servicejointask({WI},Insurance,{FlightBooking HotelBooking CarBooking
CruiseBooking TrainBooking},TravelAgent);
    servicejointask({AV},Insurance,{CarBooking},TravelAgent);
    servicejointask({EZ},Insurance,{FlightBooking},TravelAgent);
    servicejointask({WI AV},Insurance,{},TravelAgent);
    servicejointask({WI EZ},Insurance,{},TravelAgent);
    servicejointask({AV EZ},Insurance,{},TravelAgent);
    servicejointask({WI AV EZ},Insurance,{},TravelAgent);
    % PromotionPack
    servicejointask({EX},PromotionPack,{Start},TravelAgent);
    servicejointask({PL},PromotionPack,{Start},TravelAgent);
    servicejointask({TP},PromotionPack,{Start},TravelAgent);
    % TravellingDocs TouristInfo Statistics and End
    servicejointask({TD1},TravellingDocs,{Payment},TravelAgent);
    servicejointask({TD2},TravellingDocs,{Payment},TravelAgent);
    servicejointask({TI},TouristInfo,{TravellingDocs},TravelAgent);
    servicejointask({DM},Statistics,{TravellingDocs},TravelAgent);
    servicejointask({BI},Statistics,{TravellingDocs},TravelAgent);
    servicejointask({D2},End,{TravellingDocs TouristInfo
Statistics},TravelAgent);
```

Associating Tasks

```
% Associating Tasks
    %Start join 0, split 6
    tasksplittask(Start,FlightSearching,TravelAgent);
    tasksplittask(Start,HotelSearching,TravelAgent);
    tasksplittask(Start,CarSearching,TravelAgent);
    tasksplittask(Start,CruiseSearching,TravelAgent);
    tasksplittask(Start,TrainSearching,TravelAgent);
    tasksplittask(Start,PromotionPack,TravelAgent);
    %FlightSearching join 1, split 3
    taskjointask(FlightSearching,Start,TravelAgent);
    tasksplittask(FlightSearching,FlightReservation,TravelAgent);
    tasksplittask(FlightSearching,Location,TravelAgent);
    tasksplittask(FlightSearching,WeatherForecast,TravelAgent);
    %HotelSearching join 1, split 4
    taskjointask(HotelSearching,Start,TravelAgent);
    tasksplittask(HotelSearching,HotelReservation,TravelAgent);
    tasksplittask(HotelSearching,Location,TravelAgent);
    tasksplittask(HotelSearching,WeatherForecast,TravelAgent);
```

```
    tasksplittask(HotelSearching,HotelRating,TravelAgent);
    %CarSearching join 1, split 3
    taskjointask(CarSearching,Start,TravelAgent);
    tasksplittask(CarSearching,CarReservation,TravelAgent);
    tasksplittask(CarSearching,Location,TravelAgent);
    tasksplittask(CarSearching,WeatherForecast,TravelAgent);
    %CruiseSearching join 1, split 3
    taskjointask(CruiseSearching,Start,TravelAgent);
    tasksplittask(CruiseSearching,CruiseReservation,TravelAgent);
    tasksplittask(CruiseSearching,Location,TravelAgent);
    tasksplittask(CruiseSearching,WeatherForecast,TravelAgent);
    %TrainSearching join 1, split 3
    taskjointask(TrainSearching,Start,TravelAgent);
    tasksplittask(TrainSearching,TrainReservation,TravelAgent);
    tasksplittask(TrainSearching,Location,TravelAgent);
    tasksplittask(TrainSearching,WeatherForecast,TravelAgent);
    %Location join 5, split 5
    taskjointask(Location,FlightSearching,TravelAgent);
    taskjointask(Location,HotelSearching,TravelAgent);
    taskjointask(Location,CarSearching,TravelAgent);
    taskjointask(Location,CruiseSearching,TravelAgent);
    taskjointask(Location,TrainSearching,TravelAgent);
    tasksplittask(Location,FlightReservation,TravelAgent);
    tasksplittask(Location,HotelReservation,TravelAgent);
    tasksplittask(Location,CarReservation,TravelAgent);
    tasksplittask(Location,CruiseReservation,TravelAgent);
    tasksplittask(Location,TrainReservation,TravelAgent);
    %WeatherForecast join 5, split 5
    taskjointask(WeatherForecast,FlightSearching,TravelAgent);
    taskjointask(WeatherForecast,HotelSearching,TravelAgent);
    taskjointask(WeatherForecast,CarSearching,TravelAgent);
    taskjointask(WeatherForecast,CruiseSearching,TravelAgent);
    taskjointask(WeatherForecast,TrainSearching,TravelAgent);
    tasksplittask(WeatherForecast,FlightReservation,TravelAgent);
    tasksplittask(WeatherForecast,HotelReservation,TravelAgent);
    tasksplittask(WeatherForecast,CarReservation,TravelAgent);
    tasksplittask(WeatherForecast,CruiseReservation,TravelAgent);
    tasksplittask(WeatherForecast,TrainReservation,TravelAgent);
    %HotelRating join 1, split 1
    taskjointask(HotelRating,HotelSearching,TravelAgent);
    tasksplittask(HotelRating,HotelReservation,TravelAgent);
    %FlightReservation join 3, split 1
    taskjointask(FlightReservation,FlightSearching,TravelAgent);
    taskjointask(FlightReservation,Location,TravelAgent);
    taskjointask(FlightReservation,WeatherForecast,TravelAgent);
    tasksplittask(FlightReservation,FlightBooking,TravelAgent);
    %HotelReservation join 4, split 1
    taskjointask(HotelReservation,HotelSearching,TravelAgent);
    taskjointask(HotelReservation,Location,TravelAgent);
    taskjointask(HotelReservation,WeatherForecast,TravelAgent);
    taskjointask(HotelReservation,HotelRating,TravelAgent);
    tasksplittask(HotelReservation,HotelBooking,TravelAgent);
    %CarReservation join 3, split 1
    taskjointask(CarReservation,CarSearching,TravelAgent);
    taskjointask(CarReservation,Location,TravelAgent);
    taskjointask(CarReservation,WeatherForecast,TravelAgent);
    tasksplittask(CarReservation,CarBooking,TravelAgent);
```

```
%CruiseReservation join 3, split 1
taskjointask(CruiseReservation,CruiseSearching,TravelAgent);
taskjointask(CruiseReservation,Location,TravelAgent);
taskjointask(CruiseReservation,WeatherForecast,TravelAgent);
tasksplittask(CruiseReservation,CruiseBooking,TravelAgent);
%TrainReservation join 3, split 1
taskjointask(TrainReservation,TrainSearching,TravelAgent);
taskjointask(TrainReservation,Location,TravelAgent);
taskjointask(TrainReservation,WeatherForecast,TravelAgent);
tasksplittask(TrainReservation,TrainBooking,TravelAgent);
%FlightBooking join 1, split 5
taskjointask(FlightBooking,FlightReservation,TravelAgent);
tasksplittask(FlightBooking,Payment,TravelAgent);
tasksplittask(FlightBooking,Events,TravelAgent);
tasksplittask(FlightBooking,Restaurants,TravelAgent);
tasksplittask(FlightBooking,GuideandLocalTour,TravelAgent);
tasksplittask(FlightBooking,Insurance,TravelAgent);
%HotelBooking join 1, split 5
taskjointask(HotelBooking,HotelReservation,TravelAgent);
tasksplittask(HotelBooking,Payment,TravelAgent);
tasksplittask(HotelBooking,Events,TravelAgent);
tasksplittask(HotelBooking,Restaurants,TravelAgent);
tasksplittask(HotelBooking,GuideandLocalTour,TravelAgent);
tasksplittask(HotelBooking,Insurance,TravelAgent);
%CarBooking join 1, split 5
taskjointask(CarBooking,CarReservation,TravelAgent);
tasksplittask(CarBooking,Payment,TravelAgent);
tasksplittask(CarBooking,Events,TravelAgent);
tasksplittask(CarBooking,Restaurants,TravelAgent);
tasksplittask(CarBooking,GuideandLocalTour,TravelAgent);
tasksplittask(CarBooking,Insurance,TravelAgent);
%CruiseBooking join 1, split 5
taskjointask(CruiseBooking,CruiseReservation,TravelAgent);
tasksplittask(CruiseBooking,Payment,TravelAgent);
tasksplittask(CruiseBooking,Events,TravelAgent);
tasksplittask(CruiseBooking,Restaurants,TravelAgent);
tasksplittask(CruiseBooking,GuideandLocalTour,TravelAgent);
tasksplittask(CruiseBooking,Insurance,TravelAgent);
%TrainBooking join 1, split 5
taskjointask(TrainBooking,TrainReservation,TravelAgent);
tasksplittask(TrainBooking,Payment,TravelAgent);
tasksplittask(TrainBooking,Events,TravelAgent);
tasksplittask(TrainBooking,Restaurants,TravelAgent);
tasksplittask(TrainBooking,GuideandLocalTour,TravelAgent);
tasksplittask(TrainBooking,Insurance,TravelAgent);
%PromotionPack join 1, split 1
taskjointask(PromotionPack,Start,TravelAgent);
tasksplittask(PromotionPack,Payment,TravelAgent);
%Events join 5, split 1
taskjointask(Events,FlightBooking,TravelAgent);
taskjointask(Events,HotelBooking,TravelAgent);
taskjointask(Events,CarBooking,TravelAgent);
taskjointask(Events,CruiseBooking,TravelAgent);
taskjointask(Events,TrainBooking,TravelAgent);
tasksplittask(Events,Payment,TravelAgent);
%Restaurants join 5, split 1
taskjointask(Restaurants,FlightBooking,TravelAgent);
```

```
    taskjointask(Restaurants,HotelBooking,TravelAgent);
    taskjointask(Restaurants,CarBooking,TravelAgent);
    taskjointask(Restaurants,CruiseBooking,TravelAgent);
    taskjointask(Restaurants,TrainBooking,TravelAgent);
    tasksplittask(Restaurants,Payment,TravelAgent);
    %GuideandLocalTour join 5, split 1
    taskjointask(GuideandLocalTour,FlightBooking,TravelAgent);
    taskjointask(GuideandLocalTour,HotelBooking,TravelAgent);
    taskjointask(GuideandLocalTour,CarBooking,TravelAgent);
    taskjointask(GuideandLocalTour,CruiseBooking,TravelAgent);
    taskjointask(GuideandLocalTour,TrainBooking,TravelAgent);
    tasksplittask(GuideandLocalTour,Payment,TravelAgent);
    %Insurance join 5, split 1
    taskjointask(Insurance,FlightBooking,TravelAgent);
    taskjointask(Insurance,HotelBooking,TravelAgent);
    taskjointask(Insurance,CarBooking,TravelAgent);
    taskjointask(Insurance,CruiseBooking,TravelAgent);
    taskjointask(Insurance,TrainBooking,TravelAgent);
    tasksplittask(Insurance,Payment,TravelAgent);
    %Payment join 10, split 1
    taskjointask(Payment,FlightBooking,TravelAgent);
    taskjointask(Payment,HotelBooking,TravelAgent);
    taskjointask(Payment,CarBooking,TravelAgent);
    taskjointask(Payment,CruiseBooking,TravelAgent);
    taskjointask(Payment,TrainBooking,TravelAgent);
    taskjointask(Payment,Events,TravelAgent);
    taskjointask(Payment,Restaurants,TravelAgent);
    taskjointask(Payment,GuideandLocalTour,TravelAgent);
    taskjointask(Payment,Insurance,TravelAgent);
    taskjointask(Payment,PromotionPack,TravelAgent);
    tasksplittask(Payment,TravellingDocs,TravelAgent);
    %TravellingDocs join 1, split 3
    taskjointask(TravellingDocs,Payment,TravelAgent);
    tasksplittask(TravellingDocs,TouristInfo,TravelAgent);
    tasksplittask(TravellingDocs,Statistics,TravelAgent);
    tasksplittask(TravellingDocs,End,TravelAgent);
    %TouristInfo join 1, split 1
    taskjointask(TouristInfo,TravellingDocs,TravelAgent);
    tasksplittask(TouristInfo,End,TravelAgent);
    %Statistics join 1, split 1
    taskjointask(Statistics,TravellingDocs,TravelAgent);
    tasksplittask(Statistics,End,TravelAgent);
    %End join 3
    taskjointask(End,TravellingDocs,TravelAgent);
    taskjointask(End,TouristInfo,TravelAgent);
    taskjointask(End,Statistics,TravelAgent);
```

Initialising TS Formulas

```
% tssat(d0,TravelAgent,Insurrance,PL)
q01 = Composite('q01','&','F','',FlightSearching,{},{});
q02 = Composite('q02','','F','A',FlightSearching,{},{'PL'});
q03 = Composite('q03','','F','E',FlightSearching,{'EX'},{});
```

```
q01.addleft(q02);
q01.addright(q03);
p01 = Path('',[],'H');
p01.addleft(q01);
d0 = Direction('');
d0.addleft(p01);
% Composite formula for Requirement 1
% tssat(d1d1,TravelAgent,FlightSearching,PL)
% tssat(d1d2,TravelAgent,HotelSearching,PL)
% tssat(d1d3,TravelAgent,CarSearching,PL)
q11 = Composite('q11','','P','A',FlightSearching,{},{'EX'});
q12 = Composite('q12','','P','A',HotelSearching,{},{'EX'});
q13 = Composite('q13','','P','A',CarSearching,{},{'EX'});
p11 = Path('',[],'H');
p11.addleft(q11);
p12 = Path('',[],'H');
p12.addleft(q12);
p13 = Path('',[],'H');
p13.addleft(q13);
d1d1 = Direction('');
d1d1.addleft(p11);
d1d2 = Direction('');
d1d2.addleft(p12);
d1d3 = Direction('');
d1d3.addleft(p13);
% Composite formula for Requirement 2
% tssat(d2d1,TravelAgent,Restaurants,'')
q21 = Composite('q21','','P','A',Restaurants,{},{'RedChilly'});
p21 = Path('',[],'H');
p21.addleft(q21);
d2d1 = Direction('');
d2d1.addleft(p21);
% Composite formula for Requirement 3
% tssat(d3d1,TravelAgent,Events,'')
q31 = Composite('q31','','P','A',Events,{'CLEV'},{});
p31 = Path('',[],'H');
p31.addleft(q31);
d3d1 = Direction('');
d3d1.addleft(p31);
% Composite formula for Requirement 4
% tssat(d4d1,TravelAgent,HotelReservation,AG)
% tssat(d4d1,TravelAgent,HotelReservation,BO)
q41 = Composite('q41','','P','E',Payment,{'MS'},{'VS'});
q42 = Composite('q42','','P','E',Payment,{'VS'},{'MS'});
q43 = Composite('q43','&','P','',Payment,{},{});
q44 = Composite('q44','!','P','',Payment,{},{});
q43.addleft(q41);
q43.addright(q42);
q44.addleft(q43);
p41 = Path('AF',[],'H');
p41.addleft(q44);
d4d1 = Direction('');
d4d1.addleft(p41);
% Composite formula for Requirement 5
% tssat(d5d1,TravelAgent,HotelSearching,PL)
q51 = Composite('q51','','P','E',HotelRating,{'d'},{});
q52 = Composite('q52','','P','E',HotelRating,{},{'d'});
```

```
q53 = Composite('q53','&','P','',HotelRating,{},{});
q53.addleft(q51);
q53.addright(q52);
p51 = Path('EF',[],'H');
p51.addleft(q53);
d5d1 = Direction('');
d5d1.addleft(p51);
% For special test
% tssat(dt5,TravelAgent,HotelSearching,PL)
qt51 = Composite('qt51','','P','E',HotelRating,{'d'},{});
pt51 = Path('EF',[],'H');
pt51.addleft(qt51);
dt5 = Direction('');
dt5.addleft(pt51);
% Composite formula for Requirement 6
% tssat(d6d1,TravelAgent,FlightReservation,EX)
% tssat(d6d1,TravelAgent,CarReservation,EX)
% tssat(d6d1,TravelAgent,HotelReservation,EX)
q61 = Composite('q61','','P','A',Insurance,{'WI'},{});
p61 = Path('EF',[],'H');
p61.addleft(q61);
d6d1 = Direction('');
d6d1.addleft(p61);
% Composite formula for Requirement 7
% tssat(d7d1,TravelAgent,CarSearching,'')
% tssat(d7d2,TravelAgent,CarReservation,'')
% tssat(d7d3,TravelAgent,CarBooking,'')
q71 = Composite('q71','','P','A',CarSearching,{'0.8'},{});
q72 = Composite('q72','','P','A',CarReservation,{'0.8'},{});
q73 = Composite('q73','','P','A',CarBooking,{'0.8'},{});
p71 = Path('',[],'H');
p71.addleft(q71);
p72 = Path('',[],'H');
p72.addleft(q72);
p73 = Path('',[],'H');
p73.addleft(q73);
d7d1 = Direction('');
d7d1.addleft(p71);
d7d2 = Direction('');
d7d2.addleft(p72);
d7d3 = Direction('');
d7d3.addleft(p73);
% Composite formula for Requirement 8
% tssat(d8d1,TravelAgent,HotelBooking,BO)
q81 = Composite('q81','','P','A',HotelSearching,{'AG' 'BO'},{});
q82 = Composite('q82','','P','A',HotelReservation,{'AG' 'BO'},{});
p81 = Path('EF',[],'B');
p81.addleft(q81);
p82 = Path('EF',[],'B');
p82.addleft(q82);
d8d1 = Direction('&');
d8d1.addleft(p81);
d8d1.addright(p82);
% Composite formula for Requirement 9
% tssat(d9d1,TravelAgent,HotelBooking,PL)
q91 = Composite('q91','','P','A',HotelSearching,{'AG' 'PL'},{});
p91 = Path('EF',[],'B');
```

```
p91.addleft(q91);
d9d1 = Direction('');
d9d1.addleft(p91);
% Composite formula for Requirement 10
% tssat(d10d1,TravelAgent,Insurance,WI)
q101 = Composite('q101','','P','A',FlightBooking,{'BAR'},{});
q102 = Composite('q102','','P','A',HotelBooking,{'BHR'},{});
q103 = Composite('q103','','P','A',CarBooking,{'USA' 'EU'},{});
p101 = Path('EF',[],'B');
p101.addleft(q101);
p102 = Path('EF',[],'B');
p102.addleft(q102);
p103 = Path('EF',[],'B');
p103.addleft(q103);
d10d1 = Direction('&');
d10d1.addleft(Direction('&'));
d10d1.Dlnode.addleft(p101);
d10d1.Dlnode.addright(p102);
d10d1.addright(p103);
% Composite formula for Requirement 11
% tssat(d11d1,TravelAgent,HotelBooking,BO)
q111 = Composite('q111','','P','A',HotelReservation,{'AG' 'BO'},{});
p111 = Path('AF',[],'B');
p111.addleft(q111);
d11d1 = Direction('');
d11d1.addleft(p111)
% Composite formula for Requirement 12
% tssat(d12d1,TravelAgent,Insurance,EZ)
q121 = Composite('q121','','P','A',FlightReservation,{'EZ'},{});
q122 = Composite('q122','','P','A',FlightBooking,{'EZ'},{});
p121 = Path('AX',FlightBooking,'B');
p122 = Path('',[],'B');
p121.addleft(q121);
p122.addleft(q122);
d12d1 = Direction('&');
d12d1.addleft(p121);
d12d1.addright(p122);
% Composite formula for Requirement 13
% tssat(d13d1,TravelAgent,Insurance,EZ)
q131 = Composite('q131','','P','A',FlightReservation,{'EZ'},{});
q132 = Composite('q132','','P','A',FlightBooking,{'EZ'},{});
q133 = Composite('q133','','P','A',Insurance,{'EZ' 'WI'},{});
p131 = Path('AX',FlightBooking,'B');
p132 = Path('',[],'B');
p133 = Path('',[],'H');
p131.addleft(q131);
p132.addleft(q132);
p133.addleft(q133);
d13d1 = Direction('&');
d13d1.addleft(Direction('&'));
d13d1.Dlnode.addleft(p131);
d13d1.Dlnode.addright(p132);
d13d1.addright(p133);
% Composite formula for Requirement 14
% tssat(d14d1,TravelAgent,HotelBooking,BO)
q141 = Composite('q141','','F','A',HotelBooking,{'BO' 'OWNER'
'PAYMENT'},{});
```

```
p141 = Path('',HotelBooking,'H');
p141.addleft(q141);
d14d1 = Direction('');
d14d1.addleft(p141)
% Composite formula for Requirement 15
% tssat(d15d1,TravelAgent,FlightBooking,'')
% tssat(d15d1,TravelAgent,HotelBooking,'')
% tssat(d15d1,TravelAgent,CarBooking,'')
% tssat(d15d1,TravelAgent,CruiseBooking,'')
% tssat(d15d1,TravelAgent,TrainBooking,'')
q151 = Composite('q151','','P','A',Payment,{},{});
p151 = Path('AF',[],'H');
p151.addleft(q151);
d15d1 = Direction('');
d15d1.addleft(p151)
% Composite formula for Requirement 16
% tssat(d16d1,TravelAgent,TouristInfo,TI)
q161 = Composite('q161','','P','A',Payment,{},{});
p161 = Path('AF',[],'B');
p161.addleft(q161);
d16d1 = Direction('');
d16d1.addleft(p161)
% Composite formula for Requirement 17
% tssat(d17d1,TravelAgent,Payment,VS)
% tssat(d17d1,TravelAgent,Payment,MS)
% tssat(d17d1,TravelAgent,Payment,PP)
q171 = Composite('q171','','P','A',TravellingDocs,{'OWNER'},{});
p171 = Path('AX',[],'H');
p171.addleft(q171);
d17d1 = Direction('');
d17d1.addleft(p171);
% Composite formula for Requirement 18
% tssat(d18d1,TravelAgent,Payment,PP)
% tssat(d18d1,TravelAgent,Payment,MS)
% tssat(d18d1,TravelAgent,Payment,VS)
q181 = Composite('q181','','P','A',FlightBooking,{},{});
q182 = Composite('q182','','P','A',HotelBooking,{},{});
q183 = Composite('q183','','P','A',CarBooking,{},{});
q184 = Composite('q184','','P','A',TrainBooking,{},{});
q185 = Composite('q185','','P','A',CruiseBooking,{},{});
q186 = Composite('q186','','P','A',PromotionPack,{},{});
p181 = Path('|',[],'B');
p181.addleft(Path('|',[],'B'));
p181.addright(Path('|',[],'B'));
p181.Lnode.addleft(Path('|',[],'B'));
p181.Lnode.addright(Path('|',[],'B'));
p181.Lnode.Lnode.addleft(q181);
p181.Lnode.Lnode.addright(q182);
p181.Lnode.Rnode.addleft(q183);
p181.Lnode.Rnode.addright(q184);
p181.Rnode.addleft(q185);
p181.Rnode.addright(q186);
d18d1 = Direction('');
d18d1.addleft(p181)
% Composite formula for Requirement 19
% tssat(d19d1,TravelAgent,HotelSearching,BO)
q191 = Composite('q191','','F','A',[],{'SSL'},{});
```

```
q192 = Composite('q192','','P','A',Payment,{},{});
p191 = Path('AU',[],'H');
p191.addleft(q191);
p191.addright(q192);
d19d1 = Direction('');
d19d1.addleft(p191);
% Composite formula for Requirement 20
% tssat(d20d1,TravelAgent,CarBooking,AV)
q201 = Composite('q201','','F','A',[],{'PAYMENT','OWNER','AV'},{});
p201 = Path('AG',[],'H');
p201.addleft(q201);
d20d1 = Direction('');
d20d1.addleft(p201);
% Composite formula for Requirement 21
% tssat(d21d1,TravelAgent,Insurance,AV)
q211 = Composite('q211','','P','A',[],{'AV'},{});
p211 = Path('',[],'B');
p211.addleft(q211);
d21d1 = Direction('');
d21d1.addleft(p211);
%For evaluation 2
% tssat(dt11,TravelAgent,FlightSearching,'')
qt11 = Composite('qt11','||','P','M',FlightSearching,{},{});
qt12 = Composite('qt12','','P','E',FlightSearching,{'T'},{});
qt13 = Composite('qt13','','P','E',FlightSearching,{'T'},{});
qt11.addleft(qt12);
qt11.addright(qt13);
pt11 = Path('',[],'B');
pt11.addleft(qt11);
dt11 = Direction('');
dt11.addleft(pt11);
% tssat(dt21,TravelAgent,FlightSearching,'')
qt21 = Composite('qt21','','P','E',FlightSearching,{'T50' 'T80'},{});
pt21 = Path('',[],'B');
pt21.addleft(qt21);
dt21 = Direction('');
dt21.addleft(pt21);
```

F.6 Evaluation

Phase 1 (21 Requirements)

```
p=[];
% r1
disp('1');
t = cputime;
tssat(d1d1,TravelAgent,FlightSearching,PL);
tssat(d1d1,TravelAgent,FlightSearching,PL);
tssat(d1d2,TravelAgent,HotelSearching,PL);
tssat(d1d2,TravelAgent,HotelSearching,PL);
tssat(d1d3,TravelAgent,CarSearching,PL);
tssat(d1d3,TravelAgent,CarSearching,PL);
o = cputime;
p = horzcat(p,(o-t)/6);
% r2
disp('2');
```

```
t = cputime;
tssat(d2d1,TravelAgent,Restaurants,'');
tssat(d2d1,TravelAgent,Restaurants,'');
tssat(d2d1,TravelAgent,Restaurants,'');
o = cputime;
p = horzcat(p,(o-t)/3);
% r3
disp('3');
t = cputime;
tssat(d3d1,TravelAgent,Events,'');
tssat(d3d1,TravelAgent,Events,'');
tssat(d3d1,TravelAgent,Events,'');
o = cputime;
p = horzcat(p,(o-t)/3);
% r4
disp('4');
t = cputime;
tssat(d4d1,TravelAgent,HotelReservation,AG);
tssat(d4d1,TravelAgent,HotelReservation,BO);
tssat(d4d1,TravelAgent,HotelReservation,AG);
tssat(d4d1,TravelAgent,HotelReservation,BO);
tssat(d4d1,TravelAgent,HotelReservation,AG);
tssat(d4d1,TravelAgent,HotelReservation,BO);
o = cputime;
p = horzcat(p,(o-t)/6);
% r5
disp('5');
t = cputime;
tssat(d5d1,TravelAgent,HotelSearching,PL);
tssat(d5d1,TravelAgent,HotelSearching,PL);
tssat(d5d1,TravelAgent,HotelSearching,PL);
o = cputime;
p = horzcat(p,(o-t)/3);
% r6
disp('6');
t = cputime;
tssat(d6d1,TravelAgent,FlightReservation,EX);
tssat(d6d1,TravelAgent,CarReservation,EX);
tssat(d6d1,TravelAgent,HotelReservation,EX);
tssat(d6d1,TravelAgent,FlightReservation,EX);
tssat(d6d1,TravelAgent,CarReservation,EX);
tssat(d6d1,TravelAgent,HotelReservation,EX);
o = cputime;
p = horzcat(p,(o-t)/6);
% r7
disp('7');
t = cputime;
tssat(d7d1,TravelAgent,CarSearching,'');
tssat(d7d2,TravelAgent,CarReservation,'');
tssat(d7d3,TravelAgent,CarBooking,'');
tssat(d7d1,TravelAgent,CarSearching,'');
tssat(d7d2,TravelAgent,CarReservation,'');
tssat(d7d3,TravelAgent,CarBooking,'');
o = cputime;
p = horzcat(p,(o-t)/6);
% r8
disp('8');
```

```
t = cputime;
tssat(d8d1,TravelAgent,HotelBooking,BO);
tssat(d8d1,TravelAgent,HotelBooking,BO);
tssat(d8d1,TravelAgent,HotelBooking,BO);
o = cputime;
p = horzcat(p,(o-t)/3);
% r9
disp('9');
t = cputime;
tssat(d9d1,TravelAgent,HotelBooking,PL);
tssat(d9d1,TravelAgent,HotelBooking,PL);
tssat(d9d1,TravelAgent,HotelBooking,PL);
o = cputime;
p = horzcat(p,(o-t)/3);
% r10
disp('10');
t = cputime;
tssat(d10d1,TravelAgent,Insurance,WI);
tssat(d10d1,TravelAgent,Insurance,WI);
tssat(d10d1,TravelAgent,Insurance,WI);
o = cputime;
p = horzcat(p,(o-t)/3);
% r11
disp('11');
t = cputime;
tssat(d11d1,TravelAgent,HotelBooking,BO);
tssat(d11d1,TravelAgent,HotelBooking,BO);
tssat(d11d1,TravelAgent,HotelBooking,BO);
o = cputime;
p = horzcat(p,(o-t)/3);
% r12
disp('12');
t = cputime;
tssat(d12d1,TravelAgent,Insurance,EZ);
tssat(d12d1,TravelAgent,Insurance,EZ);
tssat(d12d1,TravelAgent,Insurance,EZ);
o = cputime;
p = horzcat(p,(o-t)/3);
% r13
disp('13');
t = cputime;
tssat(d13d1,TravelAgent,Insurance,EZ);
tssat(d13d1,TravelAgent,Insurance,EZ);
tssat(d13d1,TravelAgent,Insurance,EZ);
o = cputime;
p = horzcat(p,(o-t)/3);
% r14
disp('14');
t = cputime;
tssat(d14d1,TravelAgent,HotelBooking,BO);
tssat(d14d1,TravelAgent,HotelBooking,BO);
tssat(d14d1,TravelAgent,HotelBooking,BO);
o = cputime;
p = horzcat(p,(o-t)/3);
% r15
disp('15');
t = cputime;
```

```
tssat(d15d1,TravelAgent,FlightBooking,'');
tssat(d15d1,TravelAgent,HotelBooking,'');
tssat(d15d1,TravelAgent,CarBooking,'');
tssat(d15d1,TravelAgent,CruiseBooking,'');
tssat(d15d1,TravelAgent,TrainBooking,'');
tssat(d15d1,TravelAgent,FlightBooking,'');
tssat(d15d1,TravelAgent,HotelBooking,'');
tssat(d15d1,TravelAgent,CarBooking,'');
tssat(d15d1,TravelAgent,CruiseBooking,'');
tssat(d15d1,TravelAgent,TrainBooking,'');
o = cputime;
p = horzcat(p,(o-t)/10);
% r16
disp('16');
t = cputime;
tssat(d16d1,TravelAgent,TouristInfo,TI);
tssat(d16d1,TravelAgent,TouristInfo,TI);
tssat(d16d1,TravelAgent,TouristInfo,TI);
o = cputime;
p = horzcat(p,(o-t)/3);
% r17
disp('17');
t = cputime;
tssat(d17d1,TravelAgent,Payment,VS);
tssat(d17d1,TravelAgent,Payment,MS);
tssat(d17d1,TravelAgent,Payment,PP);
tssat(d17d1,TravelAgent,Payment,VS);
tssat(d17d1,TravelAgent,Payment,MS);
tssat(d17d1,TravelAgent,Payment,PP);
o = cputime;
p = horzcat(p,(o-t)/6);
% r18
disp('18');
t = cputime;
tssat(d18d1,TravelAgent,Payment,PP);
tssat(d18d1,TravelAgent,Payment,MS);
tssat(d18d1,TravelAgent,Payment,VS);
tssat(d18d1,TravelAgent,Payment,PP);
tssat(d18d1,TravelAgent,Payment,MS);
tssat(d18d1,TravelAgent,Payment,VS);
o = cputime;
p = horzcat(p,(o-t)/6);
% r19
disp('19');
t = cputime;
tssat(d19d1,TravelAgent,HotelSearching,BO);
tssat(d19d1,TravelAgent,HotelSearching,BO);
tssat(d19d1,TravelAgent,HotelSearching,BO);
o = cputime;
p = horzcat(p,(o-t)/3);
% r20
disp('20');
t = cputime;
tssat(d20d1,TravelAgent,CarBooking,AV);
tssat(d20d1,TravelAgent,CarBooking,AV);
tssat(d20d1,TravelAgent,CarBooking,AV);
o = cputime;
```

```
p = horzcat(p,(o-t)/3);
% r21
disp('21');
t = cputime;
tssat(d21d1,TravelAgent,Insurance,AV);
tssat(d21d1,TravelAgent,Insurance,AV);
tssat(d21d1,TravelAgent,Insurance,AV);
o = cputime;
p = horzcat(p,(o-t)/3);
```

Phase 2 (with 10, 20, 50, 100, and 1,000 TS Formulas)

```
size = 10; % Ranging on 10, 20, 50, 100, and 1,000
p21=zeros(1,size);
p22=zeros(1,size);
for i=1:size
    i
    % None-atomic
    t = cputime;
    for j=1:i
        %compositecheck2(FlightSearching,qt11);
        tssat(dt11,TravelAgent,FlightSearching,'');
    end
    o = cputime;
    p21(i) = o-t;
    % Atomic
    t = cputime;
    for j=1:i
        %compositecheck1(FlightSearching,qt21);
        tssat(dt21,TravelAgent,FlightSearching,'');
    end
    o = cputime;
    p22(i) = o-t;
end
```

Phase 3 (with 10, 20, 50, and 100 Services)

```
size = 100; % Ranging on 10, 20, 50, and 100
p31 = zeros(1,size);
p32 = zeros(1,size);
for i=1:100
    i
    t = cputime;
    for j=1:i
       tssat(dt11,TravelAgent,FlightSearching,'');
    end
    o = cputime;
    p31(i) = (o-t)*i;
    t = cputime;
    for j=1:i
       tssat(dt21,TravelAgent,FlightSearching,'');
    end
    o = cputime;
    p32(i) = (o-t)*i;
end
```