Formal approaches to multi-resource sharing scheduling

Mahya Rahimi

N°d'ordre NNT : 2017LYSEI129

THESE de DOCTORAT DE L'UNIVERSITE DE LYON

opérée au sein de

L'Institut national des sciences appliquées de Lyon

Ecole Doctorale N°160

Electronique, Electrotechnique, Automatique

Spécialité/ discipline de doctorat:

Automatique

Soutenue publiquement clos le 08/12/2017, par:

Mahya RAHIMI

Formal Approaches to Multi-Resource Sharing Scheduling

Devant le jury composé de :

LAHAYE, Sébastien	Professeur à l'Université d'Angers	Rapporteur
DEMONGODIN, Isabel	Professeur à l'Université de Marseille Nord	Rapporteur
PAWLEWSKI, Pawel	Professeur à l'Université de technologie de Poznań	Rapporteur
PETIN, Jean-François	Professeur à l'Université de Lorraine	Examinateur
DOLGUI, Alexandre	Professeur à IMT Atlantique	Examinateur
DUMITRESCU, Emil	Maître de conférences à l'INSA de Lyon	Examinateur
NIEL, Eric	Professeur à l'INSA de Lyon	Directeur de thèse

Abstract

The objective of scheduling problems is to find the optimal performing sequence for a set of tasks by respecting predefined constraints and optimizing a cost: time, energy, etc. Despite classical approaches, automata models are expressive and also robust against changes in the parameter setting and against changes in the problem specification. Besides, few studies have used formal verification approaches for addressing scheduling problems; yet none of them considered challenging and practical issues such as multi-resource sharing aspect, uncontrollable environment and reaching the optimal schedule in a reasonable time for industrializing the model.

The main objective of this thesis is to propose an efficient modeling and solving approach for the scheduling problem, considering multi-resource sharing and potential uncertainty in occurrence of certain events.

For this purpose, after an introduction in Chapter 1, Chapter 2 addresses the problem of scheduling through a visual, expressive and formal modeling approach, based on weighted automata and the theory of timed automata. The originality of the proposed approach lies in ability of handling the sharing of multiple resources and proposing an efficient solving approach. The proposed models have the advantage of being directly exploitable by means of formal verification tools. The results are obtained using the UPPAAL tool. To solve the problem, an algorithm is developed based on iterating reachability analysis to obtain sub-optimal makespan. Results show the proposed model and solving approach provides a very promising complexity on the class of studied problems and can be applied to industrial cases. In Chapter 3, a synchronous composition of weighted automata is proposed to solve the scheduling problem by performing an optimal reachability analysis directly on the weighted automata models. In the fourth chapter, various uncontrollable behaviors such as the start time, the duration of the task and the failure occurrence in a scheduling problem are modeled by timed game automata. Then, the problem is solved by performing an optimal strategy synthesis over time in TIGA as a synthesis tool.

Keywords: *Discrete event systems; Formal verification; Control synthesis; Timed automata; Weighted automata; Scheduling problem; Makespan; Multi-resource sharing; Uncontrollability*

Résumé

L'objectif principal de cette thèse est de proposer une approche efficace de modélisation et de résolution pour le problème d'ordonnancement, en mettant l'accent sur le partage multi-ressources et sur l'incertitude potentielle d'occurrence de certains événements.

L'ordonnancement a pour objectif de réaliser un ensemble de tâches à la fois en respectant des contraintes prédéfinies et en optimisant le temps. Ce travail s'intéresse en particulier à la minimisation du temps total d'exécution. La plupart des approches existantes préconisent une modélisation mathématique exprimant des équations et des contraintes pour décrire et résoudre des problèmes d'ordonnancement. De telles démarches ont une complexité inhérente. Cependant dans l'industrie, la tâche de planification est récurrente et peut requérir des changements fréquents des contraintes. Outre cela, la prise en compte d'événements incertains est peu supportée par les approches existantes; cela peut toutefois augmenter la robustesse d'un ordonnancement.

Pour répondre à ces problématiques, après une introduction, le chapitre 2 aborde le problème de l'ordonnancement à travers une démarche de modélisation visuelle, expressive et formelle, s'appuyant sur les automates pondérés et sur la théorie des automates temporisés. L'originalité des modèles proposés réside aussi dans leur capacité de décrire le partage de ressources multiples et proposer une approche de résolution efficace. Ces modèles ont l'avantage d'être directement exploitables par des outils de vérification formelle, à travers une démarche de preuve par contradiction vis-à-vis de l'existence d'une solution. Les résultats effectifs sont obtenus grâce à l'outil UPPAAL. La complexité inhérente à la production d'une solution optimale est abordée à travers un algorithme de recherche et d'amélioration itérative de solutions, offrant une complexité très prometteuse sur la classe de problèmes étudiés. Dans le chapitre 3, une composition synchrone est d'automates pondérés est proposée dans le but de résoudre le problème d'ordonnancement en effectuant une analyse d'atteignabilité optimale directement sur les modèles automates pondérés. Dans le quatrième chapitre, divers comportements incontrôlables tels que le temps de début, la durée de la tâche et l'occurrence d'échec dans un problème d'ordonnancement sont modélisés par des automates de jeu temporisés. Ensuite, le problème est résolu en effectuant une synthèse de stratégie optimale dans le temps dans l'outil de synthèse TIGA.

Mots-clés: *Systèmes a événements discrets; Vérification formelle; Synthèse de contrôleur; Automates temporisés; Automates pondérés; Problème d'ordonnancement; Makespan; Partage multi-ressources; Incontrollabilité*

To my beloved husband, Mohammad, for his constant love, sacrifice, and continuous support,

To my parents who give me true love, motivation, care, pray and everything for my life,

To my sister and brother who give me spirit and color in my life.

Acknowledgment

I wish to express my deepest gratitude to my supervisors Prof. Eric Niel and Dr. Emil Dumitrescu. Their advice, encouragement, support, and mentorship have been immensely valuable. I am very thankful for their genuine concern, efforts, and patience towards my development from a student to a researcher.

I am also grateful to members of my jury, Prof. Lahaye, Prof. Demongodin, Prof. Pawlewski, Prof. Pétin and Prof. Dolgui. Their insights, time and feedback are invaluable.

Special thanks are also to all faculties and staff members of Ampère laboratory for their constant academic interactions with me.

Finally, I must express my gratitude to Mohammad, my husband, for his incredible support and encouragement, which has helped me through many challenges. I can only hope to pay him back in the years to come. I am immensely indebted to my mother, father, sister and brother who experienced all of the ups and downs of my life.

Table of contents

1 GENERAL INTRODUCTION 2

1.1 INTRODUCTION 2
1.2 STATE OF THE ART 4
1.2.1 State of the art on general scheduling problems 4
1.2.2 State of the art on scheduling problems modeled by automata 7
1.2.3 State of the art on synchronous composition of weighted automata 8
1.2.4 State of the art on MRS scheduling considering uncontrollable environment 9
1.2.5 Synthesis of the state of the art 10
1.3 RESEARCH QUESTION 14
1.4 CONTRIBUTION 14
1.4.1 Chapter 2: MRS scheduling through translation of weighted to timed automata 14
1.4.2 Chapter 3: Multi-resource sharing scheduling by using synchronous composition of weighted automata 14
1.4.3 Chapter 4: Multi-resource sharing scheduling considering uncontrollable environment 15
1.5 BACKGROUND 15
1.5.1 Formal verification (or model checking) 15
1.5.2 Property-specification language 16
1.5.3 The branching time logic TCTL 16
1.5.4 TCTL properties 17
1.5.5 Techniques for constructing reachability graph of systems 18
1.5.6 Digraph traversal algorithms 20
1.6 CONCLUSION 21

2 MULTI-RESOURCE SHARING SCHEDULING THROUGH TRANSLATION OF WEIGHTED TO TIMED AUTOMATA 23

2.1 INTRODUCTION 23
2.1.1 State of the art 23
2.1.2 Synthesis of the state of the art 26
2.2 MRS SCHEDULING PROBLEM DESCRIPTION 27
2.3 MODELING MRS SCHEDULING PROBLEM BY WEIGHTED AUTOMATA 28
2.3.1 General principle of modeling procedure 28
2.3.2 Problem statement by weighted automata 29
2.4 SOLVING MRS SCHEDULING PROBLEM BY MEANS OF TRANSLATING WEIGHTED AUTOMATA MODELS INTO TIMED AUTOMATA MODELS 37
2.4.1 Translating transitions of the WA model to TA 39
2.4.2 Translating WA models to TA models 42
2.4.3 Scheduling approach 47
2.4.4 Complexity 51
2.5 CONCLUSION 56

3 SYNCHRONOUS COMPOSITION OF WEIGHTED AUTOMATA - APPLICATION TO MRS SCHEDULING 59

3.1 INTRODUCTION 59
3.1.1 State of the art on synchronous composition of weighted automata 59
3.1.2 State of the art on time-optimal reachability analysis 61
3.1.3 Synthesis of the state of the art 61
3.2 SYNCHRONOUS COMPOSITION FOR WEIGHTED AUTOMATA 63
3.2.1 Example 2 63
3.2.2 Algorithmic steps to reach synchronous composition 65
3.2.3 Synchronous composition of DSDA weighted automata 72
3.2.4 A simple example of synchronous composition (Example 2-continue) 73
3.3 FINDING THE OPTIMAL SCHEDULE 81
3.4 CONCLUSION 82

4 MULTI-RESOURCE SHARING SCHEDULING CONSIDERING UNCONTROLLABLE ENVIRONMENT 84

4.1 INTRODUCTION 84
4.1.1 State of the art 84
4.1.2 Synthesis of the state of the art 88
4.2 BACKGROUND 91
4.2.1 Timed Game Automata 91
4.2.2 Safety and reachability games 92
4.2.3 Winning games 93
4.2.4 Strategy 94
4.2.5 Synthesis tool TIGA 94
4.2.6 Winning/losing conditions 95
4.2.7 Partially Cooperative Games 95
4.2.8 Time Optimal Strategy Synthesis 96
4.2.9 Example of timed game automata 96
4.2.10 On-the-fly algorithm for timed games 97
4.2.11 Interval weighted automata 98
4.3 PROBLEM DESCRIPTION 99
4.4 DIFFERENT TYPES OF UNCONTROLLABLE PARAMETERS 100
4.5 MODELING THE SCHEDULING PROBLEM THROUGH TGA CONSIDERING UNCONTROLLABLE PARAMETERS 101
4.6 EXAMPLE 3 112
4.7 SOLVING APPROACH 118
4.8 EXAMPLE 3 (CONTINUE) 119
4.9 CONCLUSION 121

5 CONCLUSION AND PERSPECTIVES 123

5.1 GENERAL CONCLUSION 123
5.2 GENERAL PERSPECTIVES 125
5.3 RELATED PUBLICATION 126

REFERENCES 127

APPENDIX A: IMPLEMENTATION DETAILS OF THE MODEL 134

APPENDIX B: JAVA CODE FOR DEPTH-FIRST SEARCH IN A DIGRAPH 137

APPENDIX C: ABBREVIATIONS 138

List of tables

TABLE 1.1. CLASSIFICATION OF LITERATURE ABOUT SCHEDULING PROBLEMS 12
TABLE 2.1. RESOURCE ASSIGNMENT DETAILS OF EXAMPLE 1 28
TABLE 2.2. CALCULATED AND REAL MAKESPAN AND CORRESPONDING COMPUTATIONAL TIME FOR PROBLEM INSTANCES 51
TABLE 3.1. CLASSIFICATION OF LITERATURE PROPOSING SYNCHRONOUS COMPOSITION OF WA 63
TABLE 4.1. CLASSIFICATION OF LITERATURE CONCERNING UNCONTROLLABILITY 89
TABLE 4.2. TIMING INFORMATION OF TASKS IN EXAMPLE 3 113
TABLE 4.3. RESOURCE ASSIGNMENT DETAILS OF EXAMPLE 3 113
TABLE 4.4. REPAIRING DURATION OF RESOURCES 113

List of figures

FIGURE 2.1. MODELING PATTERN OF WEIGHTED ME AUTOMATON 31
FIGURE 2.2. CORRESPONDING ME AUTOMATA FOR EXAMPLE 1 31
FIGURE 2.3. MODELING PATTERN OF WEIGHTED TL AUTOMATON FOR $k = 1$ 33
FIGURE 2.4. SIMPLIFIED MODELING PATTERN OF WEIGHTED TL AUTOMATON FOR K=1 33
FIGURE 2.5. TL AUTOMATA IN EXAMPLE 1 33
FIGURE 2.6. MODELING PATTERN OF A DELAY PR AUTOMATA (A) AND A TRIGGERING PR AUTOMATA (B) 34
FIGURE 2.7. WEIGHTED TRIGGERING PR AUTOMATA IN EXAMPLE 1 34
FIGURE 2.8. WEIGHTED DELAY PR AUTOMATA IN EXAMPLE 1 35
FIGURE 2.9. GLOBAL BEHAVIOR OF THE EXAMPLE 1(G1||G2||G3||G4||G5) 35
FIGURE 2.10. GLOBAL BEHAVIOR OF EXAMPLE 1 CONSIDERING PR AUTOMATA (G1||G2||G3||G4||G5||G6) 36
FIGURE 2.11. WA MODELS OF EXAMPLE 1 36
FIGURE 2.12. TRANSLATION OF A TRANSITION WITH DURATION FROM A SPECIFICATION WA TO TA 40
FIGURE 2.13. TRANSLATION OF A TRIGGERING TRANSITION FROM A SPECIFICATION WA TO TA 41
FIGURE 2.14. TRANSLATION OF A TRANSITION FROM A PLANT WA TO TA 41
FIGURE 2.15. MODELING PATTERN OF TIMED ME AUTOMATON 43
FIGURE 2.16. TIMED ME AUTOMATON IN EXAMPLE 1 43
FIGURE 2.17. MODELING PATTERN OF TIMED TL AUTOMATON 44
FIGURE 2.18.TIMED TL AUTOMATA OF TASKS A, B AND C IN EXAMPLE 1. 44
FIGURE 2.19. MODELING PATTERN OF TRIGGERING PRECEDENCE TIMED AUTOMATON 44
FIGURE 2.20. TRIGGERING PRECEDENCE TIMED AUTOMATON IN EXAMPLE 1 45
FIGURE 2.21. MODELING PATTERN OF A DELAY PRECEDENCE TIMED AUTOMATON 45
FIGURE 2.22. DELAY PRECEDENCE TIMED AUTOMATON IN EXAMPLE 1 45
FIGURE 2.23. TA MODELS OF EXAMPLE 1 46
FIGURE 2.24. GANTT CHART OF EXAMPLE 1 WITHOUT PRECEDENCE CONDITIONS 50
FIGURE 2.25. GANTT CHART OF EXAMPLE 1 CONSIDERING DELAY PR TIMED AUTOMATON G7 51
FIGURE 2.26. CALCULATION TIME VS. NUMBER OF TASKS FOR INDIVIDUAL TASKS AND COMMON TASKS BETWEEN 2 ME AUTOMATA 54
FIGURE 2.27. CALCULATION TIME VS. NUMBER OF ME AUTOMATA, FOR TASKS THAT ARE INDIVIDUAL OR COMMON TASKS BETWEEN 2 ME AUTOMATA 55
FIGURE 2.28. CALCULATION TIME VS. NUMBER OF TASKS FOR 5 ME AUTOMATA 56
FIGURE 3.1. WA MODEL OF EXAMPLE 2. 64
FIGURE 3.2. GANTT CHART OF AUTOMATA $G1$ TO $G6$ ACCORDING TO A SAMPLE SCHEDULE 65
FIGURE 3.3. SYNCHRONOUS COMPOSITION OF AUTOMATA IN EXAMPLE 2 80
FIGURE 4.1. EXAMPLES OF WINNING (A) AND LOSING (B) TIMED GAME AUTOMATA (BEHRMANN ET AL. 2007A) 93
FIGURE 4.2. EXAMPLES OF FORCED TRANSITION: WINNING MODEL (A), EQUIVALENT MODEL WITH THE IMPLICIT TRANSITION MADE EXPLICIT (B) AND A LOSING MODEL (C) (BEHRMANN ET AL. 2007A) 93
FIGURE 4.3. EXAMPLE OF SYNCHRONIZATION WITH FORCED TRANSITIONS: ORIGINAL MODEL (A), FORCED TRANSITION MADE EXPLICIT (B), COMPLETE COMPOSITION WITH EXPLICIT FORCED TRANSITION (BEHRMANN ET AL. 2007A) 94
FIGURE 4.4. TIMED GAME AUTOMATON EXAMPLE 97
FIGURE 4.5. MODELING PATTERN OF A TASK IN A ME AUTOMATON 102
FIGURE 4.6. MODELING PATTERN OF A RESOURCE FAILURE IN ITS IDLE TIME IN A ME AUTOMATON 103
FIGURE 4.7. MODELING PATTERN OF A TASK IN A ME AUTOMATON SUBJECT TO RESOURCE FAILURE 104

FIGURE 4.8. MODELING PATTERN OF A ME AUTOMATON SUBJECT TO RESOURCE FAILURE 105
FIGURE 4.9. MODELING PATTERN OF A TASK WITH UNCERTAIN DURATION IN A ME AUTOMATON..... 106
FIGURE 4.10. MODELING PATTERN OF A ME AUTOMATON CONSISTING OF TASKS WITH UNCERTAIN DURATIONS 106
FIGURE 4.11. MODELING PATTERN OF A TASK WITH UNCERTAIN START TIME IN A ME AUTOMATON 107
FIGURE 4.12. MODELING PATTERN OF A TASK LAUNCHER AUTOMATON 108
FIGURE 4.13. MODELING PATTERN OF A LAUNCHER AUTOMATON FOR A TASK SUBJECT TO FAILURE 109
FIGURE 4.14. MODELING PATTERN OF A TASK LAUNCHER AUTOMATON FOR A TASK WITH NON-DETERMINISTIC START TIME WHEN THE RESOURCES ARE OCCUPIED FROM THE INSTANT OF EXECUTION..... 110
FIGURE 4.15. MODELING PATTERN OF A TASK LAUNCHER AUTOMATON FOR A TASK WITH NON-DETERMINISTIC START TIME WHEN THE RESOURCES ARE OCCUPIED FROM THE RELEASE TIME OF TASK..... 110
FIGURE 4.16. MODELING PATTERN OF A TRIGGERING PR AUTOMATA WITH RELIABLE RESOURCES..... 111
FIGURE 4.17. MODELING PATTERN OF A TRIGGERING PR AUTOMATA WITH TASKS SUBJECT TO FAILURE 111
FIGURE 4.18. MODELING PATTERN OF A DELAY PRAUTOMATA WITH RELIABLE RESOURCES 112
FIGURE 4.19. MODELING PATTERN OF A DELAY PR AUTOMATA WITH TASKS SUBJECT TO FAILURE..... 112
FIGURE 4.20. ME AUTOMATON $G1$ IN EXAMPLE 3..... 114
FIGURE 4.21. ME AUTOMATON $G2$ IN EXAMPLE 3..... 115
FIGURE 4.22. TL AUTOMATON $G3$ FOR TASK A IN EXAMPLE 3..... 116
FIGURE 4.23. TL AUTOMATON $G4$ FOR TASK B IN EXAMPLE 3..... 116
FIGURE 4.24. TL AUTOMATON $G5$ FOR TASK C IN EXAMPLE 3..... 117
FIGURE 4.25. PR AUTOMATA $G6$ IN EXAMPLE 3 118
FIGURE 4.26. GANTT CHART FOR A SAMPLE SCHEDULE IN EXAMPLE 3- CASE ONE 120
FIGURE 4.27. GANTT CHART FOR A SAMPLE SCHEDULE IN EXAMPLE 3- CASE TWO 120

1

General introduction

1 General introduction

1.1 Introduction

The objective of scheduling problems is to find the optimal sequence for performing a set of tasks by respecting predefined constraints and optimizing time, energy, etc. This problem splits in different classifications such as single machine scheduling, identical, uniform, unrelated and dedicated parallel machine scheduling, and open shop, flow shop and job shop scheduling (Zobolas, Tarantilis, and Ioannou 2008).

In most of these problems, researchers consider that only one resource is assigned to each task. While in some application domains, it is essential to assign more than one resource to each task. These resources can be composed of machines, dies, pipes, fixtures, guided vehicles, industrial robots, tools or even multi-skilled workforces. There are various applications for Multi-Resource Sharing (MRS) scheduling problems. For example, scheduling tasks in an oil seaport; oil transfer operations and maintenance operations can be considered as tasks and valves can be considered as critical resources. A liquid transfer can be carried out through a temporary alignment in the network by opening the valves in the alignment and closing all adjacent ones in order to isolate it from the rest of the network. Therefore, several valves, i.e. multi-resources, may be used to do a liquid transfer task. While it may be necessary to perform maintenance on the valves of the same alignment. This issue prevents the liquid transfer operation which means conflict between two tasks that share multiple resources (Quintero Garcia 2015).

It is noteworthy to mention that considering multi-resource sharing criteria in the scheduling problem, increases the problem complexity. In fact, it will be converted to an NP-hard problem (Edis, Oguz, and Ozkarahan 2013; Hartmann and Briskorn 2010). Therefore, this kind of problem should be solved in a manner that prevents rate of computational time to become exponential.

There are different approaches to model MRS scheduling problem such as (max,+) algebra, Petri nets, automata theory or approaches that are not in the field of discrete event systems (non-DES). Among these approaches, automata and Petri net models are more expressive and also robust against changes in the parameter setting and against changes in the problem specification. Petri net models demonstrate dynamic behavior of the system. Whereas, the objective of modeling by automata is to show state space of the system. For instance, task

execution in a scheduling problem through state space of an automaton can be modeled as follows: one state can show the situation that a task is not yet executed. Then, by taking a transition, the automaton moves to a state where it executes the task. After finishing the task, the automaton takes another transition and moves to a state where the task is finished. Hence, in the state space of an automaton execution of tasks can be shown visually. Therefore, automata is a proper means for visual illustration of sequence of tasks in a schedule. Through automata theory, various hypotheses of the problem can be demonstrated by automata models and its global behavior can be shown by composition of automata. Furthermore, properties can be defined to verify correctness of the model and to solve the scheduling problem.

There are mainly two types of automata that can simulate time and task durations for modeling scheduling problems aiming at minimizing makespan: 1. Automata based on weights (i.e. weighted automata) 2. Automata based on clocks. In Chapter 2, it is shown that simpler and more abstract models can be built for the MRS scheduling problem through weighted automata. A problem has different components defined in the problem description. Each component may be needed to be modeled separately in one local automaton. For solving the problem performing analyses, it is necessary to obtain the global behavior of the problem. To this end, synchronous composition of all the local automata should be build which shows global behavior of the problem. A literature review on synchronous composition of weighted automata is investigated in the next section (Section 1.2).

The problem discussed so far was completely deterministic. All the information concerning the tasks to be executed was known in advance, including their identity, inter-dependence and duration. Furthermore the starting time of tasks were deterministic. The same goes for the resources who assumed to be reliable. Real life is not like that. New tasks may arrive in the middle of the performance, while others may be canceled. Duration of tasks may be more or less that the expected time, cost of task performance may change, resources may break down, etc. In these situations the evolution of the system depends on the actions of two “players” in each moment, the scheduler which decides to wait or to start which task in a given situation and the “environment”, which denotes all sources of uncontrollable events such as the start time or termination of a task or resource failure.

Despite of importance and significant role of explained criteria, only few numbers of studies involve these aspects in scheduling problems.

1.2 State of the art

In this section, the relevant literature is reviewed to show existing research gaps in scheduling problems and to distinguish the present research work from previous ones. Finally, a classification of described studies is presented.

1.2.1 State of the art on general scheduling problems

Confessore, Giordani, and Rismondo (2007) investigate a multi-project scheduling problem. In this problem, each project consists of a set of activites with precedence constraints and require specific amounts of resources. Resources may be used only by one project or they may be shared between projects. The objective is to minimize each project schedule makespan by considering precedence and resource constraints. A makespan is defined as the maximum completion time of tasks or in other words the execution time of all the tasks only once. The authors presented a multi-agent system model, and an iterative combinatorial auction mechanism for the agent coordination. They developed a dynamic programming formulation for the combinatorial auction problem, and heuristic algorithms for both the combinatorial auction and the bidding process. Xian, Lu, and Li (2007) propos a method for scheduling multiple real-time tasks in multiprocessor systems with the objective of energy minimization. The studied system supports Dynamic Voltage Scaling (DVS). The resources are multiprocessor systems with uncertain workloads under hard real-time constraints. The investigated problem is NP-hard. For solving it, firstly through a polynomial-time heuristic method the problem is converted to a probability-based load balancing problem. Then, it is solved with worst-fit decreasing bin-packing heuristic and the efficiency of their method is shown comparing to the existing methods.

Kellerer and Strusevich (2008) deal with a minimizing the makespan of a scheduling problem that consider m parallel machines as resources. The machines in this problem are dedicated. That is, machine are assigned to jobs in advance. In addition to the main resources, there is a type of an additional resource that may be assigned to a job at any instance to accelerate its process. The authors studied two cases; the presence of a single renewable additional resource of unit amount as well as the presence of several available units of the resource for its generalization. The problem studying in this paper is NP-hard. They propose algorithms to solve the problem for the case of 2 machines, a fixed number of machines and arbitrary number of machines.

Ooshita, Izumi, and Izumi (2009) investigate minimization of global makespan in a parallel computing environment where different jobs should be

executed by machines in different organizations. The authors define a cooperation degree between organizations while each organization does not allow the completion time of its own jobs to be delayed in predefined cooperation time in global makespan. In fact, they model a cooperative multi-organization scheduling problem considering a degree cooperative as constraint. Xi, Jiang, and Zhang (2009) studies multi-resources-constrained job shop scheduling problem. The resources in this paper are machines and molds. The author proposed a heuristic algorithm to solve the scheduling problem and shows that time complexity of the algorithm is low. Therefore a schedule can be obtained in a reasonable time through the proposed algorithm. In this study, multiple resources (e.g. groups of machine, molds and fixture) are used at the same time to perform a task. Furthermore, machine group and mold group for each procedure is unique. Whereas, optional machine and mold in these groups are not only, that is there exist many machines and molds that can perform each procedure. Castro, Harjunkoski, and Grossmann (2009) deal with the scheduling of continuous plant considering cost minimization. In this problem, there energy usage should be optimized depending on the energy pricing variations during the day. Moreover, uncontrollable events are concerned in the scheduling such as machine breakdown or urgent orders that need to be satisfied.

Heimerl and Kolisch (2010) propose an integer mathematical model for scheduling of IT-projects. Their objective is to minimize the total cost which consists of skilled human resource cost. The authors consider multi-resource sharing aspect with assigning multi-skilled human resources to different projects. They solve the proposed problem using an exact method and show its efficiency compared to simple heuristics. Yusta, Torres, and Khodr (2010) addresses the problem of optimum production schedule in order to maximize the industry profit respecting to hourly variation of the electricity price in the spot market. To this end, a mathematical optimization model is proposed to simulate costs and the electricity demand of machining process. Then, it is solved through generalized reduced gradient approach.

Edis and Ozkarahan (2011) addresses a resource-constrained identical parallel machine scheduling problem with machine eligibility restrictions. The aim of this study is makespan minimization. In this problem, there exist a set of machine types for performing tasks. Each type of machine consists of certain number of machines. Moreover, for performing each task, besides machines, certain additional resources such as automated guided vehicles, machine operators, dies, tools, pallets, industrial robots, etc. maybe used. This problem is classified into the NP-hard class of problems. The problem is modeled

through three approaches: an integer programming (IP) model, a Constraint Programming (CP) model and a combined IP/CP model. The models are solved in OPL Studio 3.7™ and accordingly, time complexity for solving each of them are also discussed.

Quintero et al. (2013) presents a (max,+) optimization model for scheduling tasks in an oil seaport. The objective of this schedule is to minimize the total cost due to delay penalties. In this study, the start time of maintenance operations on valves are fixed. Oil transfer operations and maintenance operations are considered as tasks and valves are considered as critical resources. Since the oil pipeline is confined to a certain number of valves and pipes, there exist conflicts for using valves. Therefore, multi-resource sharing is considered in this study. In another work, Quintero et al. (2014b) proposes a (max,+) optimization model for the same scheduling case study considering flexible preventive maintenance. In another article, Quintero et al. (2014a) explore the integration of failure risk into their former studies. Quintero, Niel, and Aguilar (2015) investigate the same problem and case study by considering flexible resource assignment, i.e. resources are not associated to the tasks prior to the scheduling.

Luo et al. (2013) investigates a bi-objective hybrid flow shop scheduling problem. For this purpose, a new ant colony optimization meta-heuristic is proposed to improve production efficiency in and electric power cost with the presence of time-of-use electricity pricing strategy. In fact, production efficiency increases by minimizing the makespan.

Moon and Park (2014) tackle flexible job-shop problem concerning time-dependent and machine-dependent electricity costs with distributed energy resources. The aim of this study is minimizing the total production cost. The energy resources are consist of energy storage and renewable energy resources such as solar energy and wind. The solving approaches in this study are constraint programming and mixed-integer programming.

Afzalirad and Rezaeian (2016) investigates an unrelated parallel machine scheduling problem with resource constrains, sequence-dependent setup times, different release dates, machine eligibility and precedence constraints. Restricted number of additional resources such as labors, tools, jigs, etc. are taken into account. Therefore, this problem can be classified to multi-resource sharing problems. The problem is represented through an integer mathematical model. Then it is solved through two meta-heuristic algorithms including genetic algorithm and artificial immune system with objective of minimizing the makespan. Kundakcı and Kulak (Kundakcı and Kulak 2016) propose

efficient hybrid Genetic Algorithm methodologies in order to minimize makespan of a dynamic job shop scheduling problem. In this problem, uncontrollable events such as random job arrivals, machine breakdowns and changes in processing time are considered.

1.2.2 State of the art on scheduling problems modeled by automata

In this section, the literature relevant to the scheduling problems modeled by automata is reviewed. More detail is provided in Chapter 2.

Hune, Larsen, and Pettersson (2001) investigate the problem of scheduling and synthesizing distributed control programs for a batch production plant and use UPPAAL to solve the problem. In this problem, multi-resource assignment is not taken into account. Abdeddaim and Maler (2001) model classical job-shop scheduling problem by a special class of timed automata. It is noteworthy to mention that in a shop problem, e.g. job shop problem every resource is allowed to be used by a single task. Hence, studies considering shop problems do not use multiple resources to execute a task. Niebert and Yovine (2001) concern a casy study on verification of hybrid systems. In this study, an optimal dynamic scheduler is derived for a cyclic experimental chemical batch plant at Dortmund. The authors model the problem by timed automata. In this article multi-resource allocation is not allowed.

Yasmina Abdeddaim et al. (2003) address the problem of optimal job-shop scheduling of partially-ordered tasks on parallel machines. The problem is formulated by timed automata and the objective is to minimize the makespan. Multi-resource sharing is not considered in the presented model.

G. Behrmann et al. (2005) address a type of job shop scheduling problem for lacquer production. The authors investigate firstly schedulability of the problem. Then, add storage and delay costs to the problem and propose a method to minimize the cost. They model two case studies by timed automata and priced timed automata. The problems are solved by reachability analysis.

Panek, Stursberg, and Engell (2006) present a new approach to minimize the makespan of job-shop scheduling problems. In this article, the problem is modeled by timed automata. In another article, Panek, Engell, and Stursberg (2006) apply their aforementioned scheduling method in a case study from the chemical industry. The goal of this problem is to investigate schedulability of the problem and also to minimize the makespan. Abdeddaïm, Asarin, and Maler (2006a) use timed automata for minimizing the makespan of a classical job-shop problem. The authors also consider uncertain task duration.

David, Illum, and Larsen (2009) proposed a framework to model and analyze a variety of schedulability scenarios for problems that deal with multi-processor systems, timing uncertainties in arrival and execution times, possible dependencies of tasks and preemption of resources. The problem is modeled by timed automata.

Subbiah and Engell (2010) propose a timed automata model and solve a scheduling problem with sequence-dependent changeover procedures and limited discrete resources. The authors model the problem through interacting timed automata. In order to generate a schedule, a cost-optimal reachability analysis is performed to minimize the makespan.

Marangé et al. (2011) propose a job-shop scheduling model by communicating automata to handle reconfiguration of a manufacturing plant.

Alves et al. (2016) addresses a supervisory scheduling problem in manufacturing systems in order to maximize parallelism among resources. They model the problem through deterministic finite automata. Uncontrollable events may occur during the execution of sequence of tasks. The main objective of this study is not to minimize the makespan, but to maximize the parallelism of working resources. Nikou, Tumova, and Dimarogonas (2016) model a problem of cooperative task planning of multi-agent systems considering timed constraints by weighted automata. Shehabinia, Lin, and Su (2016) model a scheduling problem under multiple job deadlines through time-weighted automata. In the article, the objective is to meet all job specifications and deadlines.

1.2.3 State of the art on synchronous composition of weighted automata

In this section, the literature relevant to the synchronous composition of weighted automata is reviewed. More detail is provided in Chapter 3.

To the best of our knowledge, only four researches have proposed a new composition. Komenda, Lahaye, and Boimond (2009b) in one of the articles a synchronous composition for (max, +) automata is proposed. This composition could be employed for modeling of multi-resource sharing scheduling problems. Whereas, the minimum makespan cannot be obtained through this composition. The same authors propose another type of synchronous composition for (max, +) automata. Despite the minimum makespan can be found through a reachability analysis on this composition, simultaneous execution of actions from different local automata cannot be shown (Lahaye, Komenda, and Boimond 2015). Su, Van Schuppen, and Rooda (2012) propose a synchronous composition for weighted automata. Whereas, through this

composition, simultaneous behavior of tasks cannot be show. Quintero (2015) proposes alphabets for the time-optimal synchronous composition of six tropical automata. The transition function is not defined and through the presented alphabet, the minimum makespan can be obtained for a scheduling problem for which the only constraint is task conflict. Furthermore, this formulation is not generalized to the case of n automata.

1.2.4 State of the art on MRS scheduling considering uncontrollable environment

In this section, the literature relevant to the scheduling problems considering uncontrollable environment is reviewed. More detail is provided in Chapter 4.

Girault et al. (2003) present a new scheduling heuristic called Fault-Tolerance Based Active Replication to produces distributed fault-tolerant schedules for embedded systems. In this study, the processor failure is considered as uncontrollable parameter.

Abdeddaïm, Asarin, and Maler (2006a) use timed automata for minimizing the makespan of a classical job-shop problem. The authors also consider uncertain task duration.

Abdeddaïm, Asarin, and Sighireanu (2009) present a subclass of Timed Game Automata (TGA), called Task TGA which is defined as networks of communicating tasks. In this network, the start time of tasks are deterministic, while their duration are uncertain.

Dumitrescu et al. (2010) propose a framework for multi-criteria optimal controller synthesis to model and optimize fault-tolerant distributed systems considering task execution cost and its service quality. To model the multi-task system, labeled transition system is defined based on input and outputs events.

Atto, Martinez, and Amari (2011) provide a (max,+)-based method to supervise discrete event systems subject to tight time constraints. The authors model the studied system by Petri net. The method is applied to an example of industrial manufacturing plant. Moreover, possible failures are taken into account.

Su, Van Schuppen, and Rooda (2012) address a minimum-makespan supervisory synthesis job shop problem. They assume also occurrence of uncontrollable events. The authors models the problem by weighted and un-weighted deterministic finite state automata.

Kim, Zhou, and Lee (2014) propose a method for steady scheduling of a single-armed cluster tool based on Petri net and (max,+) algebra. They concern disruptive events during the fabrication process.

Fernández Anta et al. (2015) deal with an online system consisting of tasks with different execution times that arrive continuously to be executed on sets of machines which are subject to crashes and restarts. The objective is to minimize execution time and energy. Cimatti, Micheli, and Roveri (2015) address the problem of temporal planning considering uncontrollable duration of actions.

Dorndorf, Jaehn, and Pesch (2017) tackle the problem of assigning flights to airport gates while starting and completion times of flight activities are stochastic.

1.2.5 Synthesis of the state of the art

Table 1.1 illustrates the classification of the presented papers in the scheduling problem literature. In this table, the reviewed papers are classified based on four criteria: 1. the optimization objective of the discussed problem, 2. considering multi-resource sharing constraint, 3. concerning uncontrollable situations in the scheduling problem and 4. the modelling approach by which the problem is modeled. In the following these criteria are explained.

Based on this literature review and Table 1.1, it can be observed that some researchers consider “time” optimization with objective of makespan minimization or defining temporal constraints to control maximum delay. Another group of researchers minimize the “cost” which consists of penalty cost due to delays, cost of energy or any resource. The rest of researchers minimize total “energy” used by resources to execute tasks.

An analysis on table 1.1 shows that only few researchers consider multi-resource sharing aspect and among them no author models the problem using automata theory. Despite expressiveness of automata for modeling problems, few studies model the scheduling problem by automata. A group of them use approaches that are not classified in the field of discrete event systems. Most of these approaches use equations for modeling problems that as explained in the introduction section, there are not as expressive as automata. Some of them use (max,+) algebra equations for modeling which are also not visual. Another group utilize Petri net for modeling the problem. Modeling by Petri net is also a visual approach. Whereas, as explained in the introduction section, this approach shows the dynamic behavior of the problem and not the state space, which is our intent. Another analysis proves that the few researchers that model

the problem by automata theory, consider uncontrollable parameters in scheduling problems. Among these studies, none of them concern multi-resource sharing issue.

Beside the presented research gaps, referring section 1.2.3, no researcher proposes an appropriate synchronous composition for weighted automata to be applied to MRS scheduling problem. Since there are two issues that at least one of them concern the existing synchronous compositions in the literature: 1. Being unable to show all possible behaviors of the scheduled problem such as simultaneous execution of actions in different automata 2. Not containing a trajectory with the minimum makespan for the problem.

As explained in the introduction of this chapter, for generating a schedule, performing synchronous composition of the components of the problem is indispensable. Thus, two approaches can be followed to model the MRS scheduling problem using automata:

The first approach is to model the problem by weighted automata and then translate the models to a clock-based automata for which the formalism of synchronous composition exist. Timed automata is a clock-based automata containing necessary formal features for modeling of MRS scheduling problems containing controllable actions and tasks. Formalism of synchronous composition of timed automata is defined before and moreover, a timed automaton model is implementable in a formal verification tool. In a formal verification tool, the behavior of them problem can be synthesized and the optimal schedule can be obtained automatically.

The second approach is to model the problem by weighted automata and propose a synchronous composition for weighted automata to compose them. In this approach, the necessary analysis for finding the optimum schedule can be done on the weighted automata composition of models. Thence, it is not needed to translate models to timed automata which may reduce the complexity of the solving approach, since the translation step is omitted from the analysis procedure.

Table 1.1. Classification of literature about scheduling problems

Authors	Optimization objective	Multi-Resource Sharing	Modeling Approach	Uncontrollability
(Hune, Larsen, and Pettersson 2001)	time		automata	
(Abdeddaim and Maler 2001)	time		automata	
(P Niebert and Yovine 2001)	time		automata	
Girault et al. (2003)	time		Data-flow graph	✓
(Behrmann et al. 2005)	time-cost		automata	
(Panek, Stursberg, and Engell 2006)	time		automata	
(Panek, Engell, and Stursberg 2006)	time		automata	
(Yasmina Abdeddaïm, Asarin, and Maler 2006)	time		automata	✓
(Confessore, Giordani, and Rismondo 2007)	time	✓	Non-DES	
(Xian, Lu, and Li 2007)	energy		Non-DES	✓
(Kellerer and Strusevich 2008)	time	✓	Non-DES	
(David et al. 2009)	time		automata	✓
(Y. Abdeddaïm, Asarin, and Sighireanu 2009)	time		automata	✓
(Xi, Jiang, and Zhang 2009)	time		Non-DES	
(Ooshita, Izumi, and Izumi 2009)	time		Non-DES	
(Castro, Harjunkoski, and Grossmann 2009)	cost		Non-DES	✓
(Subbiah and Engell 2010)	time		automata	
(Yusta, Torres, and Khodr 2010)	cost		Non-DES	
(Dumitrescu et al. 2010)	time		automata	✓
(Heimerl and Kolisch 2010)	cost	✓	Non-DES	
(Marangé et al. 2011)	time		automata	✓

Authors	Optimization objective	Multi-Resource Sharing	Modeling Approach	Uncontrollability
(Edis and Ozkarahan 2011)	time	✓	Non-DES	
(Atto, Martinez, and Amari 2011)	time		Petri net	✓
(Su, Van Schuppen, and Rooda 2012)	time		automata	✓
(Luo et al. 2013)	time-cost		Non-DES	
(Quintero et al. 2013)	cost	✓	(max,+) algebra	
(Moon and Park 2014)	cost		Non-DES	
(Kim, Zhou, and Lee 2014)	time		Petri net	✓
(Quintero et al. 2014a)	cost	✓	(max,+) algebra	✓
(Quintero et al. 2014b)	cost	✓	(max,+) algebra	
(Fernández Anta et al. 2015)	time-energy		Non-DES	✓
(Quintero, Niel, and Aguilar 2015)	cost	✓	(max,+) algebra	
(Cimatti, Micheli, and Roveri 2015)	time		Non-DES	✓
(Shehabinia, Lin, and Su 2016)	time		automata	
(Kundakcı and Kulak 2016)	time		Non-DES	✓
(Nikou, Tumova, and Dimarogonas 2016)	time		automata	
(Alves et al. 2016)	time		automata	✓
(Afzalirad and Rezaeian 2016)	time	✓	Non-DES	
(Dorndorf, Jaehn, and Pesch 2017)	time		Non-DES	✓

1.3 Research question[1]

According the presented research gaps, five research questions are identified:

- To model multi-resource sharing scheduling problem using automata theory
- To develop an efficient solving approach for the MRS scheduling problem modeled by automata theory which can solve the problem in a duration appropriate for decision making.
- To develop a synchronous composition for weighted automata appropriate for MRS scheduling problem
- To develop a solving approach using the synchronous composition of weighted automata
- To model and solve a MRS scheduling problem considering uncontrollable events and parameters using timed game automata

1.4 Contribution

1.4.1 Chapter 2: MRS scheduling through translation of weighted to timed automata

In the second chapter, a two-step modeling approach is presented to integrate multi-resource sharing issue in scheduling problems using automata theory. In the first step, the problem is modeled by weighted automata which yields a simple and abstract model. In the second step, the weighted model is translated to timed automata. The advantage of this translation is using formal verification tools for solving the problem. Thence, the timed models are implemented in UPPAAL which is a mature formal verification tool.

To solve the problem, an algorithm is developed based on iterating reachability analysis to obtain sub-optimal makespan.

1.4.2 Chapter 3: Multi-resource sharing scheduling by using synchronous composition of weighted automata

The main contribution of this chapter is defining a new synchronous composition for weighted automata. The weighted automata models of MRS scheduling problem is compose through this definition. Then, a time-optimal reachability analysis algorithm is developed to find the time optimal schedule.

[1] Verrou

1.4.3 Chapter 4: Multi-resource sharing scheduling considering uncontrollable environment

In this chapter, various uncontrollable events and parameters such as start time, duration of task and failure occurrence in a MRS scheduling problem is modeled by timed game automata. Then, a synthesis tool named TIGA is used to solve the problem by performing a time-optimal strategy synthesis.

1.5 Background

Formal verification techniques are engaged with TCTL property verification, which is a type of property languages. These techniques are based on constructing reachability graph of the system and traversing through this graph that can be done in different manners such as breadth-first search or depth-first search. In this section, a brief introduction to these concepts is provided.

1.5.1 Formal verification (or model checking)

The domain of verification is concerned with proving or disproving that systems behave as required under all possible circumstances. In fact, a model checker uses an algorithm to check if a logical formula holds in a system. A system can be represented by an automaton containing some states and transitions. Thus, the set of all paths in the transition graph of the automaton represents the set of all possible behaviors of the system. So the problem of verification can be reduced to the problem of checking the existence of certain paths in the transition graph. Reachability properties are a type of the properties to which verification is applied; they are verified by a path to know whether or not it reaches some specific states. In some cases it should be checked if all behaviors avoid a set of “forbidden” states, e.g. a state where the system is in the risk of failure, and in some others behaviors should be found that proceed to a desired final state, e.g. a state where all tasks have terminated. Such properties are examples of *safety* and *liveness* properties, respectively. If the system fails to meet a specific property, the model checker can be asked to generate a counterexample to show how the property was violated. Furthermore if the system meets the property, the model checker can be also asked to generate a witnessing trace to illustrate one of the possible paths for which the property holds.

By assigning numerical weights to the automaton transitions, one can associate real numbers with paths and search for paths that have minimum weight, using fastest path algorithms. Therefore by performing reachability

analysis along with using shortest path algorithms, the minimum makespan can be found.

1.5.2 Property-specification language

In this section required formalisms for expressing properties of timed systems is presented. There exist two types of formalisms, linear-time and branching-time. In linear time, properties are interpreted as sets of executions and specifications are evaluated on runs. While in branching time, properties are validated on sets of execution trees and specifications are checked on semantic graphs (Tripakis 1998).

Since branching time properties are used by powerful tools such as UPPAAL and Kronos, in this thesis, only this kind of properties is considered.

1.5.3 The branching time logic TCTL

Branching time properties are expressed by Timed Computation Tree Logic (TCTL) and has been introduced in (R Alur, Courcoubetis, and Dill 1993). Let's $\mathcal{I}$ be the set of all intervals of real numbers of the form $[c, c'], [c, c'), (c, c'], (c, c'), (c, \infty)$ and $[c, \infty)$, where $c, c' \in \mathbb{N}$. Also let $Props$ be a set of atomic proposition.

Syntax and semantics: A formula ϕ in TCTL is defined according to the following syntax:

$$\phi \coloneqq true \mid p \mid \neg\phi \mid \phi \vee \phi \mid \exists \phi \mathcal{U}_I \phi \mid \forall \phi \mathcal{U}_I \phi \tag{1.1}$$

Let A be a timed automata with the set of states Q and $P: Props \rightarrow 2^Q$ assign a set of discrete states of A to each atomic proposition. $TCTL$ formulae are interpreted over states of A. For a state s, a TCTL formula ϕ the satisfaction relation $s \vDash_p \phi$ is defined inductively as follows:

$$\begin{array}{lll}
s \vDash true & & \\
s \vDash p & \textit{iff} & s \in P(p) \\
s \vDash \neg\phi_1 & \textit{iff} & \text{not } s \vDash \phi_1 \\
s \vDash \phi_1 \vee \phi_2 & \textit{iff} & s \vDash \phi_1 \text{ or } s \vDash \phi_2 \\
s \vDash \exists(\phi_1 \mathcal{U}_I \phi_2) & \textit{iff} & \exists \rho = s \xrightarrow{\delta_1} \xrightarrow{e_1} \ldots \text{s.t. } time(\rho) = \infty \text{ and} \\
 & & \exists i \,.\, \Sigma_{j \le i} \delta_j \in I \text{ and } \rho(i) + \delta_i \vDash \phi_2 \text{ and} \\
 & & \forall j < i. \forall \delta \le \delta_j . \rho(j) + \delta \vDash \phi_1 \vee \phi_2 \\
s \vDash \forall(\phi_1 \mathcal{U}_I \phi_2) & \textit{iff} & \forall \rho = s \xrightarrow{\delta_1} \xrightarrow{e_1} \ldots s.t. time(\rho) = \infty \text{ and} \\
 & & \exists i \,.\, \Sigma_{j \le i} \delta_j \in I \text{ and } \rho(i) + \delta_i \vDash \phi_2 \text{ and}
\end{array} \tag{1.2}$$

$$\forall j < i. \forall \delta \leq \delta_j. \rho(j) + \delta \vDash \phi_1 \vee \phi_2$$

s satisfies $\exists(\phi_1 \mathcal{U}_I \phi_2)$ if for some run starting from s and a point along the run such that the time spent until that point belongs to the interval I, ϕ_2 holds at that point and ϕ_1 holds continuously until that point. The difference between meaning of this property and $\forall(\phi_1 \mathcal{U}_I \phi_2)$ is that in the second one, all such runs should meet the condition.

The following abbreviations are defined:

$$\begin{aligned} &\exists \Diamond_I \phi \coloneqq \exists true\, \mathcal{U}_I \phi \\ &\forall \Diamond_I \phi \coloneqq \forall true\, \mathcal{U}_I \phi \\ &\forall \Box_I \phi \coloneqq \neg \exists \Diamond_I \neg \phi \\ &\exists \Box_I \phi \coloneqq \neg \forall \Diamond_I \neg \phi \end{aligned} \tag{1.3}$$

If the initial state of a timed automata satisfies a formula, it can be said that the automata satisfies the formula (Tripakis 1998).

Example: Safety and liveness properties can be expressed by TCTL formulas. The formula $\forall \Box_{\leq 2} \phi$ means that ϕ holds before 2 time units.

1.5.4 TCTL properties

TCTL formulae are classified to state formulae and path formulae. A state formula describes individual states. While path formula quantifies over paths of the model. Path formulae are consist of reachability, safety and liveness properties.

a. State formulae:

A state formula is an expression for evaluating a state without considering the behavior of the model. For example verifying the value of a variable in a state can be expressed as a state formula.

Safety and liveness properties are two main groups of properties.

b. Safety properties:

Safety properties requires that for every possible execution of the system nothing undesirable happens. For example a safety property could be avoiding a process from being in the failure state. A variation of this property is to ask if something will possibly never happen. For example in a game, a safe state would be where there is still possibility to win the game; in another word, it is the possibility for not losing.

In UPPAAL safety properties should be formulated positively, e.g. something good is always true. There are two types of safety expressions. The first one is to say that φ is true in all reachable states with the formula $\forall\Box\varphi$ ($A[]\varphi$ in UPPAAL). The second one is to ask if there is a maximal path[2] such that φ is always true and can be expressed with the formula $\exists\Box\varphi$ ($E[]\varphi$ in UPPAAL).

c. Liveness properties:

Liveness properties are of the form something good eventually happens. For instance when pressing the hibernate button of the computer, it eventually should pass to the hibernate mode. The simplest form of a liveness property is expressing with a path formula $\forall \Diamond \varphi$ which means that φ is eventually satisfied. Another form of liveness property is *response* property with the form $\varphi \rightsquigarrow \psi$ that is equivalent to $\forall\Box(\varphi \Rightarrow \forall \Diamond \psi)$. This property means that whenever φ is satisfied, ψ will be eventually satisfied. In UPPAAL, $\forall \Diamond \varphi$ and $\varphi \rightsquigarrow \psi$ can be written as $A <> \varphi$ and $\varphi - -> \psi$ respectively.

d. Reachability properties:

This properties ask whether a state formula φ possibly can be satisfied in any reachable state. In another words, it verifies if there exists any path from the initial state along which φ is eventually satisfied. For example in a model that expresses a communication protocol that involves a sender and a receiver, a question can be whether it is possible for a sender to send any message.

A reachability property can be expressed by the path formula $\exists \Diamond \varphi$. In UPPAAL, this formula can be written as $E <> \varphi$ (Behrmann, David, and Larsen 2006).

1.5.5 Techniques for constructing reachability graph of systems

Size of the reachability graph of the system has a great impact on the performance of the model checking problem. It is also name size of the state space and the larger this size is the slower the verification process will be. State space explosion is one of the major limitations in model checking problems (Al-Bataineh 2015). In this section, some techniques for construction of state space are represented. Verification problems are based on searching traces to prove or disprove a property φ. In general, two main approaches are introduced, the fixed-point approach and on-the-fly approach. In the fixed-point approach,

[2] A maximal path is either an infinite path or where the last state don't have any outgoing transition.

an exhaustive search need to be done and all the states should be represented in the memory at the same time. Whereas in on-the-fly approach, only a part of the graph need to be generated and the property will be verified while the graph is constructed.

a. Backward versus forward analysis

Backward and forward analyses are both fixed-point approaches. In the forward analysis method, model checker constructs a characterization of all reachable states from the initial state. While in backward analysis, model checker constructs a characterization of all states that can reach the goal state respecting to behavioral structure. For some examples the speed of model checking with one of these methods may be higher. However backward method is necessary for checking certain modalities such as “Until” and “Eventually” and they can’t be handled by forward analysis (Al-Bataineh 2015). Kronos supports both forward and backward model checking.

b. On the fly approach

In on-the-fly approach (Bouajjani, Tripakis, and Yovine 1997), the state space of the system in generated dynamically and therefore just the minimal amount of information is needed to be stored in the memory. The property is checked while the model checker is generating the graph. On-the-fly model checking is a powerful technique especially when the intent is to disprove a property and to generate a counterexample. In fact, errors are discovered so early during search and thus avoiding the exploration of the entire state space. On the other hand, whenever it is needed to prove that the system is entirely correct respecting to a property, a comprehensive search of the state space is needed and this method will be less efficient. Hence in a group of examples on-the-fly approach is more efficient than fixed-point approach and vice-versa (Al-Bataineh 2015). Model checking in UPPAAL is based on on-the-fly approach.

c. Compositional model checking

The state space explosion occurs in systems with many concurrent components where most of the model checking techniques are inefficient or impossible. A solution in this case is to decompose the system into components and then to verify them individually (Berezin, Campos, and Clarke 1998). If the all the components satisfy the properties properly, it is concluded that the system behaves correctly. The main difficulty is to find if after the parallel composition, all properties still remain satisfied. In fact, in some example,

different processes of the system need to interact with each other to satisfy a property and hence cannot be verified individually (Al-Bataineh 2015).

1.5.6 Digraph traversal algorithms

In this section, two basic approaches of graph searching are introduced. These methods are used in exploring the state space when performing a reachability analysis. The following algorithms explore the automaton in a forward direction. While they could also be done in a backward manner.

a. Breadth first search

The simplest algorithm for performing a reachability analysis is a Breadth-First Search (BFS). Below is the BFS algorithm (Sedgewich and Wayne 2017).

Algorithm 1.1. BFS (from state s)

Put s onto a FIFO queue, and mark s as visited.
Repeat until the queue is empty:
  remove the least recently added state q
  for each unmarked state pointing from q: add to queue and mark as visited.

a. Depth-first search

There is a recursive algorithm for depth-first search of an automaton. Firstly one state is selected for visiting. States in an automaton are assumed to split to two sets, the ones that are not visited yet and those that are visited. Once each non-visited state is visited, it will be removed from the first set and will be added to the second set. Below is the algorithm of Depth-First Search (DFS) (Sedgewich and Wayne 2017).

Algorithm 1.2. DFS (to visit a state q) (Sedgewich and Wayne 2017)

Mark state q as visited.
Recursively visit all unmarked states pointing from q.

In a random-depth-first search method, the first state for visiting should be chosen randomly.

This algorithm is the same as DFS algorithm for a directed graph (digraph). Java code of a digraph DFS is detailed in Appendix B.

Technically the difference between a DFS and BFS algorithm is the place where new nodes are added to the waiting list for visiting, at the end(BFS) or at the beginning (DFS). In other words, the list is FIFO (First In First Out) in BFS and LIFO (Last In First Out) is DFS (Yasmina Abdeddaïm, Asarin, and Maler 2006).

1.6 Conclusion

In this chapter importance of modeling by automata theory, taking into account multi-resource sharing and uncontrollable events and parameters in the scheduling problem are explained. Then, the previous studies on scheduling problems are reviewed and classified to show the existing research gaps. Afterwards, the research question and contribution are presented. Finally, a basic background on the main keywords is provided.

The remainder of this thesis is organized as follows. In Chapter 2 a MRS scheduling problem is modeled by weighted and timed automata and solved. In Chapter 3, a synchronous composition is proposed for weighted automata to solve the problem by performing time-optimal reachability analysis on weighted automata models. In Chapter 4, a MRS scheduling problem considering uncontrollable parameters modeled by timed game automata and solved. Finally, Chapter 5 summarizes results and concludes with future research opportunities.

2

Multi-resource sharing scheduling through translation of weighted to timed automata

2 Multi-resource sharing scheduling through translation of weighted to timed automata

2.1 Introduction

In this chapter, new models and solving approaches are proposed for multi-resource sharing scheduling problems through automata theory while all the tasks are controllable.

As mentioned in Chapter 1, scheduling problems can be modeled by either weight-based automata like weighted automata, or clock-based automata like timed automata. During this chapter, it is shown that simpler and more abstract models can be built for the MRS scheduling problem through weighted automata. Whereas there are two issues for using current synchronous compositions. Either the minimum makespan cannot be obtained through using them, or they don't show a complete composed behavior of the components of the model. Thence, it is not possible to analyze weighted automata (WA) models directly. Therefore, firstly the MRS scheduling problem should be modeled by WA. Then, after modeling the problem, there exist two strategies to solve the scheduling problem. One solution is to translate the WA model to another type of automata for which there exists synchronous composition (e.g. timed automata) and analyze the new model to attain a schedule. Another solution is to propose a new synchronous composition for WA and use it to analyze the WA models directly and without creating intermediate timed automata models. In this chapter, it is tried to find schedules by means of the first strategy in which timed automata models are used as an intermediate to obtain the optimal schedule from the WA models.

In some studies, automata theory, verification methods and controller synthesis have been used for addressing scheduling problems. These articles are reviewed to show existing research gaps in scheduling problems and to distinguish the present research work from previous ones.

2.1.1 State of the art

Gaubert and Mairesse (1995) present a method for modeling timed concurrent systems modeled as automata with multiplicities in the (max,+) semiring. The authors present applications of this modeling method to performance evaluation

and for finding the makespan in a scheduling problem. Despite formulating equations for finding the minimal makespan, the aim of this study is not generating a schedule.

Norström, Wall, and Yi (1999) develop a formalism for an extended version of timed automata with real-time tasks to solve problems in event-driven systems. This automata can be used for modeling, schedulability analysis, formal verification, and code generation. The authors assign a task with its worst time execution to each transition with a guard specifying possible arrival times of the task. By translating the extended timed automata to a standard timed automata, it is possible to verify properties such as schedulability, functionality and safety properties of the model. An example is also presented to check the schedulability and safety properties of a control program for a turning lathe by using UPPAAL.

Hune, Larsen, and Pettersson (2001) investigate the problem of scheduling and synthesizing distributed control programs for a batch production plant and use UPPAAL to solve the problem. A plant model which is enough accurate for program synthesis is usually so complex. To overcome this complexity, the authors apply an approach of guiding a model according to certain strategies. In this problem, multi-resource assignment is not taken into account. Abdeddaim and Maler (2001) model classical job-shop scheduling problem by a special class of timed automata. To obtain the optimal schedule, algorithms and heuristics are presented to find the fastest path in timed automata. Furthermore the algorithms are implemented in the tool Kronos. Comparing to traditional modles of operations research, proposed technics in this study allow to model more complex dynamic resource allocation problems. It is noteworthy to mention that in a shop problem, e.g. job shop proble, every resource is allowed to be used by a single task. Hence, studies considering shop problems do not use multiple resources to execute a task. Niebert and Yovine (2001) concern a casy study on verification of hybrid systems. In this study, an optimal dynamic scheduler is derived for a cyclic experimental chemical batch plant at Dortmund. In the first step, the behaviour of the plant is modeled by timed automata. In the second step, the models are implemented in the tool Open-kronos and the optimal production schemes are obtaned using reachability analysis. Finaly in the third step high level control codes are derived through post-process of the output of the verification tool. In this article multi-resource allocation is not allowed.

Yasmina Abdeddaim et al. (2003) address the problem of optimal job-shop scheduling of partially-ordered tasks on parallel machines. The problem is

formulated by timed automata and in fact, the optimal schedule is found through searching the fastest path in the automaton. In the presented model, release time and deadline of tasks, communication costs between task and additional resourced are taken into account.

G. Behrmann et al. (2005) expresses that unlike classical approaches, timed automata models are expressive and also robust against changes in the parameter setting and against changes in the problem specification. Therefore they allow modelling of scheduling problems in different kinds. Furthermore, in this paper a type of job shop scheduling problem for lacquer production as a case study is investigated. The authors use a heuristic approach to reduce the search space. They also propose solutions that are applicable for other scheduling cases.

Panek, Stursberg, and Engell (2006) present a new approach to minimize the makespan of job-shop scheduling problems. This approach combines reachability computations for timed automata with a branch-and-bound principle to improve the efficiency of the reachability algorithm by excluding sub-optimal or redundant solutions from the search space. The authors have shown that in large size problems and in a given computation time, the proposed approach produces better schedules than pure Mixed-Integer Programming (MIP) techniques. In another article, Panek, Engell, and Stursberg (2006) apply their aforementioned scheduling method in a case study from the chemical industry. Abdeddaïm, Asarin, and Maler (2006a) use timed automata for solving the classical job-shop problem. They propose shortest path algorithms for timed automata to find the optimal schedules. The authors also investigate non-lazy scheduling with uncertain task duration.

David, Illum, and Larsen (2009) proposed a framework to model and analyze a variety of schedulability scenarios, particularly problems that deal with multi-processor systems, timing uncertainties in arrival and execution times, possible dependencies of tasks and preemption of resources. Scheduling policies in this study include FIFO, Earliest Dead-line First (EDF), and Fixed Priority Scheduling (FPS).

Subbiah and Engell (2010) propose a timed automata model and solve a scheduling problem with sequence-dependent changeover procedures and limited discrete resources. The authors model processing units and the recipes as interacting timed automata components. In addition, they modeled the setup and changeover procedures as operations in the recipe. In order to generate a schedule, a cost-optimal reachability analysis is performed. The computational time complexity of the solving approach used in the article is less than Mixed-

Integer Linear Programming (MILP) techniques. Therefore the approach is easily applicable to practical large-scale problems.

Marangé et al. (2011) propose a job-shop scheduling model by communicating automata to handle reconfiguration of a manufacturing plant. Following a reconfiguration request, a scheduling is generated for a set of products that are produced by a set of machines. This schedule can be obtained by means of reachability analysis on the model.

Alves et al. (2016) addresses a supervisory scheduling problem in manufacturing systems in order to maximize parallelism among resources. The authors model the scheduling problem through deterministic finite automata. The schedule corresponds to the supremal controllable sublanguage contained in the desired behavior of the system. Uncontrollable events may occur during the execution of sequence of tasks and at consequently rescheduling will be necessary. The sequence acquired by this algorithm doesn't have necessarily optimal makespan, whereas it generates good solutions that can be used in real applications. Therefore, the main objective of this study is not to minimize the makespan, but to maximize the parallelism of working resources. Nikou, Tumova, and Dimarogonas (2016) address the problem of cooperative task planning of multi-agent systems considering timed constraints given by Metric Interval Temporal Logic (MITL). A method is presented for control synthesis in a two-stage systematic procedure to satisfy individual and team task specifications by agents as well as the global team. Same as the previous article, this article do not optimizes the makespan. Shehabinia, Lin, and Su (2016) model a scheduling problem under multiple job deadlines through time-WA. In the article, a supremal controllable job satisfaction sublanguage is computed to determine if all job specifications and deadlines are met. In the case the sublanguage is not empty, one of its controllable sublanguages is computed that ensures the minimum total job earliness by adding proper delays. If the sublanguage was empty in order to get a feasible schedule, a set of job deadlines will be determined to be relaxed. Consequently, the goal of this article is to meet deadlines and not minimizing the makespan.

2.1.2 Synthesis of the state of the art

Based on the conducted literature review, automata models are expressive and also robust against changes in the problem specification. Therefore automata theory is an appropriate means for modeling scheduling problems. Whereas, a few studies have successfully used automata theory to solve scheduling problems; yet none of them take into account multi-resource sharing aspect.

Furthermore the objective of some of these studies is to meet task deadlines and not minimizing the makespan. Thus, the aim of this chapter is to minimize the makespan of a set of tasks with possible precedence constraints. The problem is modeled and solved through automata theory and formal verification.

To this regard, the remainder of this chapter is organized as follows: A detailed description of the MRS scheduling problem is given in section 2.2. In sections 2.3 the problem is modeled by WA. Section 2.4 explains the solving approach which needs refinement of WA models to timed automata models. Finally, section 2.5 is devoted to the conclusion.

2.2 MRS scheduling problem description

In the context of this work, various conflicting tasks should be scheduled according to the following constrains:

(1) The duration of each task is predefined and fixed before scheduling.
(2) Task preemption or cancellation is not allowed, i.e. once they are started, they cannot be canceled and should be processed until completion.
(3) There may be conflicts for performing tasks at the same time, but when there is no conflict between them, they should be performed simultaneously.
(4) There may exist precedence constraints between tasks.
(5) All tasks are ready to be executed at time zero.
(6) Resources are pre-assigned to tasks.
(7) Resources are reusable (they are not raw materials and by performing maintenances, they can be used in every cycle).
(8) Each resource can be used to execute only one task at a time, but a task may use more than one resource simultaneously.
(9) Resources are available at time zero.
(10) Resources are reliable and don't breakdown, but they are subject to preventive maintenance which should be defined as a task.

Tasks in this problem consist of either operations or preventive maintenances on resources. The aim of this work is to find a schedule of minimum duration considering these constraints, for executing all the tasks just once, by assigning a start time to each task.

It is necessary to understand the concept of multi-resource sharing as the most important constraint. In a scheduling problem considering multi-resource sharing constraint, each task may utilizes multiple resources simultaneously to be performed. Thus, if two tasks needing multiple resources share a resource,

they cannot be executed at the same time. Moreover, by finishing the first task, the second task may not be able to be executed immediately. Since other resources needed for its execution are still busy by another tasks. Hereafter, multi-resource sharing is explained through an example.

Example 1: Assume there is a set of tasks $T = \{A, B, C\}$ to be done which their durations are 7, 5 and 3 time units respectively. A set of resources $R = \{R1, \dots, R5\}$ is assigned to tasks with resource association details shown in Table 2.1. (e.g. for doing task A resources $R1$ and $R5$ should be engaged.) Hence,

- Tasks A and B are in conflict because of sharing $R1$. Thus they cannot be performed simultaneously.
- Tasks B and C are in conflict since they share $R2$ and $R3$. So they also cannot be executed simultaneously.
- A and C are not in conflict because they don't share any resource. Therefore according to the constraint (3), they should be performed simultaneously. While, if they every task would use a single resource, A and C was in conflict and couldn't be launched at the same time.

Table 2.1. Resource assignment details of Example 1

task	resource				
	R1	R2	R3	R4	R5
A	✓				✓
B	✓	✓	✓		
C		✓	✓	✓	

2.3 Modeling MRS scheduling problem by weighted automata

In this section, the modeling procedure of the proposed problem is introduced.

2.3.1 General principle of modeling procedure

The scheduling problem statement needs to capture four features: task triggering, simultaneity, mutual exclusion and timing. For this purpose, a semi-formal model based on WA is proposed. Inspired from the (max, +) automata formalism, they offer an intuitive way of modeling the behavioral features mentioned above. Thus, each scheduling problem considered in this work is defined as a collections of WAs. The actual scheduling should comply with the WA problem statement. As explained in the introduction, the existing

synchronous compositions for WA cannot be used for the scheduling purpose. Since they either don't contain a trajectory with the optimum makespan to be used as the optimal schedule or are not capable of illustrating simultaneous execution of tasks which is a non-separable issue from scheduling. Hence, there are two possible solutions for solving a scheduling problem that is modeled by WA. The first solution is to employ timed model checking techniques (Baier and Katoen 2008) which are defined for timed automata. To this end, each WA model of the problem statement is systematically translated into a Timed Automaton (TA), according to specific rules defined in the sequel. The resulting model is directly usable by a timed model checking tool. The second solution is to define a new synchronous composition for WA in order to compose WA models directly and solve the scheduling problem without translating to timed automata.

In the next section the scheduling problem is modeled by WA.

2.3.2 Problem statement by weighted automata

Definition 2.1 (weighted automata): A deterministic single-duration-action weighted automata is defined as a tuple $G = (Q, Q_0, \Sigma, f, d, Q_m)$ where

- Q is the set of states,
- Q_0 is the set of initial states,
- Σ is the set of symbols representing actions and silent action (ε) whose duration is zero,
- Q_m is the set of marked states,
- $f: Q \times \Sigma \times \mathbb{R}_{\geq 0} \longrightarrow Q$ is the transition function $(q, a, d(a)) \longrightarrow q'$ where q' is the state reached from q by starting action a which lasts $d(a)$ time units and $d: T \rightarrow \mathbb{R}_{\geq 0}$ assigns a duration to every action.

In should be noted that hereafter, all the WA used for modeling purpose, are deterministic single-duration-action WA.

When taking each transition in WA, two steps are taken, a discrete step and a timed step.

Discrete step: $q \xrightarrow{a} q'$ where a is an action, q is the source and $q'$is the target of transition $q \xrightarrow{a/d(a)} q'$ .This kind of step doesn't take time.

Timed step: $q \xrightarrow{d(a)} q'$ where $d(a)$ is an action, $d(a)$ is the duration of a, q is the source and $q'$is the target of the transition $q \xrightarrow{a/d(a)} q'$. It is clear that the duration of this step is $d(a)$ time units.

Definition 2.2 (MRS scheduling problem): A MRS scheduling problem statement $S = (T, E, Dyn, d)$ consists of

- a set $T = \{t_i | i = 1, \dots N\}$ of tasks that might be necessary to be executed simultaneously,
- a family of mutual exclusion constraint sets $E \in 2^T$ that are modeled as a set G_{me} of mutual exclusion automata.
- a duration function $d: T \to \mathbb{R}_{\geq 0}$ assigning durations to tasks,
- a set $Dyn = G_{tl}^k \cup G_p$ of dynamic models such that $G_{tl}^k \cap G_p = \emptyset$. G_l^k denotes the set of task launcher automata. They are modeled for triggering each task for k times. $|G_{tl}^k| = |T| = N$ which means that for executing every task, there exists one task launcher automaton. G_p denotes the set of precedence automata modeling precedence constraints among tasks. This set could be empty.

In the sequel, the formal models of the three automata models are defined and clarified by applying to Example 1.

a. The mutual exclusion automaton model

Each mutual exclusion requirement is specified by a set T_{me} of tasks that are forbidden to run simultaneously. A natural way to model this requirement is a single state WA featuring one self-loop transition for each task. This model allows all the potential sequences of tasks among T_{me} and prevents two or more tasks of T_{me} to be executed simultaneously.

Definition 2.3 (formal model of the mutual exclusion automaton): A Mutual Exclusion (ME) automata is a WA defined as $ME = (\{q_{0me}\}, \{q_{0me}\}, T_{me}, f_{me}, d, \{q_{0me}\})$ where

- $q_{0me} \in Q_{me}$ is the only state which is both initial and marked state,
- $T_{me} \in E$ is a set of tasks among which there is mutual exclusion,
- $d: T_{me} \to \mathbb{R}_{\geq 0}$ assigns a duration to a task $t \in T_{me}$,
- $f_{me}: \{q_{0me}\} \times T_{me} \times \mathbb{R}_{\geq 0} \to \{q_{0me}\}$ is the transition function where $\forall t \in T_{me},\ q_{0me} = f_{me}(q_{0me}, t, d(t))$ and duration of each transition is equal to $d(t)$, i.e. the duration of the task.

Figure 2.1 shows the generic pattern of ME automaton. In this figure, t_i represents the name of the task and $d(t_i)$ denotes its duration. For any scheduling problem statement, the number of ME automata is equal to the number of sets of mutual exclusions among tasks.

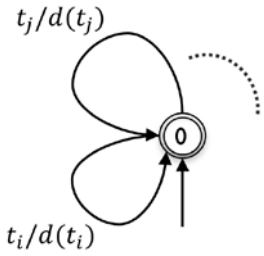

Figure 2.1. Modeling pattern of weighted ME automaton

Example 1 (continue): Let us apply the proposed ME automaton model to Example 1. As explained above, there are a number of sets of mutual exclusion. Each set should be illustrated in a separate ME automaton. In a ME automaton, each task should be modeled as a loop transition joint to the only state existing in the automaton.

Corresponding ME automata for Example 1 is depicted in Figure 2.2. In this figure,

- In automaton $G1$, first task A should be performed for 7 time units and then task B for 5 time units or vice versa, i.e. A and B cannot be executed at the same time.
- In automaton $G2$, first task C should be performed for 3 time units and then task B for 5 time units or vice versa, i.e. B and C cannot be executed at the same time.
- B is in common between automata $G1$ and $G2$. Therefore, separate transitions labeled with B should be synchronized.
- Duration of A is 7 time units and duration of C is 3 time units. Since they are not in conflict, transitions labeled with B and C can be taken at the same time. Whereas the transition labeled C reaches to the state 0 after 3 time units and the transition labeled A reaches this state after 7 time units.

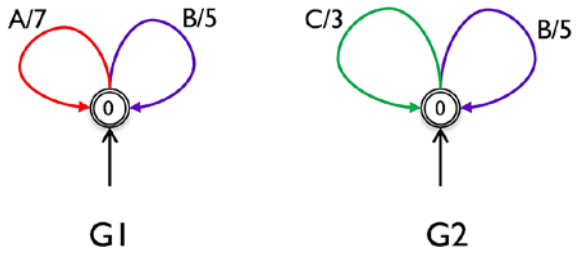

Figure 2.2. Corresponding ME automata for Example 1

As illustrated in Figure 2.2, multi-resource sharing creates a special kind of task assignment to automata. This conflict is such that some tasks become

common among two or more ME automata (task B). If only one resource would be used by each task, no task was is common among ME automata. This kind of tasks are named common tasks. Conversely, those that occur in just one ME automaton (tasks A and C) are named individual tasks.

b. The task launcher automaton model

A task launcher model specifies the number of times a task should be triggered within the desired scheduling.

Definition 2.4 (formal model of the task launcher automaton): A task launcher automaton TL^k is a WA which triggers task t for k times. It is an element of G_{tl}^k defined as $TL^k = (Q_{tl}, \{q_{0tl}\}, \{t\}, f_{tl}, d, \{q_{mtl}\})$ in which TL^k is a task launcher automata that launches task t, k times. In this tuple,

- Q_{tl} is the set of states,
- $q_{0tl} \in Q_{tl}$ is the initial state which is not a marked state,
- $t \in T$ is the task to be launched,
- $d: \{t\} \to \{0\}$ assigns zero duration to the task t,
- $q_m \in Q_{tl}$ is the marked state,
- $f_{tl}: Q_{tl} \times \{t\} \times \mathbb{R}_{\geq 0} \to Q_{tl}$ is the transition function such that:

$$\forall k \in \mathbb{N}, \underbrace{(f_{tl} \circ \ldots \circ f_{tl})}_{k\ times}(q_{0tl}, t, 0) = q_{mtl} \quad (2.1)$$

$$\forall i, j \leq k-1, i \neq j, \underbrace{(f_{tl} \circ \ldots \circ f_{tl})}_{i\ times}(q_{0tl}, t, 0) \neq \underbrace{(f_{tl} \circ \ldots \circ f_{tl})}_{j\ times}(q_{0tl}, t, 0) \quad (2.2)$$

In relation 2.1 it can be seen that by applying k times the transition function to the initial state, i.e. k times executing task t, the marked state will be reached. Relation 2.2 represents that by applying i times and j times transition function on the initial state, different states will be reached. In other words, transitions of the model are in a sequence manner and the automata passes every state exactly one time.

The aim of the scheduling problem is to calculate the makespan. A makespan is the global completion time of all the tasks. In this study, it is assumed that every task is executed only once. For the sake of simplicity, in the rest of the chapter, TL^1 is written as TL. Figure 2.3 represents the generic pattern of a TL automaton triggering task t once. Since the duration of the transition in this automata is zero time units, for the sake of simplicity, the label of the zero duration can be removed from the model and only the name of the task be labeled on the transition (Figure 2.4).

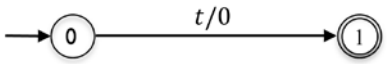

Figure 2.3. Modeling pattern of weighted TL automaton for $k = 1$

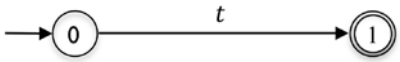

Figure 2.4. Simplified modeling pattern of weighted TL automaton for k=1

Example 1 (continue): Let us demonstrate the TL automaton model on Example 1. The result is displayed in Figure 2.5. In this figure, the models for launching task A, B and C correspond respectively to $G3, G4$ and $G5$ respectively.

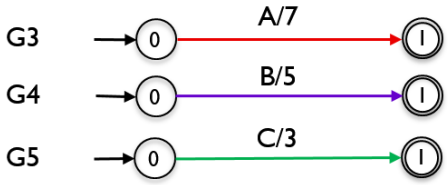

Figure 2.5. TL automata in Example 1

c. The task precedence automaton model

In the proposed framework, two types of precedence constraints have been identified. The first type, considers precedence constraints for delay between start times of tasks and is denoted as *delay precedence constraint*. For example, task B should be started 2 time units after starting A. This delay can be equal to the duration of tasks, e.g. duration of A. The second type is a particular case of the first type. This constraint considers precedence constraints for start times of tasks and is denoted as *triggering precedence constraint*. For example, if the precedence constraint is like $A \rightarrow B$, task B should be started at the same time or within an arbitrary delay after starting B. In this model, precedence constraints between tasks are described as sequences of transitions that follow precedence rules.

Definition 2.5 (formal model of the precedence automata): A precedence automaton is a subset of G_p automata set. This automaton model is defined as $PR = (Q_{pr}, \{q_{0pr}\}, T_{pr}, del, f_{pr}, \{q_{mpr}\})$ where

- Q_{pr} is the set of states,
- $q_{0pr} \in Q_{pr}$ is the initial state which is not a marked state,
- $T_{pr} \subseteq T$ is a set of tasks for which a total order is defined by the transition function f_{pr},
- $q_{mpr} \in Q_{pr}$ is the marked state,

- $del: T_{pr} \rightarrow \mathbb{R}_{\geq 0}$ assigns the amount of delay time between starting time of the task $t \in T_{pr}$ and starting time of its next task,
- The transition function is defined as $f_{pr}: Q_{pr} \times T_{pr} \times \mathbb{R}_{\geq 0} \rightarrow Q_{pr}$ that associates to a task $t \in T_{pr}$ and a state $q \in Q_{pr}$, a reaching state $q' \in Q_{pr}$ by a transition where this transition has a duration equal to $del(t)$. In addition, the following predicate holds:

$$t \rightarrow t' \; iff \; \exists q, q', q'' \in Q_{pr} \; s.t \; f_{pr}(q,t) = q' and \; f_{pr}(q',t') = q'' \qquad (2.3)$$

where $t \rightarrow t'$ means task t' should be started after task t within a delay of $del(t)$ time units. This predicate indicates that transitions of the model are in a sequence manner and the automata passes every state exactly one time.

Figure 2.6(a) and Figure 2.6(b) represent the generic pattern for modeling delay and triggering PR automata respectively. Same as TL automaton, since all weights in a triggering PR automaton are equal to zero, this model is simplified and zero weights are not labeled on the transitions.

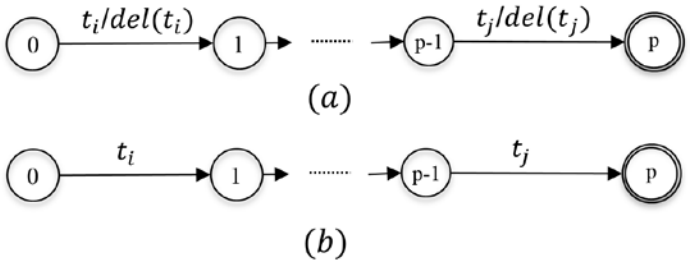

Figure 2.6. Modeling pattern of a delay PR automata (a) and a triggering PR automata (b)

Example 1 (continue): Let us apply the proposed PR automaton model to Example 1. Figure 2.7 shows weighted triggering PR automaton where start time of A should be less than or equal to start time of B. As demonstrated, only labels of symbols are put on the transitions which are in a sequence manner. While, in Figure 2.8 for modeling the delay PR automaton, on each transition, in addition to the name of the task, the delay duration for starting the next task is labeled on the transitions.

Figure 2.7. Weighted triggering PR automata in Example 1

Figure 2.8. Weighted delay PR automata in Example 1

d. Expected global behavior of the problem in Example 1

The global behavior of the system is the composition of all the automata where all transitions with the same label should be synchronized. A possible schedule is defined as a trajectory from the composed initial state to the composed marked state.

In this section, the global behavior of Example 1 is explained and illustrated in Figure 2.9. This system is synchronous composition of $G1$ to $G5$ which are depicted in Figure 2.2 and Figure 2.5 and doesn't consider PR automata. The composed initial state is the composition of all local initial states, and the composed marked state is the composition of all local marked states. Therefore, if all the local automata are in their initial state (marked state), the composed automata is in the composed initial state (marked state). Figure 2.9 shows that there are two possible schedules due to existing two traces from the initial state to the marked state. At the beginning, all automata are in their local initial states, so the composed automata is in the composed initial state. In the upper trace, firstly task B is done in 5 time units. Then, tasks A and C are done in parallel. After 3 time units from their beginning, C is finished and it remains 4 time units for finishing A. By elapsing 4 time units, A finishes and thereby, all the automata reach their local mark states and the composed state will be the marked state. This state is where all the tasks are done one time.

The lower trace can be explained same as the upper trace; the only difference is that in this schedule, first tasks A and C are done in parallel and then task B is done.

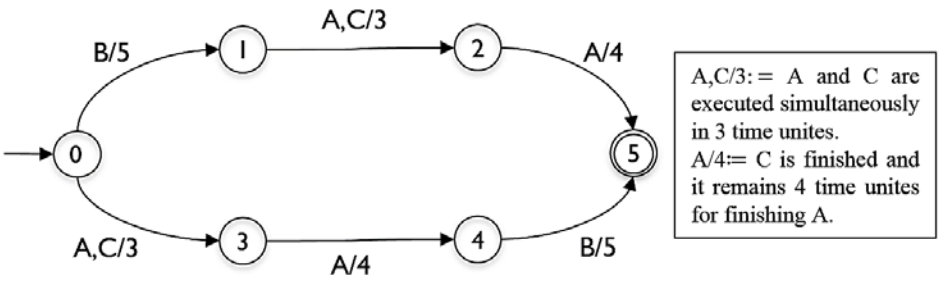

Figure 2.9. Global behavior of the Example 1(G1||G2||G3||G4||G5)

The behavior of the above-mentioned system by applying both types of precedence orders in Figure 2.7 and Figure 2.8 will be the same and like Figure

2.10. As it is obvious, the second trajectory will be no more possible since *A* should be executed after starting *B*.

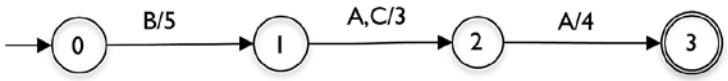

Figure 2.10. Global behavior of Example 1 considering PR automata (G1||G2||G3||G4||G5||G6)

e. Weighted automaton properties for the scheduling model

In this section, basic structural properties of the WA model is investigated. The rules for verification of structural properties of WA are same as conventional automata (Cassandras, Christos G., Lafortune 2008). Since the model of the MRS scheduling problem differs depending to the number of tasks, these properties are verified for the models of Example 1.

- *Definition 2.6 (Reachable automaton):* automaton *G* is reachable if there is a path in state transition diagram of the automaton from the initial state to every state.
  It is obvious from Figure 2.11 that in all models, all states are reachable from the initial states (i.e. states named 0).

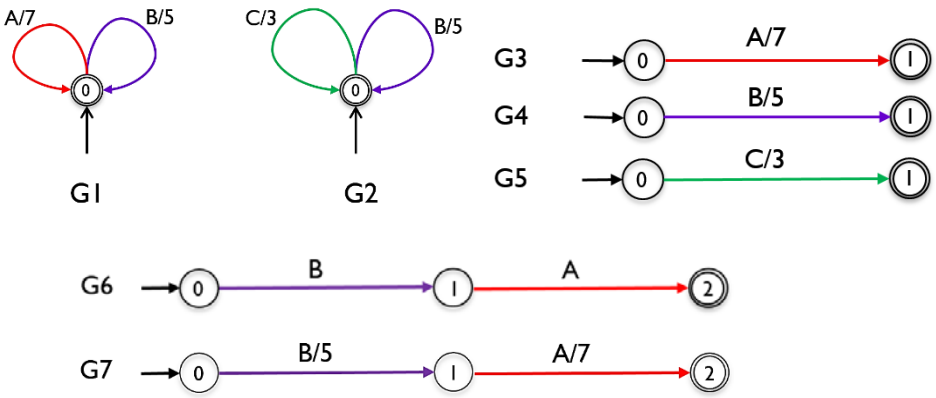

Figure 2.11. WA models of Example 1

- *Definition 2.7 (Co-reachable automaton):* automaton *G* is co-reachable if there is a path in state transition diagram of the automaton from any state to the marked state.
  In Figure 2.11, marked states are shown with double circles. It is clear that in all models, there is a path from all states to the marked states.
- *Definition 2.8 (Non-blocking automaton)*: automaton *G* is non-blocking if each state is reachable and co-reachable.
  Therefore, all the WA models are non-blocking.

2.4 Solving MRS scheduling problem by means of translating weighted automata models into timed automata models

The previous section defines WA modeling frameworks for defining scheduling problems. In the introduction section, it is explained that for obtaining a schedule, it is needed to perform a parallel composition on automata models of the scheduling problem, Parallel composition of a set of automata enables us to compose and explore them. Whereas, the existing parallel composition definitions for WA either don't contain a trajectory with the optimum makespan to be used as the optimal schedule or are not capable of illustrating simultaneous execution of tasks which is a non-separable issue from scheduling. Therefore, WA models cannot be explored and analyzed directly. Hence, after modeling the problem by WA, one solution is to translate the WA model to another type of automata for which an efficient definition of parallel composition exists. Then, compose the automata model through that definition and analyze the new model. Timed automata formalism has this characteristic and furthermore, there exist formal verification tools for timed automata that can be used to compose and analyze timed models automatically. Thus, in this thesis, the WA models are translated into timed automata. Thence, after translating the WA models to timed automata model, they are implemented in a formal verification tool named UPPAAL and the optimum makespan are found automatically. Translation of the models in the previous section are done through certain rules that are expressed in this section.

In the sequel, firstly definition of timed automata is recalled and afterwards, procedure of translating WA model to timed automata model is explained.

Definition 2.9 (timed automata): A timed automaton (TA) is defined as tuple $G = (L, l_0, C, A, E, I, V, V_0)$, where L is the set of locations, $l_0 \in L$ is the initial location, C is the set of clock, A is a set of actions, $E \subseteq L \times A \times B(C) \times 2^c \times L$ is a finite set of edges that each edge contains a source location, a set of actions, a set of guards, a set of clocks to be reset, and a target location. $I: L \rightarrow B(C)$ assigns clock constraints called invariants to locations (Behrmann, David, and Larsen 2006). An invariant is an inequality that shows the maximum time that automaton can stay in a location. After reaching this time limit, the automaton should change the state. V is the set of integer or Boolean variables or the output of functions that are defined over variables. They can also be updated or incremented on the edges. Furthermore predicates can be used over these variables as guards on the edges of the automaton. V_0 is the initial values of them (Bengtsson et al. 2004).

Each *state* of an automaton consists of a pair $\langle l, cl\rangle$ where l is a location and cl is the values of the clocks. The initial location of automaton G is $\langle l_0, cl_0\rangle$ where cl_0 assigns zero to all clocks in C (Behrmann, David, and Larsen 2006).

There are two kind of *steps* in an automaton:

- *Discrete step*: $\langle l, cl\rangle \xrightarrow{a} \langle l', cl'\rangle$ where in each transition $l \xrightarrow{a,g,r} l'$,$\rightarrow$ means a transition from location l to l' such that u satisfies g, all of the clocks in r are set to zero, and cl' satisfies $I(l')$ which is the invariant of the location l'. Note that this kind of step doesn't take time.
- *Timed step*: $\langle l, cl\rangle \xrightarrow{d} \langle l, cl + d\rangle$ that takes d time units where $d \in \mathbb{R}_+$ and $cl + d$ satisfies $I(l)$ (Yasmina Abdeddaïm, Asarin, and Maler 2006).

A *trajectory* of automaton $G = (L, l_0, C, A, E, I)$ is a possibly infinite sequence of steps starting from the initial state $\langle l_0, cl_0\rangle$ (Yasmina Abdeddaïm, Asarin, and Maler 2006; Bengtsson et al. 2004):

$$\xi: \langle l_0, cl_0\rangle \xrightarrow{d_1}\xrightarrow{a_1} \langle l_1, cl_1\rangle \xrightarrow{d_2}\xrightarrow{a_2} \langle l_2, cl_2\rangle \xrightarrow{d_3}\xrightarrow{a_3} \dots \tag{2.4}$$

and it's duration is obtained by sum of durations of all timed steps: $d_1 + d_2 + d_3 + \cdots$.

Definition 2.10 (schedule): A schedule is a trajectory that starts from the initial state, where no task is started, and ends in a state in which all tasks are completed respecting task precedence constraints. The goal of this chapter is to find a schedule with the minimum duration time.

Definition 2.11 (network of a set of TA): A network of a set of TA $G_i = (L_i, l_i^0, C, A, E_i, I_i, V_i, V_i^0), i = 1, \dots, n$ is the synchronous composition of them which is defined as $G_1 || \dots || G_n = \big(L_1 \times \dots \times L_n, (l_{1,0}, \dots, l_{n,0}), C, A, E, I, V, \big)$, where $\bar{l} = (l_1, \dots, l_n),\ C = C_1 \cup \dots \cup C_n,\ I\big(\bar{l}\big) = \wedge_i I_i(l_i),\ V = V_1 \dots \cup V_n$ and $V_0 = V_{1,0} \cup \dots \cup V_{n,0}$. Timed step rules are similar to the case of single TA but with the new invariant. Despite single TA, there are two rules for discrete steps in network of a set of TA. The first one is for defining local and individual actions where one of the automata moves on its own that is named *individual discrete step*. Another one defines synchronizing actions when two automata synchronize on a channel and move at the same time and is called *synchronization discrete step*. Let $\bar{l}[l_i'/l_i]$ denote the vector that ith element l_i of $\bar{l}$ is substituted with l_i'. Taking a step is based on following rules (Panek,

Stursberg, and Engell 2006; Behrmann, David, and Larsen 2006; Bengtsson et al. 2004):

- Timed step: $\langle \bar{l}, cl \rangle \xrightarrow{d} \langle \bar{l}, cl + d \rangle$ where $d \in \mathbb{R}_+$ and $cl + d$ satisfies $I(\bar{l}) = \wedge_i I_i(l_i)$
- Individual discrete step: $\langle \bar{l}, cl \rangle \xrightarrow{a} \langle \bar{l}[l_i'/l_i], cl' \rangle$ if there exists $l_i \xrightarrow{a,g,r} l_i'$ s.t. cl satisfies g, all of the clocks in r are set to zero, and cl' satisfies $I(\bar{l}[l_i'/l_i])$. Note that a corresponds to a local action in one automaton.
- Synchronization discrete step: $\langle \bar{l}, cl \rangle \xrightarrow{a} \langle \bar{l}[l_j'/l_j, l_i'/l_i], cl' \rangle$ if there exist $l_i \xrightarrow{c?,g_i,r_i} l_i'$ and $l_j \xrightarrow{c!,g_j,r_j} l_j'$ s.t. cl satisfies $g_i \wedge g_j$, all of the clocks in $(r_i \cup r_j)$ are set to zero, and cl' satisfies $I(\bar{l}[l_j'/l_j, l_i'/l_i])$. Note that $C!$ and $C?$ correspond to an action and a co-action respectively. In continue C is called a communication channel or signal (Behrmann, David, and Larsen 2006). Figure 2.16 and Figure 2.18 show an example of five automata that synchronize on channels A, B, and C.

Hereafter, procedure of translating WA models to TA models is explained. For this purpose, firstly three rules are defined to translate WA transitions to TA. Then, WA models are translated to TA according to these rules.

2.4.1 Translating transitions of the WA model to TA

Translation of WA to TA is not trivial, since firstly there is no weight in TA and weights should be simulated by adding a clock to each automaton; and secondly, synchronization of more than two transitions inside TA is not blocking. It means that if in one local automaton a synchronized transition is taken, there might be situations where the the transitions modeling the same action in other automata don't synchronize with it. Therefore, shared actions should be modeled in a special way.

The translation process of transitions relies on the following three rules:

Rule i: In a specification WA, if there exists a transition with duration, it will be translated as Figure 2.12:

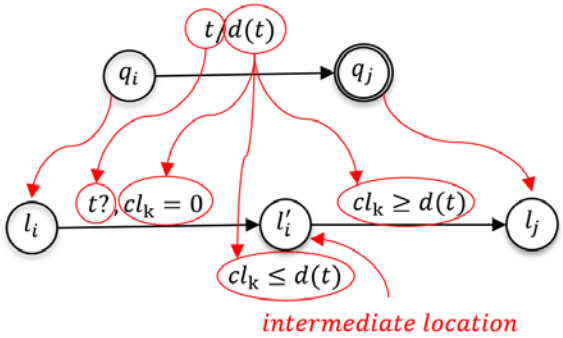

Figure 2.12. Translation of a transition with duration from a specification WA to TA

In the above figure, t repserents a task with duration $d(t)$. As illustrated in this figure, source and target states are translated without any change. In fact, the changes in the name of source and target of a transition are optional. A state q in WA is a single location, while a state q in TA is equal to a pair consisted of location and clock $\langle l, cl \rangle$. Therefore, for translation to TA, locations are only renamed to be distinguished from states. The single task transition is translated to a double-task-transition. Thereby an intermediate location l'_i is added where the automaton should wait a duration of time equal to $d(t)$. To ensure this purpose, an invariant is assigned to l'_i which is defined by $cl_k \leq d(t)$ and means the local clock of the kth automaton, cl_k, is not allowed to be bigger than $d(t)$. In other words, this invariant doesn't let the automaton to stay more than $d(t)$ time units in l'_i. Furthermore, a guard defined by $cl_k \geq d(t)$ is associated to the outgoing transition from l'_i to prevent changing the location before $d(t)$ time units. Moreover, label of the transition is translated to a receiving communication channel on the first transition. This transition will receive a communication signal from the plant automaton and will be synchronized with its transition.

Besides, marked locations are not defined in TA. For this reason, when doing the scheduling, marked locations should be determined and verified to be reachable.

Rule ii: In a specification WA, if there exists a triggering transition, it will be translated as follows:

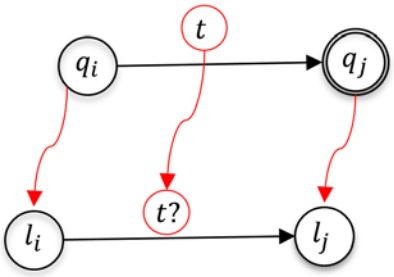

Figure 2.13. Translation of a triggering transition from a specification WA to TA

Triggering transitions don't take time. As it can be seen in Figure 2.13, source and target locations are translated without any changes (the simple changes in their names are optional). Whereas label of the transition is translated to a receiving communication channel. Thereby, whenever this transition receives a communication signal from a plant automata, it will be executed. l_j should be verified as a reachable location as well.

Rule iii: Triggering transitions in a plant automaton will be translated as:

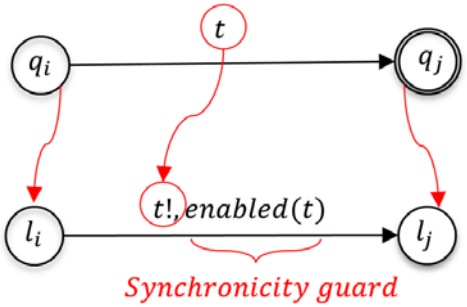

Figure 2.14. Translation of a transition from a plant WA to TA

In this type of transition, source and target locations are translated without any change (the simple changes in their names are optional). While as it is shown in Figure 2.14, label t of the transition is translated to $t!$ which is a sending communication channel. This transition broadcasts signals to specification automata to synchronize with the transitions that are receiving the signal.

In WA, all transitions with the same label in different components synchronize together (Lahaye, Komenda, and Boimond 2015; Landau et al. 2013). Whereas, in translating WA to TA, transitions with durations in specification automata should be translated to two transitions. Hence, it should be ensured that in executing a task of a timed TL automaton, all ME automata that share the same task are ready in initial location to synchronize their corresponding transitions. Therefore, a guard function $enabled(t): \Sigma \rightarrow Bool$ is associated to the transition of plant automaton that is true if and only if the following condition holds:

$$\forall i \in [1..N]: t \in TM_i \rightarrow \forall t' \in TM_i \backslash \{t\}: \neg pending(t', l_k) \quad (2.5)$$

where $pending: TM \times L_{me} \rightarrow Bool$ is defined as

$$pending(t, l_k) = \begin{cases} false, & l_k = l_0 \\ true, & otherwise \end{cases} \quad (2.6)$$

where N is the number of ME automata, t is the name of the task, TM_k is the set of tasks engaged in kth ME automaton and l_k is the current location and l_0 is the initial location of kth ME automaton. In the above equation, pending function verifies if a ME automaton is in a pending location where a task is during execution. This guard will be clarified in the next section.

In the following section, WA models (ME, TL and PR automata) are translated into TA models by applying proposed transition translation rules.

2.4.2 Translating WA models to TA models

a. Translating WA model of mutual exclusion automata to TA :

The only location of the WA, i.e. q_0, is translated to l_0 and remains the initial location. Note that in TA initial location should be displayed by double circles. Transitions are translated by following rules expressed in previous section as well.

In Figure 2.15, modeling pattern of a translated ME automaton is depicted. As tasks are modeled as loops, after completing all the tasks, all ME automata will reach their initial locations and don't have to stay in other locations. Figure 2.15 indicates that tasks $\{t_i, t_j, t_k, \dots, t_z\}$, which are presented as communication signals, belong to the demonstrated automaton.

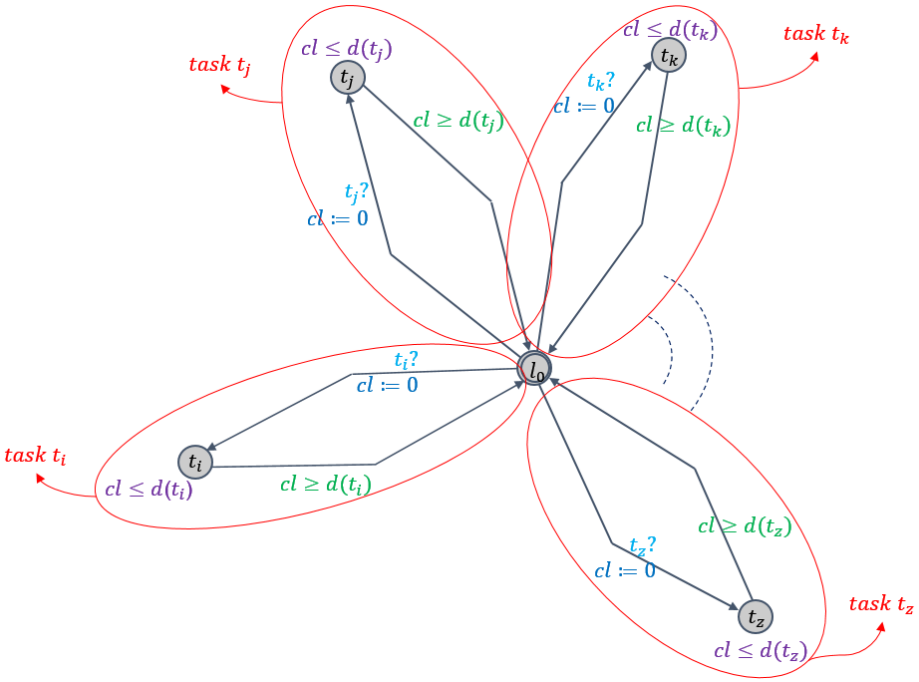

Figure 2.15. Modeling pattern of timed ME automaton

By applying aforementioned rules to WA model of Example 1, its timed ME automata will be obtained as Figure 2.16.

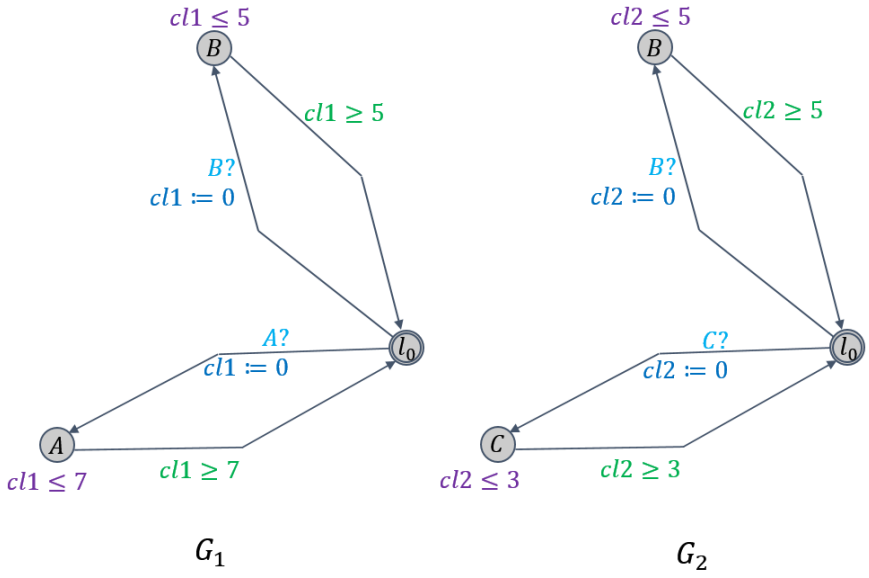

Figure 2.16. Timed ME automaton in Example 1

b. Translating WA model of task launcher automata to TA

From the Figure 2.3 it can be noted that to compute the makespan of a system, every weighted TL automaton should be composed of one single transition. This transition can be translated to TA following the explained rules. Figure

2.17 displays the modeling pattern of a TL automaton. This figure illustrates that when the launching signal of the task is sent, the automaton reaches location f. Hence, in order to make sure that a schedule is done, reachability of this location should be verified for all TL automata.

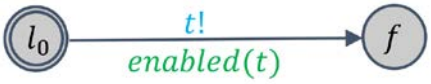

Figure 2.17. Modeling pattern of timed TL automaton

TA model of TL automaton related to Example 1 is shown in Figure 2.18.

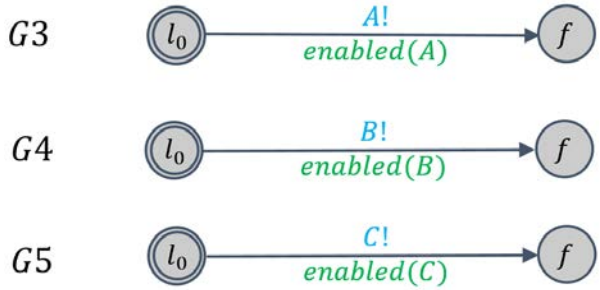

Figure 2.18.Timed TL automata of tasks A, B and C in Example 1.

According to Figure 2.16 and Figure 2.18, for taking the transition in the TL automata of task B, $enabled(B)$ verifies validity of this predicate:

$$B \in TM_1 \wedge B \in TM_2 \rightarrow \neg pending(A, l_1) \wedge \neg pending(C, l_2) \quad (2.7)$$

Where TM_1 and TM_2 are the set of mutual exclusions in automata $G1$ and $G2$ and l_1 and l_2 are current locations in $G1$ and $G2$ respectively.

$$pending(A, l_1) = \begin{cases} false, & l_1 = l_0 \\ true, & o.w \end{cases} \quad (2.8)$$

$$pending(C, l_2) = \begin{cases} false, & l_2 = l_0 \\ true, & o.w \end{cases} \quad (2.9)$$

which means that ME automata $G1$ and $G2$ are not in pending locations and during execution of tasks A or C respectively. Therefore, if predicate 2.7 is true, task B can be execute.

c. Translating WA model of precedence automata to TA

The triggering precedence WA is composed of triggering transitions so as to request starting of tasks in a specific order. By taking into account presented translation rules, the resulting TA model is obtained as shown in Figure 2.19.

Figure 2.19. Modeling pattern of triggering precedence timed automaton

The triggering precedence timed automaton of Example 1 is depicted in Figure 2.20.

Figure 2.20. Triggering precedence timed automaton in Example 1

The delay precedence WA model features transitions with durations. According to the first translation rule, TA pattern of this model is obtained as Figure 2.21. From this figure it can be seen that tasks $\{t_i, t_j, \dots, t_k\}$ belong to this set of precedence constraint. Two locations are assigned to each task (e.g. t_j). The first location (e.g. l_2) is where the automaton waits for receiving signal t from the TL automaton of task t. The second location (e.g. l_3) corresponds where it waits $del(t)$ time units. By receiving signal t, clock is reset and the automaton reaches the second location (e.g. l_3). After $del(t)$ time units, task t finishes and the automaton changes the location in order to wait for receiving launching signal from TL automata of the next task. If the precedence is correctly followed, automaton will reach location f. This implies that while sending communication signal of every task by its TL automaton, PR automata should be at a location from which a transition with the receive-action $t?$ can be taken. Thereby, the automata will be able to receive the $t?$ synchronization.

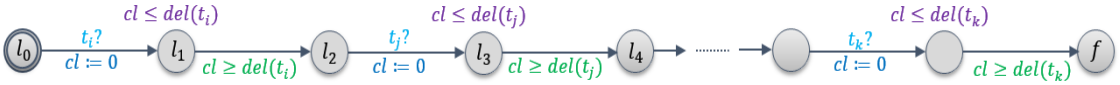

Figure 2.21. Modeling pattern of a delay precedence timed automaton

The delay precedence WA in Example 1 is depicted in Figure 2.22. In this automaton all the delays between tasks are assumed to be equal to the duration of tasks.

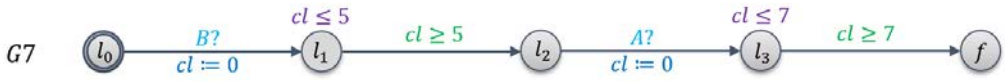

Figure 2.22. Delay precedence timed automaton in Example 1

It should be mentioned that in Figure 2.17, it is clear that verifying the guard of the TL automata needs accessing to current locations of ME automata. Since this issue cannot be implemented directly in a formal verification tool, some technical modifications are needed in order to simulate this guard and are explained in Appendix A.

d. TA structural properties for the scheduling model

In this section, basic structural properties of the TA models are investigated (Bornot, Gößler, and Sifakis 2001). Since the model of the MRS scheduling problem differs depending to the number of tasks, these properties are verified for the models of Example 1.

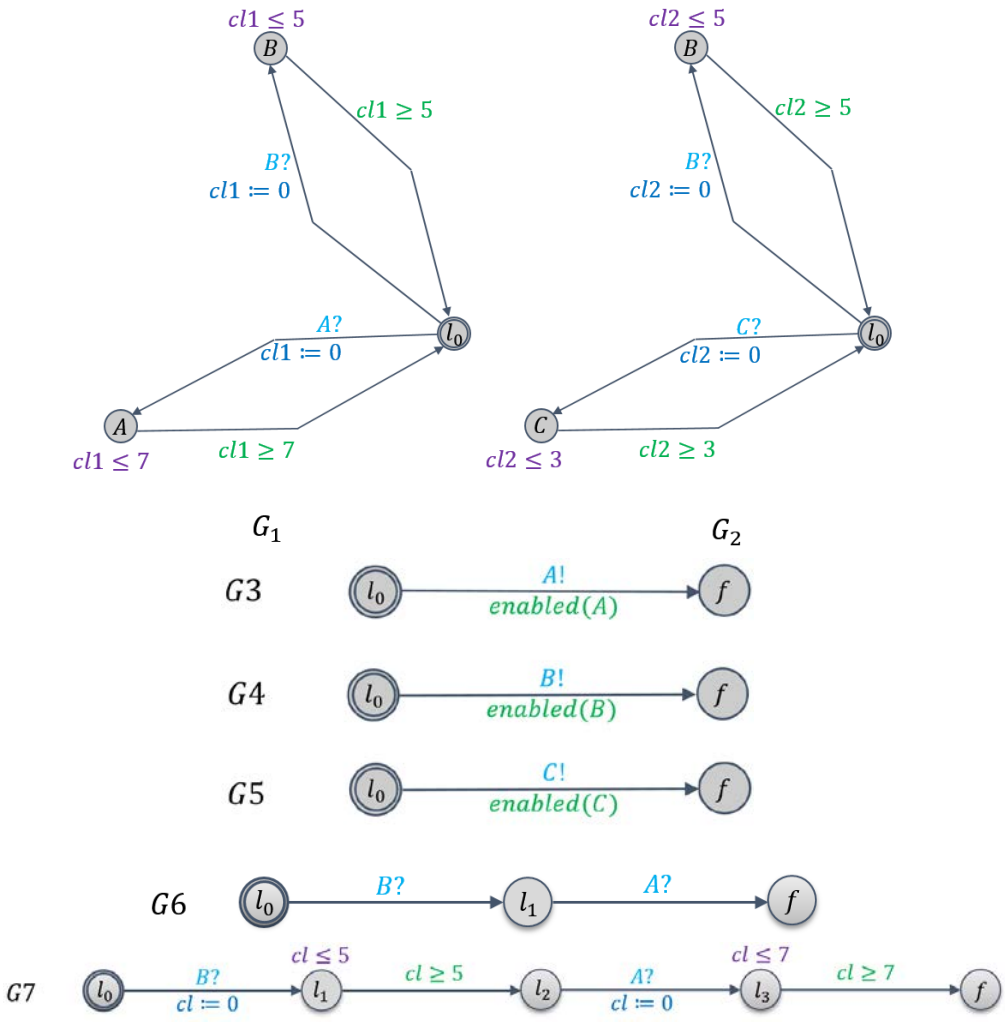

Figure 2.23. TA models of Example 1

- *Definition 2.12 (Timelock-freedom):* A TA is timelock-free if any trajectory $\langle l_0, cl_0\rangle \xrightarrow{d_1} \xrightarrow{a_1} \langle l_1, cl_1\rangle \xrightarrow{d_2} \xrightarrow{a_2} \langle l_2, cl_2\rangle \xrightarrow{d_3} \xrightarrow{a_3} \ldots$ diverges.
- *Definition 2.13 (livelock-freedom):* A timed system is livelock-free if in any trajectory, some action occurs infinitely often.
  According to the Figure 2.23, after finishing tasks and when TL automata and PR automata are reached to location f, all the trajectories terminate and the automaton reaches a deadlock where the automaton cannot move anymore. While, this deadlock state is the

goal state in the scheduling problem. In fact, continuation of the trajectory is not intended in this thesis. Therefore, the model is neither timelock-free nor livelock-free.

- *Definition 2.14 (Liveness):* An automaton is called live if it is not timelock-free and livelock-free.
  Thence, the TA model is not live. This fact happens because the scheduling model is supposed to compute the makespan for which every task should be done only one time. Therefore, the model is acyclic and in fact, this property is important for cyclic models.

2.4.3 Scheduling approach

In this section, an efficient solving approach is presented to solve the problem through TA models.

a. General principle of solving approach

In order to schedule by TA, a trajectory in the model should be found in which all the tasks are done. In other words, the scheduling problem amounts to a formal verification problem to verify a property that guarantees that all the tasks are eventually done. In fact, this property expresses the reachability of a set of locations where the condition to reach this set is to complete all the tasks. By verifying this property, a witnessing trajectory will be obtained that indicates an order of tasks and a makespan for doing all of them.

b. Model checking tools

In this section, UPPAAL and Kronos are introduced as two well-known model checking tools for timed systems. Generally, model checking tools of timed systems support TA of Alur and Dill or an extension of that model as the description language. Also all of them use TCTL logic or a fragment of it.

- *Kronos*

Kronos is a tool developed with the aim to verify complex real-time systems. It is developed at Verimag, a joint laboratory of UJF, Ensimag and CNRS. It supports full TCTL language. It allows on-the-fly analysis for reachability properties as well as forward and backward searching algorithms (Wilson and France 2000). Kronos uses a very restricted data type that allows only declaration of clock variables. Whereas it has been extended to several successors such as Open-Kronos that provides a more convenient modeling language (e.g., with discrete variables). Kronos is freely available for academic users at http://www-verimag.imag.fr/DIST-TOOLS/TEMPO/kronos.

- *UPPAAL*

UPPAAL is an integrated tool environment for verification of real-time systems modeled. It is developed in collaboration between the Department of Information Technology at Uppsala University, Sweden and the Department of Computer Science at Aalborg University in Denmark. The tool supports an extension of TA with additional features such as integer variables, structured data types, user defined functions, and channel synchronization. One of the technical differences between Kronos and UPPAAL is that in UPPAAL, for synchronization of more than three transitions, it is needed to use a special type of non-blocking synchronization named broadcast channels. Synchronization with a channel can be realized by a sending signal $a!$ and some receiving signals $a?$. Being non-blocking is in the sense that if a transition emits a sending signal, then this emission can be performed even when none of related receiving transitions in other processes are activated. While in Kronos synchronization process is blocking for any number of synchronized transitions. UPPAAL uses a fragment of TCTL language which is restricted to properties based on reachability analysis (Wilson and France 2000). The verification process in this tool is based on on-the-fly analysis which is a powerful technique (Larsen, Pettersson, and Yi 1997).

Two indicators are considered to use UPPAAL 4.1.19 as a verification tool in this thesis. First, the last version of this software is released recently and the team developer of this software is active to fix potentially existing bugs in the resealed software. Second, this tool has an available contact center to answer potential questions of users. The tool is available for free for academic users at http://uppaal.org.

c. Schedule generation by model checking

In the proposed algorithm, to find a schedule, UPPAAL is used as a model checker to explore the state space for determining if there is a trajectory through which all the tasks are done. For this purpose, a safety property will be verified by performing a reachability analysis. This analysis concerns reachability of all the ME automata to their initial locations and all TL and PR automata to their final locations (f). In TCTL language this property can be formalized as the following:

$$P: \exists \Diamond ((\Lambda_{1\leq j\leq m} ME_j.l_{0j}) \wedge (\Lambda_{1\leq i\leq n} TL_i.f_i) \wedge (\Lambda_{1\leq k\leq p} PR_k.f_k)) \quad (2.10)$$

where l_{0j} are the initial locations of ME automata, and f_i and f_k are final locations of TL and PR automata respectively. Generically $\exists \Diamond \beta$ means that some reachable states must satisfy β (Larsen, Pettersson, and Yi 1997).

In this algorithm, in each iteration, property P is verified following a different trajectory. By verifying the property, the model checker issues a witnessing trajectory that corresponds to one of the possible schedules. The maximum value of clock in each witnessing trajectory is equal to the makespan of the schedule. Different schedules are generated randomly by means of *random depth first searching* method. The first schedule will be generated and selected as the optimal schedule. Its makespan will be also selected as the optimal makespan. By generating the second schedule, its makespan will be compared with the optimal schedule. If it was less than the optimal makespan, it will be selected as the new optimal makespan and its schedule will be selected as the new optimal schedule. To find the optimal schedule, several iterations will be done. The iterating procedure will be continued in a predetermined time by generating new schedules and comparing to the optimal schedule. After reaching the predetermined time as stop criteria, a suboptimal schedule will be obtained. This algorithm is implemented in a bash file and is detailed in Algorithm 2.1.

It is noteworthy to mention that although there exist timed optimal reachability algorithms that can be used to find the optimal schedule (Peter Niebert, Tripakis, and Yovine 2000), since the automata should take all transitions to do all the tasks and reach the final location, this analysis is rather expensive. Therefore it is preferred to save the time and find the sub-optimal schedule.

The stop criteria for finding the optimal schedule could be either number of iterations or a boundary time. In order to obtain a schedule, it is more applicable in industry to set a predetermined time as the stop criteria. While for finding complexity of the model and algorithm, stop criteria should be a fixed number of iterations. Algorithm 2.1 uses the first criteria, i.e. boundary time, as the stop criteria.

Algorithm 2.1

Input: M ME automata and N TL automata, a time bound B
Output: SubOptimal Makespan (SOM), optimum trajectory in file OptimalTrajectory.

$//Initialize$:
$D \coloneqq set\ of\ duration\ of\ tasks$
$//search\ the\ optimal\ schedule\ in\ several\ random\ reachability\ analyses$:
$TM \coloneqq time$
$OptimalTrajectory \coloneqq a\ trajectory\ in\ UPPAAL\ such\ that\ P\ is\ satisfied$
$SOM \coloneqq makespan\ of\ the\ trajectory$
$while\ (time < B + TM)\{$
$trajectory \coloneqq a\ trajectory\ in\ UPPAAL\ such\ that\ P\ is\ satisfied\ and$

$compute\ makespan\ T$
$//(analysis\ is\ based\ on\ random\ depth\ first\ search)$
$if\ \ (T < SOM)\{$
$\ \ OptimalTrajectory := trajectory$
$\ \ SOM := T$
$\}end\ if$
$\}end\ while$

Since the output trajectory should be understandable for the decision maker, the obtained trajectory is implemented in the concurrent simulator of UPPAAL to obtain the Gantt chart of the schedule. In this way, concurrent execution of tasks can be visualized. Using this simulator, the waiting duration of every automaton in each pre-specified location can be displayed in the Gantt chart by a specified color. Furthermore, common tasks in different automata could be shown by the same colors. Therefore by assigning colors to task locations in ME automata, each bar in the Gantt chart would show the task performing moments. Bars with same colors would represent also common tasks in different ME automata.

Figure 2.24 demonstrates the Gantt chart of Example 1 without considering any precedence constraint. For making this Gantt chart, specific patterns are assigned to task locations A, B and C in which the automata stays during performing tasks. In this Figure, the upper bars belong to automaton $G1$ and the lower bars belong to $G2$. In the upper line, the first bar (▥) represents the moments of performing task A that lasts 7 time units. The first bar of the second line (▤) shows the performing moments of task C with 3 time units. As it is obvious, A and C are not in conflict with each other and hence can be executed simultaneously. When two tasks are executed simultaneously, UPPAAL performs them at the earliest time. For example if UPPAAL is going to execute task A between time 0 and 7, and C should be done in parallel, it starts C at time 0 and does not delay it to be started at time 3. The second bars at both lines (▩) represent performing moments of task B that lasts 5 time units. As it can be seen in the figure, two automata perform C synchronously from time 7 to time 12. So in this schedule, the makespan is equal to 12.

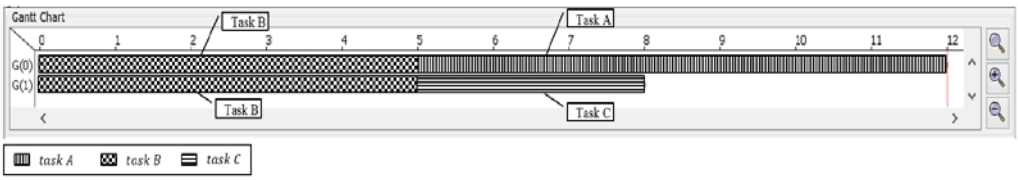

Figure 2.24. Gantt Chart of Example 1 without precedence conditions

Figure 2.25 shows the Gantt chart of Example 1 considering the delay PR timed automaton G7 which is depicted in Figure 2.22. As expected in automaton G7, despite the previous schedule, task *B* is done before task *A*. Whereas the makespan doesn't changes and remains 12 time units.

Figure 2.25. Gantt chart of Example 1 considering delay PR timed automaton G7

2.4.4 Complexity

In this section, the efficiency and applicability of the proposed model and solving approach are discussed. For this purpose, a number of problem instances are defined with different sizes that vary between 20 tasks in 5 ME timed automata to 220 tasks in 20 ME timed automata which is close to industrial size problems and offers a reasonable insight for evaluating the complexity trend. Then, all the problem instances are solved by applying the approach proposed in Algorithm 2.1. Referring to the previous section, the stop criteria for acquire computational time complexity of the model and algorithm should be a fixed number of iterations. Therefore, in this section, in order to find the complexity, the algorithm is adapted to find the optimal schedule with 100 iterations. It should be mentioned that this number doesn't have any impact on the efficiency of the proposed algorithm. In other words, by multiplying this number by any arbitrary number, the complexity of the algorithm doesn't changes. The algorithm is coded using bash file on Ubuntu 14.04 on a personal computer Core i5, 2.27 GHZ with 5.0 GB RAM. In order to show its efficiency, time complexity of the algorithm is compared to similar previous studies (Edis and Ozkarahan 2011; Kellerer and Strusevich 2008).

Table 2.2. calculated and real makespan and corresponding computational time for problem instances

# A.*	# Task	# Ind. tasks**	# Shared tasks between			Optimal Makespan	Obtained Makespan	Time (s)
			2A***	3A	4A			
5	20	20	0	0	0	60	60	6.47
		16	4	0	0	120	120	6.85
		12	4	4	0	180	180	7.64
		8	4	4	4	240	240	8.89
10	20	20	0	0	0	10	10	12.56

# A.*	# Task	# Ind. tasks**	# Shared tasks between			Optimal Makespan	Obtained Makespan	Time (s)
			2A***	3A	4A			
20	20	16	4	0	0	20	20	13.62
		12	4	4	0	30	30	15.25
		8	4	4	4	40	45	17.65
		20	0	0	0	15	30	27.41
		16	4	0	0	30	45	29.56
		12	4	4	0	45	60	34.08
		8	4	4	4	40	40	42.29
5	60	60	0	0	0	60	65	44.28
		48	12	0	0	120	120	46.53
		36	12	12	0	180	180	53.52
		24	12	12	12	240	240	64.05
10	60	60	0	0	0	18	21	78.18
		48	12	0	0	36	36	84.15
		36	12	12	0	54	57	95.79
		24	12	12	12	72	72	112.94
20	60	60	0	0	0	15	20	161.98
		45	15	0	0	30	35	178.76
		30	15	15	0	45	55	210.89
		15	15	15	15	60	65	265.47
5	100	100	0	0	0	100	110	113.42
		80	20	20	0	200	200	123.10
		60	20	20	0	300	300	142.32
		40	20	20	20	400	400	172.89
10	100	100	0	0	0	100	110	200.19
		80	20	20	0	200	200	218.61
		60	20	20	0	300	320	245.18
		40	20	20	20	400	420	292.57
20	100	100	0	0	0	50	60	404.17
		80	20	20	0	100	120	430.30
		60	20	20	0	150	180	491.06
		40	20	20	20	200	230	583.14
5	140	140	0	0	0	140	155	219.41
		112	28	0	0	280	280	238.01
		84	28	28	0	420	420	277.29
		56	28	28	28	560	560	339.71

# A.*	# Task	# Ind. tasks**	# Shared tasks between			Optimal Makespan	Obtained Makespan	Time (s)
			2A***	3A	4A			
10	140	140	0	0	0	140	160	319.50
		112	28	0	0	280	280	407.29
		84	28	28	0	420	450	466.64
		56	28	28	28	560	580	555.49
20	140	140	0	0	0	140	160	714.03
		112	28	0	0	280	320	764.67
		84	28	28	0	420	500	870.72
		56	28	28	28	560	660	1032.71
5	180	180	0	0	0	180	195	345.79
		144	36	0	0	360	360	376.85
		108	36	36	0	540	540	441.61
		72	36	36	36	720	720	544.15
10	180	180	0	0	0	180	200	596.60
		144	36	0	0	360	360	640.53
		108	36	36	0	540	590	730.66
		72	36	36	36	720	750	869.57
20	180	180	0	0	0	180	200	1155.16
		144	36	0	0	360	360	1193.98
		108	36	36	0	540	580	1277.14
		72	36	36	36	720	740	1402.16
5	220	220	0	0	0	220	240	467.75
		176	44	0	0	440	440	504.42
		132	44	44	0	660	660	604.88
		88	44	44	44	880	880	765.67
10	220	220	0	0	0	110	120	779.45
		176	44	0	0	220	225	844.04
		132	44	44	0	330	360	978.34
		88	44	44	44	440	445	1184.55
20	220	220	0	0	0	110	120	1487.03
		176	44	0	0	220	250	1605.81
		132	44	44	0	330	390	1842.95
		88	44	44	44	440	510	2226.59

: Number of ME automata* *: Number of individual tasks* ****: 2 ME automata*

In Table 2.2, the expected and sub-optimal makespan of problem instances are shown. Furthermore the computational time for the problem instances are

mentioned. This table show that the longest time for computing the makespan belongs to the problem instance with 220 tasks and 20 ME automata which is equal to 2226.59 seconds, i.e. about 37 minutes. This time is reasonable enough to obtain a schedule for a large size problem. Classification of problems relate to the number of tasks that ME automata share, number of ME automata and number of tasks.

Figure 2.26 illustrates the trend of calculation time against increasing number of tasks for three different numbers of ME automata. In these graphs, tasks are individual or common between two ME automata. This figure illustrates the obvious fact that increase in number of tasks and ME automata causes increase in calculation time. The trend-line of this increment is obtained by Excel software. This trend is polynomial and is demonstrated by dash-dotted lines that correspond exactly to the trend of the graphs. The term “Poly.” in the legend of the graph denotes the polynomial trends for each graph.

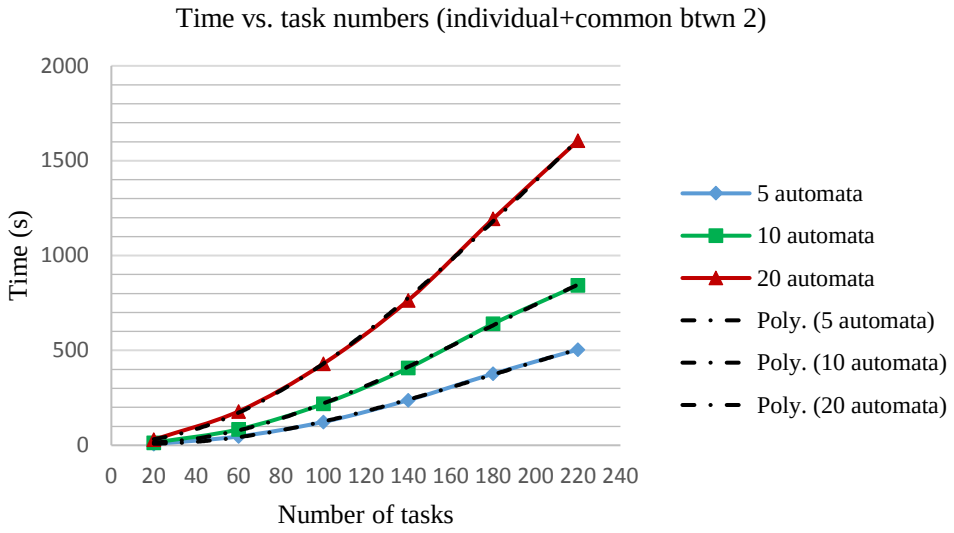

Figure 2.26. Calculation time vs. number of tasks for individual tasks and common tasks between 2 ME automata

Figure 2.27 illustrates variation of calculation time against increasing number of ME automata. This figure shows that in the problems with fewer tasks, as the number of conflicting sets of tasks (i.e. ME automata) increases, the computation time increases slightly. Whereas in problems with huge number of tasks, a small variation in number of ME automata causes a dramatic increase in computational time. For example, this figure shows that in the problem with 20 tasks, increasing number of ME automata from 10 to 20 ME automata, increases computational time 15.94 time units. Whereas in the problem with 220 tasks, it causes an increase of 761.77 time units.

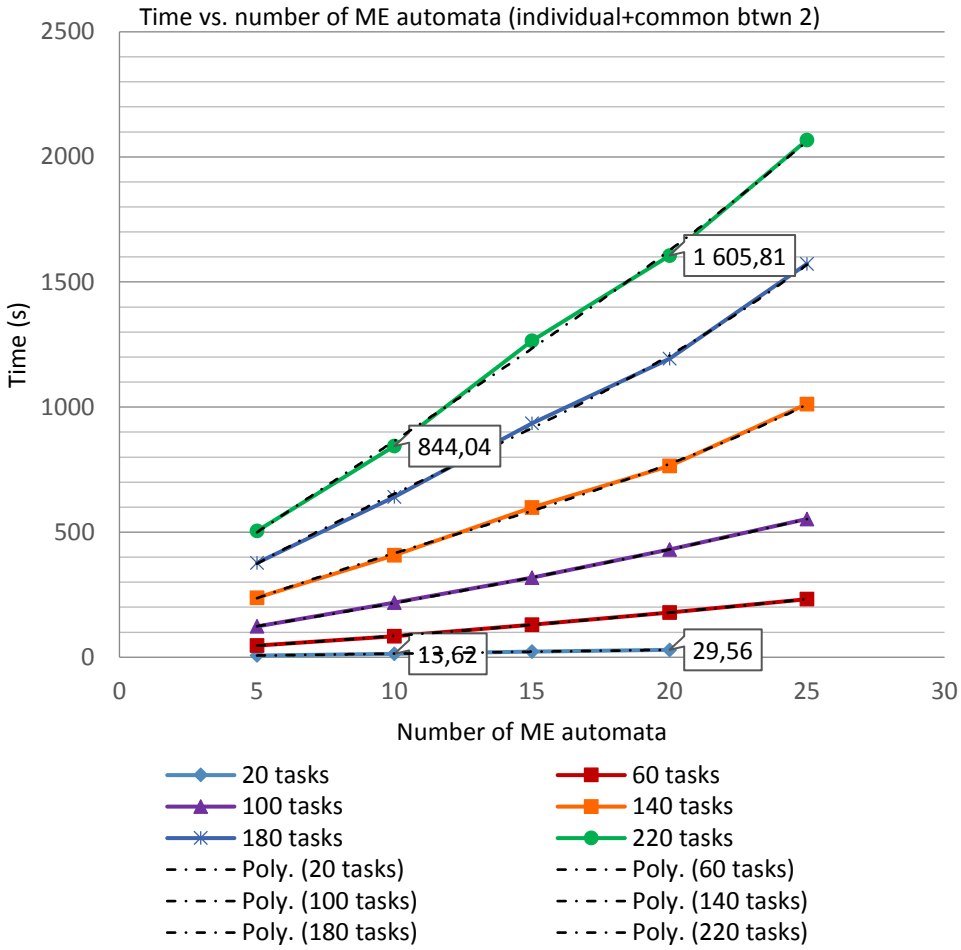

Figure 2.27. Calculation time vs. number of ME automata, for tasks that are individual or common tasks between 2 ME automata

In Figure 2.28, the calculation time respecting to number of tasks for problems with 5 ME automata featuring conflicts is demonstrated. This number is arbitrary and doesn't effect on the complexity of the graphs. In this figure, multiples graphs demonstrate different types of task-sharing. It can be seen that in addition to the impact of variation in number of tasks on calculation time, increasing number of task-sharing causes a significant change in calculation time. For example, in the problem with 220 tasks, by increasing the maximum number of ME automata in which every task is shared from 2 to 4, computational time increases 51.79%.

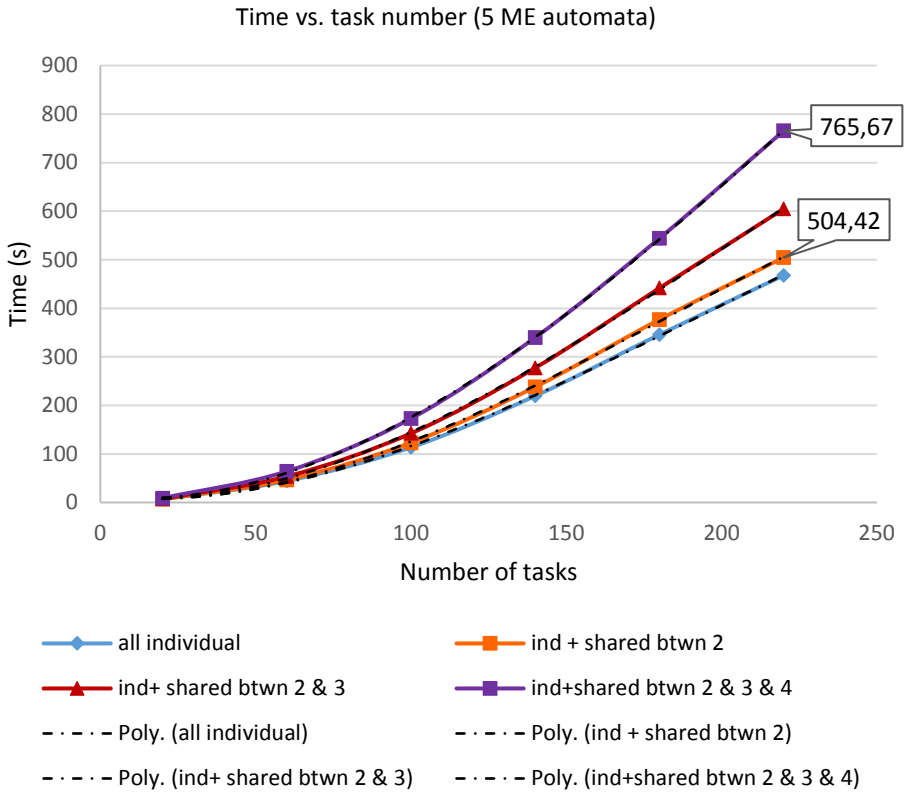

Figure 2.28. Calculation time vs. number of tasks for 5 ME automata

In similar studies, researchers classify this kind of problem as an NP-hard problem which trend of its computational time against size of the problem is adapted to an exponential function (Edis and Ozkarahan 2011; Kellerer and Strusevich 2008). This issue becomes important facing large size problems. In fact, having exponential complexity causes taking unreasonable time for obtaining the optimal schedule in industrial problems. Whereas, as mentioned earlier, in some industries it is preferred to save time and obtain a sub-optimal schedule. In this study, results show that trend of increments in computational time against increasing size of the problem follows a polynomial function. It means that in industrial cases, through this model and algorithm, the sub-optimal schedule can be obtained in a reasonable time.

2.5 Conclusion

In this chapter, a novel approach is presented to model and solve multi-resource sharing scheduling problem through WA. Furthermore, it focuses on proposing an efficient solving approach to reach a sub-optimal schedule in a reasonable computational time for industrial size problems. In fact, since MRS scheduling

problem is NP-hard, there were two choices; either to find the optimal schedule in a long and unreasonable time, or to save the time and find the sub-optimal schedule. The second option is preferred in this work.

Scheduling problems can be modeled by either weight-based automata like weighted automata, or clock-based automata like timed automata. In this chapter, it is shown that simpler and more abstract models can be built for the MRS scheduling problem through weighted automata. Whereas there are two issues for using current synchronous compositions. Either the minimum makespan cannot be obtained through using them, or they don't show a complete composed behavior of the components of the model. Thence, it is not possible to analyze (WA) models directly. Therefore, firstly the MRS scheduling problem should be modeled by WA. Then, in order to obtain the schedule, the proposed model is translated to TA. Afterwards, an algorithm is proposed to obtain the sub-optimal schedule by performing iterations of reachability analysis on the timed model using UPPAAL as a formal verification tool. The analyses are based on random depth first search.

The results show that the proposed model and algorithm can be efficiently applied to industrial sized problems with 220 tasks and 20 sets of task-conflicts in using various resources. It has been proved that time complexity of the proposed method is polynomial which allows the decision maker to solve an industrial size problem in reasonable time.

3

Synchronous composition of weighted automata - application to MRS scheduling

3 Synchronous composition of weighted automata - application to MRS scheduling

3.1 Introduction

In Chapter 2, the timed automata (TA) and timed model checking technique have been used as the core formalism and technology to deal with MRS scheduling problem. First, the scheduling problem is specified using weighted automata (WA). The semantics of a weighted automaton has been defined strictly locally, without any notion of compound behavior resulting from the concurrent operation of two or more weighted automata. Instead, rules have been defined in order to translate each weighted automaton model into a timed automaton. Thus, the consistency of the global WA model relies on the product of timed automata which is formally defined.

Even though the WA to TA translation rules are systematic, they require to handle TA-specific modeling mechanisms such as clocks, clock guards, invariants and communication channels. Considering the specific modeling needs in this work, such mechanisms are technical overload and hardly maintainable. Therefore, the proposed solution in chis chapter to overcome this intricacy is based on definition of a new synchronous composition for WA. Though this definition, the WA components can be composed directly and the analysis for finding the best schedule can be performed directly on the composed model.

Various investigations of automata compositions have been conducted, relying on different kinds of weighted automata and composition approaches. Moreover, several studies proposed time or cost optimal reachability analysis that can be inspired for the analysis on the new WA synchronous composition. Hereafter, these investigations are detailed.

3.1.1 State of the art on synchronous composition of weighted automata

Milner (1983) proposes a notion of synchronized product between automata. In this product, two or more automata may run concurrently and independently, but they must synchronize on a set of events called shared events. Individual actions in local automata are allowed to have an interleaving execution. There is also a notion of global sharing of actions among automata, i.e. an action is

shared between all automata and not some of them. Duration of actions cannot be handled through this model.

Komenda, Lahaye, and Boimond (2009) propose a synchronous product for multi-event (max,+) automata which correspond to a class of timed automata with serveral clocks. Komenda, Lahaye, and Boimond (2010) propose a synchronous composition model that results in multi-event interval weighted automata.

Su, Van Schuppen, and Rooda (2012) address a minimum-makespan supervisor synthesis problem. They propose a terminable algorithm to solve the problem and compute the execution time of each string by the theory of heaps-of-pieces. The authors also present a timed supervisory control map to implement the synthesized minimum-makespan sublanguage. In order to formulate the problem, they propose a synchronous product for WA. Komenda and Boimond (2012) proposes a modular approach for the modeling of discrete event systems using (max, +) automata. This method consists in decomposing the system into subsystems, each to be modeled by a deterministic (max, +) automaton. The interactions between these subsystems is made through common events occurring simultaneously in each component. A synchronous product is proposed to compose these components in which synchronization between common events is modeled.

Lahaye, Komenda, and Boimond (2015) propose a compositional modeling approach by means of (max, +) automata. Firstly the authors introduce modeling of safe timed Petri nets using (max, +) automata. Afterwards, two types of synchronous composition of (max, +) automata are proposed to model safe timed Petri nets. Furthermore, an asynchronous composition is introduced to represent particular bounded timed Petri nets. Sébastien Lahaye and Jean-Baptiste Fasquel have implemented a library in python software to model (max,+) automata synchronous composition (Lahaye, Komenda, and Boimond 2015). Makespan of different given orders of tasks can be obtained through this library. Quintero Garcia (2015) models mutual exclusions among tasks with underlying resource sharing conflicts as subsystems through local tropical automata (generalized version of (max, +) and (min, +) automata which are type of weighted automata). Moreover, the author describes fundamental aspects of a makespan minimization methodology considering minimization of idle times on resources.

Ware and Su (2017) present a method for finding a time optimal accepting trace for large DESs based on sequential language projection, and pruning. The algorithms are tested on a linear cluster tool to show their effectiveness. In this

article, a cluster tool is modeled through timed-weighted automata considering a specific type of synchronous product.

3.1.2 State of the art on time-optimal reachability analysis

Gaubert (1995) addresses performance evaluation of (max, +) automata in the worst, mean, and optimal cases. A simple algebraic reduction is provided for the worst case. Whereas, the author uses the Kolmogorov equation of a Markov chain and a Hamilton-Jacobi-Bellman equation to obtain the mean performance and optimal performances respectively.

Gerd Behrmann et al. (2001) present an algorithm for computing the minimum cost of reaching a goal state in a Uniformly Priced Timed Automaton (UPTA) with optimal number of explored states. This model is a sub-model of Linearly Priced Timed Automata (LPTA), which extends timed automata with prices on both locations and transitions. In fact, despite LPTA, the rate of prices associated to locations of a UPTA is uniform. The authors implement algorithms in tool UPPAAL which are based on branch and bound techniques that can be used for limiting the search space for finding the optimal solution faster. Larsen and Vaandrager (2001) introduces the model of linearly priced timed automata. The authors propose a minimum-cost reachability algorithm based on branch-and-bound technique for reaching from the initial state to the goal state. Interval weighted automata are defined as automata with weights in a product dioid. The formalism of this automata makes it possible to model temporal constraints for transitions instead of exact durations.

Alur, La Torre, and Pappas (2004) deal with the optimal-reachability problem for weighted timed automata. In this article, an approach is presented to reduce this problem to computing parametric shortest paths in a finite weighted directed graph. Complexity reduction techniques for this analysis is also presented.

Behrmann, Larsen, and Rasmussen (2005) apply timed automata technology to optimal scheduling and planning problems. They also implemented the problem in the tool Cora which is specialized for cost-optimal reachability for the extended model of priced timed automata. It is noteworthy to mention that Cora is an extended version of UPPAAL.

3.1.3 Synthesis of the state of the art

Table 3.1 demonstrates a classification for studies that propose synchronous or synchronized composition of automata. In this table, the composition proposed in the reviewed papers, are investigated from two point of view:

- Simultaneity: The composition can show simultaneous execution of different actions in different local automata
- Yielding minimum makespan: Minimum makespan can be obtained through the composition

The synthesis of this state of the art shows that the by using so far proposed synchronous compositions for WA, the models of the components of the scheduling problem cannot be composed. Among these reviewed articles, only four researches have proposed a new composition that can handle duration of tasks. Disadvantages of these compositions are not illustrating simultaneous execution of tasks or not containing a trajectory with the optimum makespan to be used as optimal schedule. For example, in one of the articles a synchronous composition for (max, +) automata is proposed. This composition could be employed for modeling of multi-resource sharing scheduling problems. Whereas, the minimum makespan cannot be obtained through this composition (Komenda, Lahaye, and Boimond 2009c). The same authors propose another type of synchronous composition for (max, +) automata. Despite the minimum makespan can be found through a reachability analysis on this composition, simultaneous execution of actions from different local automata cannot be shown (Lahaye, Komenda, and Boimond 2015).

Su, Van Schuppen, and Rooda (2012) propose a synchronous composition for WA. Yet, this composition does not feature the simultaneous behaviors of actions.

The approach proposed by Quintero (2015) is described on a particular example, and hence lacks generality. Besides, the time is handled exclusively by decomposing durations into discrete steps, which excludes almost always optimal solutions.

According to the enumerated issues, in this chapter a new synchronous composition of WA is proposed. The specificities of this composition is that firstly, in addition to interleaving of action, they can be also executed simultaneously and at the earliest time; and secondly, the optimal performance of the model representing the minimum makespan can be obtained through the composition. Afterwards, by inspiration from the performed literature review in Section 3.1.1, a time-optimal reachability algorithm is proposed to find the fastest trajectory to reach from the initial state to the marked state. This analysis can be directly used and can yield the timed-optimal schedule successfully.

Table 3.1. Classification of literature proposing synchronous composition of WA

Authors	Simultaneity	Minimum Makespan
(Milner 1983)		
(Komenda, Lahaye, and Boimond 2009c)	✓	
(Komenda, Lahaye, and Boimond 2010)	✓	
(Su, Van Schuppen, and Rooda 2012)		✓
(Lahaye, Komenda, and Boimond 2012)		✓
(Lahaye, Komenda, and Boimond 2015)		✓
(Ware and Su 2017)		✓

In this Chapter, all the problem hypotheses are the same as Chapter 2. The remainder of this chapter is organized as follows. Section 3.2 explains the concept of synchronicity and simultaneity in the composition of automata through an example. In section 3.3, a new synchronous composition is defined for WA. In section 3.4, a time-optimal reachability algorithm is defined in order to find the optimal schedule. Finally section 3.5 is devoted to conclusion.

3.2 Synchronous composition for weighted automata

Prior to define a new synchronous composition, let's explain the meaning of simultaneity and synchronicity of actions in a scheduling problem and its corresponding automata composition through an example.

3.2.1 Example 2

Assume there is a set of tasks $T = \{a, b, c\}$ to be done and their durations are 7, 5 and 3 time units respectively. Mutual exclusions between tasks are such that a and c are in conflict with each other and b is not in conflict with none of them. The desired behavior is such that c must be executed after finishing task a and every task must be done only once. Each hypothesis should be modeled in a separate automaton. WA model of these hypotheses are depicted in Figure 3.1. In this figure, automata $G1$ and $G2$ demonstrate the mutual exclusion between a and c. Automata $G3$, $G4$ and $G5$ correspond to TL automata of tasks

a, b and c respectedly and automaton $G6$ is the PR automata presenting precedence constraint between tasks a and task c.

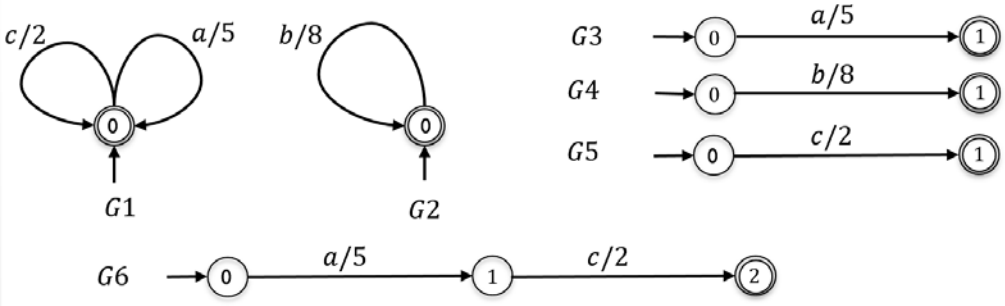

Figure 3.1. WA model of Example 2.

According to Figure 3.1, task b can be executed is parallel with task a or task c. Intuitively it can be seen that one of the optimal schedule is to firstly do task a in parallel with task b and after finishing a, to do task c in parallel and during the remaining time of b. The Gantt chart corresponding to this schedule is shown in Figure 3.2. As it can be seen in the figure, in the middle of execution of task b in time 5, tasks a is finished and c should be started. While, when executing all automata in parallel, for example when G_2 takes the transition to execute action b, it doesn't return to the state 0 until time 8 when it finishes the action. The same goes for the automaton G_4; from state 0, it takes a transition for executing action b and reaches the state 1 when b is finished. Therefore, in the global behavior of the automata, when taking transitions with label a and b in all automata, it is not possible to stop the global automata before finishing action b G_2 and G_4.

For solving this issue, intermediate states should be added in these transition in G_2 and G_4 after doing action b for 5 time units. Therefore, at time 5, all automata stop execution and the ones that have finished previous actions can start new actions.

Thence, a generic rule could be defined like this: synchronous composition of automata should be such that in each trajectory, whenever an action is finished in a local automaton, intermediate states in all other automata are defined. Therefore, the local automaton can start another action in that time instant.

Furthermore, synchronicity of transitions should be such that all transitions executing the same actions, be executed at the same time. For example, despite automaton G_5 is idle from time 0 to 5, action c in automata G_1 and G_5 and G_6 are executed synchronously from time 5 to time 7.

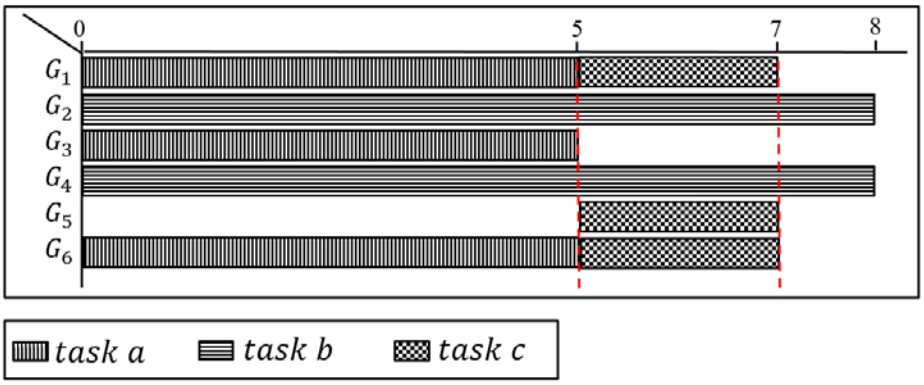

Figure 3.2. Gantt chart of automata G_1 to G_6 according to a sample schedule

3.2.2 Algorithmic steps to reach synchronous composition

The synchronous product computation is defined for *n* WA. The proposed approach operates in five steps:

(1) Define the set *X* of all potential global transitions, such that whenever an action is shared among several WA, it should be executed synchronously. The remaining combinations represent actions to be taken either in parallel or independently;
(2) Split transitions of *X* modeling simultaneous actions which do not have the same duration;
(3) Aggregate transitions enabling simultaneous actions at the earliest possible time;
(4) Repeat steps 2 and 3 until a fixed point is reached;
(5) Build a global transition relation from the preceding result.

These steps are described by Algorithm 3.1 and detailed below.

Algorithm 3.1: $expansion\ (X, Q_1, \dots, Q_n)$

$X^{(0)} = X$
$i \coloneqq 0$
$z \coloneqq 1$
$for(j \coloneqq 1\ to\ n)\{$
  $for(all\ q_j \in Q_j)\{$
    $q_j.origin \coloneqq q_j$
  $\}end\ for$
$\}end\ for$
$repeat$
  $XS \coloneqq expand\_by\_splitting(X^{(i)})$
  $X^{(i+1)} \coloneqq expand\_by\_aggregation(XS, Q_1, \dots, Q_n)$
  $i \coloneqq i + 1$
$until\ X^{(i)} = X^{(i-1)}$

$return\ X^{(i)}$

Step 1. Definition of the global transition set *X*.

For all $G_i, i = 1..n$ the tuple

$$\big((q_1, \dots, q_n), (a_1, \dots, a_n), (d_1, \dots, d_n), (q'_1, \dots, q'_n)\big)$$

belongs to X iff transition (q_i, a_i, d_i, q'_i) exists in G_i via δ_i, where q_i is the source and q'_i is the target of a transition in G_i that executes an action a_i with duration d_i. Therefore for n automata, X is defined as

$$X = \{\big((q_1, \dots, q_n), (a_1, \dots, a_n), (d_1, \dots, d_n), (q'_1, \dots, q'_n)\big) | \ (\forall i \in [1,n]: a_i \in \Sigma_i, d_i = d(a_i),\ q_i, q'_i \in Q_i) \wedge \big(\forall i \neq j \in [1,n]: if\ a_i \in (\Sigma_i \cap \Sigma_j) \rightarrow (a_i = a_j) \wedge (d_i = d_j)\big) \wedge (q_i, a_i, d_i, q'_i) \in \delta_i\} \quad (2.11)$$

In this definition, $\forall i \neq j \in [1,n]: a_i \in (\Sigma_i \cap \Sigma_j) \rightarrow (a_i = a_j) \wedge (d_i = d_j)$ denotes the fact that shared actions (having the same symbol) are executed synchronously.

Step 2. Splitting transitions of X

In the following, vectors of symbols are denoted between the symbols "⟨" and "⟩" and $(\langle q_1, \dots, q_n\rangle, \langle a_1, \dots, a_n\rangle, \langle d_1, \dots, d_n\rangle, \langle q'_1, \dots, q'_n\rangle)$ is denoted by $(\langle q\rangle, \langle a\rangle, \langle d\rangle, \langle q'\rangle)$.

All transitions of *X* are expanded into two sequential transitions exhibiting either 0 or synchronous timing. All processed transitions are memorized inside the set *S*. In order to make the subsequent aggregation easier, the components of each vector are permuted according to an ascending order of the corresponding weights. This order is established by the function $sort$. For example, if $d_i > d_j$ and $i < j$, q_i, q'_i and a_i are permuted with q_j, q'_j and a_j. A track of the permutation is stored inside the tuple $(t_1, \dots, t_n)$.

Algorithm 3.2:$expand\_by\_spliting(X)$

$S := \emptyset$

$for\ (all\ (\langle q\rangle, \langle a\rangle, \langle d\rangle, \langle q'\rangle) \in X)\{$

$y := 1$ *//it helps tracking number of times a tuple is split*

$if\big((\langle q\rangle, \langle a\rangle, \langle d\rangle, \langle q'\rangle) \notin S\big)\{$

$S := S \cup \{(\langle q\rangle, \langle a\rangle, \langle d\rangle, \langle q'\rangle)\}$

$\big((\langle q\rangle, \langle a\rangle, \langle d\rangle, \langle q'\rangle), (t_1, \dots, t_n)\big) := sort\big((\langle q\rangle, \langle a\rangle, \langle d\rangle, \langle q'\rangle)\big)$

$X := expand\_one\_tuple\big((\langle q\rangle, \langle a\rangle, \langle d\rangle, \langle q'\rangle), (t_1, \dots, t_n), X, y\big)$

$\}end\ if$

$\}end\ for$

$return\ X$

Algorithm 3.3: $expand\_one\_tuple\big((\langle q\rangle,\langle a\rangle,\langle d\rangle,\langle q'\rangle),(t_1,\dots,t_n),X,S,y\big)$

$m := nonzeromin\big((\langle q\rangle,\langle a\rangle,\langle d\rangle,\langle q'\rangle)\big)$

$eql := arequal\big((\langle q\rangle,\langle a\rangle,\langle d\rangle,\langle q'\rangle),m\big)$

//eql=1 if all non-silent actions of the tuple have the same duration as the fastest action

$if\ (eql = 1\ and\ y > 1)\{$ *//if all non-silent actions of the tuple have the same duration and the tuple is the output of the previous split*

$(\langle q\rangle,\langle a\rangle,\langle d\rangle,\langle q'\rangle) := restore\big((\langle q\rangle,\langle a\rangle,\langle d\rangle,\langle q'\rangle),(t_1,\dots t_n)\big)$

$X := X \cup \{(\langle q\rangle,\langle a\rangle,\langle d\rangle,\langle q'\rangle)\}$

$S := S \cup \{(\langle q\rangle,\langle a\rangle,\langle d\rangle,\langle q'\rangle)\}$

$return\ X$

$\}elseif\ (eql = 0)\{$ *//if the duration of non-silent actions in the tuple were not equal, the element should be split into two tuples.*

$((\langle q_1\rangle,\langle a_1\rangle,\langle d_1\rangle,\langle q_1'\rangle),(\langle q_2\rangle,\langle a_2\rangle,\langle d_2\rangle,\langle q_2'\rangle)) := split((\langle q\rangle,\langle a\rangle,\langle d\rangle,\langle q'\rangle),m)$

$(\langle q_1\rangle,\langle a_1\rangle,\langle d_1\rangle,\langle q_1'\rangle)) := restore\big((\langle q_1\rangle,\langle a_1\rangle,\langle d_1\rangle,\langle q_1'\rangle),(t_1,\dots t_n)\big)$

$X := X \cup \{(\langle q_1\rangle,\langle a_1\rangle,\langle d_1\rangle,\langle q_1'\rangle)\}$

$S := S \cup \{(\langle q_1\rangle,\langle a_1\rangle,\langle d_1\rangle,\langle q_1'\rangle)\}$

$y++$ *//it helps to find out if a tuple have been split previously*

$expand\_one\_tuple\big((\langle q_2\rangle,\langle a_2\rangle,\langle d_2\rangle,\langle q_2'\rangle),(t_1,\dots,t_n),X,y\big)$

$\}end\ if$ *//if all non-silent actions of the tuple have the same duration and the tuple is not split formerly, do nothing*

$return\ X$

Algorithm 3.4: $nonzeromin\big((\langle q_1,\dots,q_n\rangle,\langle a_1,\dots,a_n\rangle,\langle d_1,\dots,d_n\rangle,\langle q_1',\dots,q_n'\rangle)\big)$

$i := 1$

$while\ (d_i = 0\ and\ i \le n)\{$

$i++$

$m := i$

$\}end\ while$

$return\ m$

Algorithm 3.5: $arequal\big((\langle q_1,\dots,q_n\rangle,\langle a_1,\dots,a_n\rangle,\langle d_1,\dots,d_n\rangle,\langle q_1',\dots,q_n'\rangle),m\big)$

$eql := 1$

$i := m + 1$

$while\ ((i \leq n)\ and\ (eql = 1))\{$
$\quad if\ (d_i \neq d_m)\{$
$\quad\quad eql := 0$
$\quad \}end\ if$
$\quad i++$
$\}end\ while$
$return\ eql$

As explained above, Algorithm 3.3 ($expand\_one\_tuple$) splits each tuple to two or more tuples recursively for which all duration elements d_i have the same value or are equal to zero. In this algorithm, firstly through function $nonzeromin$ (Algorithm 3.4), the index of the first non-silent element of the tuple is assigned to m. Then, the function $arequal$ (Algorithm 3.5) finds out if all non-silent actions of the tuple have the same duration as the fastest action. In other words, in this function if all duration elements are equal to d_m or zero, the output would be one. Otherwise the output would be zero. If they were equal, it means that the tuple doesn't need to be expanded; and if y was equal to one, it means that it is not the result of a previous expansion and already exists in X. So the algorithm terminates. While if eql is equal to one and y was bigger than one it means that the tuple is the output of the previous recursion. So it doesn't need to be expanded anymore. Whereas, as it is not integrated into X yet, the order of its elements is firstly restored according to their original order kept in t and through $restore$ algorithm (Algorithm 3.7). Then the tuple is integrated into X. It is also included in S to prevent going through process of splitting another time.

If the duration of actions in the tuple were not equal, it should be split to two tuples. $split$ algorithm (Algorithm 3.6) yields two outputs. In the first one (i.e. $(\langle q_1\rangle, \langle a_1\rangle, \langle d_1\rangle, \langle q_1'\rangle)$), all non-silent actions have equal durations and hence it should be included in the set E. Therefore, first the order of elements is restored and then it is integrated into E.

There is a possibility that duration elements of the second input of $split$ algorithm (i.e. $(\langle q_2\rangle, \langle a_2\rangle, \langle d_2\rangle, \langle q_2'\rangle)$) may not have the same value. Thence, $expand\_one\_tuple$ is called for the second tuple.

The $split$ algorithm (Algorithm 3.6) is based on the first non-silent action (i.e. mth action). $B = (\langle qb_1, \dots, qb_n\rangle, \langle b_1, \dots, b_n\rangle, \langle db_1, \dots, db_n\rangle, \langle qb_1', \dots, qb_n'\rangle)$ and $C = (\langle qc_1, \dots, qc_n\rangle, \langle c_1, \dots, c_n\rangle, \langle dc_1, \dots, dc_n\rangle, \langle qc_1', \dots, qc_n'\rangle)$ represent the resulting tuples. In the splitting process, first states and last states don't change. It means that qb_1 to qb_n are equal to q_1 to q_n and qc_1' to qc_n' are equal to q_1' to q_n'. Transitions executing ε or other silent actions are executed through the first

tuple and don't need to be split. So their related elements are places in tuple B before mth elements. Thereby, b_1 to b_{m-1} are equal to a_1 to a_{m-1} and d_1 to d_{m-1} are equal to zero. Since these actions finish in one transition, qb'_1 to qb'_{m-1} and qc_1 to qc_{m-1} are equal to q'_1 to q'_{m-1} respectively. In addition, c_1 to c_{m-1} are ε and their related durations are equal to zero.

The rest of the elements in B have the same duration as the mth. In tuple C, those actions who are already finished, will be replaced by ε with zero duration. Thence, their corresponding target and source in the first and second tuples respectively, are the same as their original transition target. Furthermore those that are not terminated yet, continue executing in the next transition and appear in C. Hence dc_m to dc_n are equal to the remaining durations of actions, i.e. $d_j - d_m$. Since their related local transitions are split to two transitions, an intermediate state Iq_z is created between each two transitions. The information of the source of the original transition before splitting is also kept in $Iq_z.origin$ to be used mainly in $expand\_by\_aggregation$ functions.

For example, let us give the sorted tuple $(\langle 0,0,0\rangle, \langle c, a, b\rangle, \langle 3,5,7\rangle, \langle 1,1,1\rangle)$ to $split$ algorithm as input. It can be seen that since no duration elements is equal to zero, m is equal to 1. For splitting the tuple through $split$ algorithm, the first tuple will have duration elements equal to the smallest element, but with their original action elements. Furthermore, local sources of transitions will remain unchanged, while only those local transitions will reach their targets that have finished execution of their related actions. For those that have not finished execution, an intermediate state should be added. Hence, the first split tuple is $(\langle 0,0,0\rangle, \langle c, a, b\rangle, \langle 3,3,3\rangle, \langle 1, Iq_1, Iq_2\rangle)$ where Iq_1 and Iq_2 are intermediate states.

Duration elements of the second tuple are obtained by substituting 3 from the original durations. Those actions for which the duration element vanishes, will be replaced by ε which means their execution are terminated. It is obvious that the source elements of the second tuple are the same as the target elements of the first tuple. Furthermore, through this algorithm, the target elements of the tuple are the same as the target elements of the non-split tuple. Thereby, the second tuple is obtained as $(\langle 1, Iq_1, Iq_2\rangle, \langle \varepsilon, a, b\rangle, \langle 0,2,4\rangle, \langle 1,1,1\rangle)$.

Algorithm 3.6: $split((\langle q\rangle, \langle a\rangle, \langle d\rangle, \langle q'\rangle), m)$

$for\ (j := 1\ to\ m-1)\{$
  $b_j := a_j$
  $db_j := d_j$
  $qb_j := q_j$

$qb'_j \coloneqq q'_j$
$c_j \coloneqq \varepsilon$
$dc_j := 0$
$qc_j \coloneqq qc'_j \coloneqq q'_j$
$\}end\ for$
$for(j := m\ to\ n)\{$
$b_j \coloneqq a_j$
$db_j := d_m$
$qb_j \coloneqq q_j$
$if\big(d_j = d_m\big)\{$
$qb'_j \coloneqq q'_j$
$c_j := \varepsilon$
$dc_j \coloneqq 0$
$qc_j \coloneqq qc'_j \coloneqq q'_j$
$\}else\ if(d_j > d_m)$
$c_j \coloneqq a_j$
$dc_j \coloneqq d_j - d_m$
$qb'_j \coloneqq qc_j \coloneqq Iq_z$
$if\ (q_j \in Q_j)\{$
$Iq_z.origin \coloneqq q_j$
$\}else\{$
$Iq_z.origin \coloneqq q_j.origin$
$\}end\ if$
$qc'_j \coloneqq q'_j$
$z++$
$\}end\ if$
$\}end\ for$
$return((\langle qb\rangle, \langle b\rangle, \langle db\rangle, \langle qb'\rangle), (\langle qc\rangle, \langle c\rangle, \langle dc\rangle, \langle qc'\rangle))$

Algorithm 3.7: $restore\big((\langle q\rangle, \langle a\rangle, \langle d\rangle, \langle q'\rangle), (t_1, \dots t_n)\big)$

$for\ (i := 1\ to\ n)\{$
$h_{t_i} \coloneqq a_i$
$dh_{t_i} \coloneqq d_i$
$qh_{t_i} \coloneqq q_i$
$qh'_{t_i} \coloneqq q'_i$
$\}end\ for$
$return\ \big((\langle qh\rangle, \langle h\rangle, \langle dh\rangle, \langle qh'\rangle)\big)$

Step 3. Aggregating transitions

As explained above, after expanding elements of X, through Algorithm 3.8, all tuples are compared and aggregated with each other if possible. For each

two tuple $A_i = (\langle q_{i1}, \dots, q_{in}\rangle, \langle a_{i1}, \dots, a_{in}\rangle, \langle d_{i1}, \dots, d_{in}\rangle, \langle q'_{i1}, \dots, q'_{in}\rangle)$ and $A_j = (\langle q_{j1}, \dots, q_{jn}\rangle, \langle a_{j1}, \dots, a_{jn}\rangle, \langle d_{j1}, \dots, d_{jn}\rangle, \langle q'_{j1}, \dots, q'_{jn}\rangle)$, first it should be verified if for all non-ε actions in A_i, the corresponding actions in A_j are ε and vice versa. If it was true, then it should be checked if sub-transitions belong to the transitions with the same source, in other words, origin of q_{i1} to q_{in} are equal to the origin of q_{j1} to q_{jn}. If so, the two tuple can be aggregated. The aggregated tuple is composed of elements related to all non-ε actions of A_i and A_j. If in both of the tuples the action of an automaton was ε, the corresponding aggregated action would be ε. The process of comparison and aggregation is continued till X doesn't change anymore.

Algorithm 3.8: $expand\_by\_aggregatation\ (X, Q_1, \dots, Q_n)$

$X' := X$
$M := |X|$
$for\ (i := 1\ to\ M\ s.t.\ (\langle q_{i1}, \dots, q_{in}\rangle, \langle a_{i1}, \dots, a_{in}\rangle, \langle d_{i1}, \dots, d_{in}\rangle, \langle q'_{i1}, \dots, q'_{in}\rangle) \in X)\{$
  $for\ \Big(\big(j := 1\ to\ M\ st.\ (\langle q_{j1}, \dots, q_{jn}\rangle, \langle a_{j1}, \dots, a_{jn}\rangle, \langle d_{j1}, \dots, d_{jn}\rangle, \langle q'_{j1}, \dots, q'_{jn}\rangle) \in X\big)\ and\ i \neq j\Big)\{$
    $compatible := true$
    $for\ (k := 1\ to\ n)\{$
      //if non-epsilon actions are not equal
      $if\big(a_{ik} \neq \varepsilon\ and\ a_{jk} \neq \varepsilon\ and\ (a_{ik}, d_{ik}) \neq (a_{jk}, d_{jk})\big)\{$
        $compatible := false$
        $break$
      //if the sub-transitions don't belong to the transitions with the same source
      $\}else\ if\big(\big(q_{ik} \in Q_k\ and\ q'_{ik} <> q_{jk}\big)\ or\ (q_{ik} \notin Q_k\ and\ q'_{ik} <> q'_{jk})\{$
      $\}else\ if\ \big(q_{ik}.origin <> q_{jk}.origin\big)\{$

        $compatible := false$
        $break$
      $\}end\ if$
    $\}end\ for$
    $if\ (compatible = true)\{$
      $for(z := 1\ to\ n)\{$
        $if(a_{iz} = \varepsilon)\{$
          $s_z := a_{jz}$
          $ds_z := d_{jz}$
          $qs_z := q_{jz}$
          $qs'_z := q'_{jz}$
        $\}else\{$
          $s_z := a_{iz}$
          $ds_z := d_{iz}$
          $qs_z := q_{iz}$
          $qs'_z := q'_{iz}$

$\}end\ if$
$\}end\ for$
$X \coloneqq X \cup \{\langle qs_1, \dots, qs_n\rangle, \langle s_1, \dots, s_n\rangle, \langle ds_1, \dots, ds_n\rangle, \langle qs'_1 \dots qs'_n\rangle\}$
$\}end\ if$
$\}end\ for$
$\}end\ for$
$if(X' \neq X)\{$
$expand\_by\_aggregatation\ (X, Q_1, \dots, Q_n)$
$\}end\ if$

Step 4. Steps 2 and 3 will be repeated until a fixed point is reached.

Step 5. In the next section, a global transition relation will be built from the preceding result.

3.2.3 Synchronous composition of DSDA weighted automata

In the previous section, local intermediate states were created to connect split transitions. Union of the previous states in automata G_i and newly added intermediate states, build the new set of states named Q'_i.

Definition 3.1 (synchronous composition of DSDA weighted automata): The automaton resulting from the synchronous composition of $G_1, \dots, G_n$ is denoted as $G_{sc} = (Q', \Sigma, Q^{init}, Q^m, \delta)$ where:

(1) $Q' = (Q'_1 \times \dots \times Q'_n)$ in which Q'_i is the expansion of the set of states Q_i obtained by Algorithm 3.1.

(2) $\Sigma = (\Sigma_1 \times \dots \times \Sigma_n)$ where Σ_i is the set of symbols in G_i.

(3) $\forall[q = (q_1, \dots, q_n)] \in Q', q \in Q^{init}\ iff\ \forall i \in [1, n]: q_i \in Q_i^{init}$ where Q_i^{init} is the set of initial locations in local DSDA weighted automaton G_i.

(4) $\forall[q = (q_1, \dots, q_n)] \in Q', q \in Q^m\ iff\ \forall i \in [1, n]: q_i \in Q_i^m$ where Q_i^m is the set of marked locations in local DSDA weighted automata G_i.

(5) X is defined as

$$X = \{\big((q_1, \dots, q_n), (a_1, \dots, a_n), (d_1, \dots, d_n), (q'_1, \dots, q'_n)\big) | \ (\forall i \in [1, n]: a_i \in \Sigma_i, d_i = d(a_i),\ q_i, q'_i \in Q_i) \wedge \big(\ \forall i \neq j \in [1, n]: if\ a_i \in \big(\Sigma_i \cap \Sigma_j\big) \rightarrow \big(a_i = a_j\big) \wedge (d_i = d_j)\big) \wedge (q_i, a_i, d_i, q'_i) \in \delta_i\}.$$

(6) δ is the transition relation of the composed automata and defined as

$$\delta = \{((q_1, \dots, q_n), (a_1, \dots, a_n), d, (q'_1, \dots, q'_n)) | (d = max_{i\in[1,n]}\, d_i) \wedge ((q_1, \dots, q_n), (a_1, \dots, a_n), (d_1, \dots, d_n), (q'_1, \dots, q'_n)) \in X' \wedge (\forall i \neq j \in [1,n]: d_i = d_j)\}$$

where the set X' is the expansion of the set X through Algorithm 3.1.

In each transition of the composition, q'_i is either the same as q_i or is the state reached from q_i by partially or totally execution of action a_i that takes d time units. In a transition, all local sub-actions have the same durations. This rule is considered to enable maximum simultaneity of tasks and to prevent unnecessary delays in task executions.

3.2.4 A simple example of synchronous composition (Example 2-continue)

For constructing the synchronous composition of different components of the problem, firstly the set X should be made and expanded through algorithms represented in section 3.2.2. To this aim, at the beginning, the set of transition relations related to each automaton should be defined:

$\delta_1 = \{(0, a, 5,0), (0, c, 2,0)\}$

$\delta_2 = \{(0, b, 8,0)\}$

$\delta_3 = \{(0, a, 5,1)\}$

$\delta_4 = \{(0, b, 8,1)\}$

$\delta_5 = \{(0, c, 2,1)\}$

$\delta_6 = \{(0, a, 5,1), (1, c, 2,2)\}$

According to local set of transition relations, the set X can be defined:

$X^{(0)} = X = \{(\langle 0,0,0,0,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,0,0,1\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,0,1,2\rangle),$

$(\langle 0,0,1,0,1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,8,0,8,2,2\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,0,1,0,0\rangle),$

$(\langle 0,0,0,1,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,1,1,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, \langle 5,8,5,8,0,5\rangle, \langle 0,0,1,1,0,1\rangle)\}$

After building X, its elements should be expanded according to Algorithm 3.1 explained in the previous section. In this algorithm, first the information of the source of original transition relations before splitting are kept in $origin.q_j$. In fact, the origin of each non-intermediate state is equal to itself, i.e. $origin.q_j = q_j$.

In the next step, $X^{(0)}$ is expanded through $expand\_by\_splitting$ algorithm. There exist some tuples in $X^{(0)}$ in which duration of non-silent actions (or sub-actions) are not equal. These tuples should be split to tuples that have either zero or equal duration-elements. For example in the tuple $(\langle 0,0,1,0,0,1\rangle, \langle c,b,\varepsilon,b,c,c\rangle, \langle 2,8,0,8,2,2\rangle, \langle 0,0,1,1,1,2\rangle)$, non-zero duration elements are either 2 or 8 and so are not equal. Thus, it should be split to tuples with the same duration elements. To this end, first elements of a tuple are sorted respecting to ascending order of their related duration elements. The sorted tuple is $(\langle 1,0,0,1,0,0\rangle, \langle \varepsilon,c,c,c,b,b\rangle, \langle 0,2,2,2,8,8\rangle, \langle 1,0,1,2,0,1\rangle)$. When sorting, its original order is kept in vector t to enable restoring its order later. After sorting, it should be split through algorithm $expand\_one\_tuple$. In this algorithm, first it is assessed if duration-elements are equal. As explained in previous section, variable y that was initialized to 1 in algorithm $expand\_by\_splitting$, determines number of times that a tuple goes through algorithm $expand\_one\_tuple$. Therefore, in the first iteration, y is equal to 1 and duration-elements are not equal. Hence, the tuple should be split through $split$ algorithm. The output is two tuples $(\langle 1,0,0,1,0,0\rangle, \langle \varepsilon,c,c,c,b,b\rangle, \langle 0,2,2,2,2,2\rangle, \langle 1,0,1,2,Iq_1,Iq_2\rangle)$ and $(\langle 1,0,1,2,Iq_1,Iq_2\rangle, \langle \varepsilon,\varepsilon,\varepsilon,\varepsilon,b,b\rangle, \langle 0,0,0,0,6,6\rangle, \langle 1,0,1,2,0,1\rangle)$. In addition, the origins of Iq_1 and Iq_2 are equal to 0, i.e. $origin.Iq_1 = origin.Iq_2 = 0$. As a result, the order of the first tuple is restored as $(\langle 0,0,1,0,0,1\rangle, \langle c,b,\varepsilon,b,c,c\rangle, \langle 2,2,0,2,2,2\rangle, \langle 0,Iq_1,1,Iq_2,1,2\rangle)$ and it is stored in X. Then the second tuple goes through the splitting process by recalling $expand\_one\_tuple$ and considering the second tuple as input.

Therefore as the second iteration of the function, $(\langle 1,0,1,2,Iq_1,Iq_2\rangle, \langle \varepsilon,\varepsilon,\varepsilon,\varepsilon,b,b\rangle, \langle 0,0,0,0,6,6\rangle, \langle 1,0,1,2,0,1\rangle)$ goes through the process of splitting. For this purpose, firstly duration elements of the tuple are evaluated. Both non-zero durations are 6 and equal and therefore there is no need for splitting. y is equal to two, which means that the tuple is the result of a previous splitting. Therefore, according to vector t, the order of the tuple should be restored. The result is $(\langle 0,Iq_1,1,Iq_2,1,2\rangle, \langle (\varepsilon,b,\varepsilon,b,\varepsilon,\varepsilon)\rangle, \langle 0,6,0,6,0,0\rangle, \langle 0,0,1,1,1,2\rangle)$ that should be stored in X.

All elements of X should be processed by algorithm $expand\_one\_tuple$.

As the result of this process, some elements are added to the set $X^{(0)}$ that are shown in bold letters. The resulting set is as follows:

$XS = \{(\langle 0,0,0,0,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,0,0,1\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,0,1,2\rangle),$

$(\langle 0,0,1,0,1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,8,0,8,2,2\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,0,1,0,0\rangle),$

$(\langle 0,0,0,1,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,1,1,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, \langle 5,8,5,8,0,5\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{1}\rangle, \langle \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{c}\rangle, \langle \mathbf{2}, \mathbf{2}, \mathbf{0}, \mathbf{2}, \mathbf{2}, \mathbf{2}\rangle, \langle \mathbf{0}, \boldsymbol{q_1}, \mathbf{1}, \boldsymbol{q_2}, \mathbf{1}, \mathbf{2}\rangle),$

$(\langle \mathbf{0}, \boldsymbol{q_1}, \mathbf{1}, \boldsymbol{q_2}, \mathbf{1}, \mathbf{2}\rangle, \langle \boldsymbol{\varepsilon}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}\rangle, \langle \mathbf{0}, \mathbf{6}, \mathbf{0}, \mathbf{6}, \mathbf{0}, \mathbf{0}\rangle, \langle \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{2}\rangle),$

$(\langle \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}\rangle, \langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{a}\rangle, \langle \mathbf{5}, \mathbf{5}, \mathbf{5}, \mathbf{5}, \mathbf{0}, \mathbf{5}\rangle, \langle \mathbf{0}, \boldsymbol{q_3}, \mathbf{1}, \boldsymbol{q_4}, \mathbf{0}, \mathbf{1}\rangle),$

$(\langle \mathbf{0}, \boldsymbol{q_3}, \mathbf{1}, \boldsymbol{q_4}, \mathbf{0}, \mathbf{1}\rangle, \langle \boldsymbol{\varepsilon}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}\rangle, \langle \mathbf{0}, \mathbf{3}, \mathbf{0}, \mathbf{3}, \mathbf{0}, \mathbf{0}\rangle, \langle \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{0}, \mathbf{1}\rangle)\}$

After expanding $X^{(0)}$ by means of splitting and obtaining XS, XS should be expanded through $expand\_by\_aggregation$ algorithm (Algorithm 3.8). In this algorithm, all tuples are assessed pairwise to be compatible. If they were compatible, they should be integrated to one tuple. By comparing first tuple $(\langle 0,0,0,0,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,0,0,1\rangle)$ with other tuples in the set, it can be found that it is compatible with some other tuples like $(\langle 0,0,0,0,0,0\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,0,1,0,0\rangle)$. The integrated tuple is $(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, \langle 5,8,5,8,0,5\rangle, \langle 0,0,1,1,0,1\rangle)$ which already exists in the set. There are some other tuples that integrating them doesn't have any effect on the set. But during this comparison, two tuples are be found that are compatible and also their integration enlarges the set X. As it is clear, action elements of $(\langle 0,0,1,0,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,0,1,2\rangle)$ and $(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,3,0,3,0,0\rangle, \langle 0,0,1,1,0,1\rangle)$ are compatible. Because firstly, if one of the actions is not ε, its corresponding element in the other tuple is ε. Secondly, since the both *origin* state for Iq_3 and Iq_4 are 1, elements in $\langle 0,0,1,0,0,1\rangle$ and $\langle 0, Iq_3, 1, Iq_4, 0,1\rangle$ have the same origin. Integrating these two tuples yields tuple $(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,3,0,3,2,2\rangle, \langle 0,0,1,1,1,2\rangle)$ that will be integrated in X. Thereby $X^{(1)}$ is as follows:

$X^{(1)} = \{(\langle 0,0,0,0,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,0,0,1\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,0,1,2\rangle),$

$(\langle 0,0,1,0,1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,8,0,8,2,2\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,0,1,0,0\rangle),$

$(\langle 0,0,0,1,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,1,1,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, \langle 5,8,5,8,0,5\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,2,0,2,2,2\rangle, \langle 0, Iq_1, 1, Iq_2, 1,2\rangle),$

$(\langle 0, Iq_1, 1, Iq_2, 1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,6,0,6,0,0\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, \langle 5,5,5,5,0,5\rangle, \langle 0, Iq_3, 1, Iq_4, 0,1\rangle),$

$(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,3,0,3,0,0\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle \mathbf{0}, \boldsymbol{Iq_3}, \mathbf{1}, \boldsymbol{Iq_4}, \mathbf{0}, \mathbf{1}\rangle, \langle \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{c}\rangle, \langle \mathbf{2}, \mathbf{3}, \mathbf{0}, \mathbf{3}, \mathbf{2}, \mathbf{2}\rangle, \langle \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{2}\rangle)\}$

In this moment, one repetition is finished, whereas since $X^{(1)}$ is not equal to $X^{(0)}$, the process of expansion by splitting and aggregation should be done once more. Thus, first function $expand\_by\_splitting$ is called for the new set $X^{(1)}$ as input. This algorithm is such that whenever it investigates any tuple, it includes it in the set S to prevent its reinvestigation. Therefore all the processed tuples in the previous iteration were integrated in S. Comparing $X^{(1)}$ and S reveals that that only the recently added tuple $(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,3,0,3,2,2\rangle, \langle 0,0,1,1,1,2\rangle)$ is not processed. Therefore, it should be sorted to obtain ascending order of duration-elements. The sorted tuple is $(\langle 1,0,0,1, Iq_3, Iq_4\rangle, \langle \varepsilon, c, c, c, b, b\rangle, \langle 0,2,2,2,3,3\rangle, \langle 1,0,1,2,0,1\rangle)$. Since its non-zero duration-elements are not equal, it should be split through $split$ function.

The result of the $split$ function is $(\langle 1,0,0,1, Iq_3, Iq_4\rangle, \langle \varepsilon, c, c, c, b, b\rangle, \langle 0,2,2,2,2,2\rangle, \langle 1,0,1,2, Iq_5, Iq_6\rangle)$ and $(\langle 1,0,1,2, Iq_5, Iq_6\rangle, \langle \varepsilon, \varepsilon, \varepsilon, \varepsilon, b, b\rangle, \langle 0,0,0,0,1,1\rangle, \langle 1,0,1,2,0,1\rangle)$. In the next step, the order of the first tuple is restored as $(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle (2,2,0,2,2,2)\rangle, \langle 0, Iq_5, 1, Iq_6, 1,2\rangle)$. Then this tuple is integrated in the set X. In the next step, the algorithm $expand\_one\_tuple$ is executed anew with the second tuple $(\langle 1,0,1,2, Iq_5, Iq_6\rangle, \langle \varepsilon, \varepsilon, \varepsilon, \varepsilon, b, b\rangle, \langle 0,0,0,0,1,1\rangle, \langle 1,0,1,2,0,1\rangle)$ as input. Since all non-zero duration-elements are equal (i.e. $eql = 1$) and the tuple is a result

of previous splitting process (i.e. $y > 1$), firstly its order is restored and then the result is included in X. The tuple with its original order is $(\langle 0, Iq_5, 1, Iq_6, 1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,1,0,1,0,0\rangle, \langle 1,0,1,2,0,1\rangle)$ that is stored in X. Here, $expand\_by\_splitting$ algorithm ends and XS is equal to:

$XS = \{(\langle 0,0,0,0,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,0,0,1\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,0,1,2\rangle),$

$(\langle 0,0,1,0,1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,8,0,8,2,2\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,0,1,0,0\rangle),$

$(\langle 0,0,0,1,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,1,1,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, \langle 5,8,5,8,0,5\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,2,0,2,2,2\rangle, \langle 0, Iq_1, 1, Iq_2, 1,2\rangle),$

$(\langle 0, Iq_1, 1, Iq_2, 1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,6,0,6,0,0\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, \langle 5,5,5,5,0,5\rangle, \langle 0, Iq_3, 1, Iq_4, 0,1\rangle),$

$(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,3,0,3,0,0\rangle, \langle 0,0,1,1,0,1\rangle),$

$(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,3,0,3,2,2\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle \mathbf{0}, \boldsymbol{Iq_3}, \mathbf{1}, \boldsymbol{Iq_4}, \mathbf{0}, \mathbf{1}\rangle, \langle \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{c}\rangle, \langle \mathbf{2}, \mathbf{2}, \mathbf{0}, \mathbf{2}, \mathbf{2}, \mathbf{2}\rangle, \langle \mathbf{0}, \boldsymbol{Iq_5}, \mathbf{1}, \boldsymbol{Iq_6}, \mathbf{1}, \mathbf{2}\rangle),$

$(\langle \mathbf{0}, \boldsymbol{Iq_5}, \mathbf{1}, \boldsymbol{Iq_6}, \mathbf{1}, \mathbf{2}\rangle, \langle \boldsymbol{\varepsilon}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{b}, \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}\rangle, \langle \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}\rangle, \langle \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{2}\rangle)\}$

In the next step, XS is processed through $expand\_by\_aggregation$ algorithm. Whereas after processing, it is found that this algorithm can't change and expand XS. So $X^{(2)}$ is equal to XS.

In the next repetition, $X^{(2)}$ is investigated through $expand\_by\_splitting$ algorithm; whereas since all its members are previously processed, they are not processed and split anymore. Therefore $X^{(2)}$ is assigned to XS. Since XS is not changed from the previous process of $expand\_by\_aggregation$, it doesn't changed anymore by this algorithm and is again assigned to $X^{(3)}$. Thereby, the resulting X' is as follows:

$X' = X^{(3)} = \{(\langle 0,0,0,0,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,0,0,1\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,0,1,2\rangle),$

$(\langle 0,0,1,0,1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,1,2\rangle),$

$(\langle 0,0,1,0,0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,8,0,8,2,2\rangle, \langle 0,0,1,1,1,2\rangle)$,

$(\langle 0,0,0,0,0,0\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,0,1,0,0\rangle)$,

$(\langle 0,0,0,1,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, \langle 5,0,5,0,0,5\rangle, \langle 0,0,1,1,0,1\rangle)$,

$(\langle 0,0,1,1,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, \langle 2,0,0,0,2,2\rangle, \langle 0,0,1,1,1,2\rangle)$,

$(\langle 0,0,1,0,0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,8,0,8,0,0\rangle, \langle 0,0,1,1,0,1\rangle)$,

$(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, \langle 5,8,5,8,0,5\rangle, \langle 0,0,1,1,0,1\rangle)$,

$(\langle 0,0,1,0,0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,2,0,2,2,2\rangle, \langle 0, Iq_1, 1, Iq_2, 1,2\rangle)$,

$(\langle 0, Iq_1, 1, Iq_2, 1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,6,0,6,0,0\rangle, \langle 0,0,1,1,1,2\rangle)$,

$(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, \langle 5,5,5,5,0,5\rangle, \langle 0, Iq_3, 1, Iq_4, 0,1\rangle)$,

$\big((0, Iq_3, 1, Iq_4, 0,1), (\varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon), (0,3,0,3,0,0), (0,0,1,1,0,1)\big)$,

$(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,3,0,3,2,2\rangle, \langle 0,0,1,1,1,2\rangle)$,

$(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, \langle 2,2,0,2,2,2\rangle, \langle (0, Iq_5, 1, Iq_6, 1,2)\rangle)$,

$(\langle 0, Iq_5, 1, Iq_6, 1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, \langle 0,1,0,1,0,0\rangle, \langle 0,0,1,1,1,2\rangle)\}$

After expansion of X, synchronous composition could be built. According to the definition of synchronous composition in the previous section, the set of composed transition relations can be achieved as follows:

$\delta = \{(\langle 0,0,0,0,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, 5, \langle 0,0,1,0,0,1\rangle)$,

$(\langle 0,0,1,0,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, 2, \langle 0,0,1,0,1,2\rangle)$,

$(\langle 0,0,1,0,1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, 8, \langle 0,0,1,1,1,2\rangle)$,

$(\langle 0,0,0,0,0,0\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, 8, \langle 0,0,0,1,0,0\rangle)$,

$(\langle 0,0,0,1,0,0\rangle, \langle a, \varepsilon, a, \varepsilon, \varepsilon, a\rangle, 5, \langle 0,0,1,1,0,1\rangle)$,

$(\langle 0,0,1,1,0,1\rangle, \langle c, \varepsilon, \varepsilon, \varepsilon, c, c\rangle, 2, \langle 0,0,1,1,1,2\rangle)$,

$(\langle 0,0,1,0,0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, 8, \langle 0,0,1,1,0,1\rangle)$,

$(\langle 0,0,1,0,0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, 2, \langle 0, Iq_1, 1, Iq_2, 1,2\rangle)$,

$(\langle 0, Iq_1, 1, Iq_2, 1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, 6, \langle 0,0,1,1,1,2\rangle)$,

$(\langle 0,0,0,0,0,0\rangle, \langle a, b, a, b, \varepsilon, a\rangle, 5, \langle 0, Iq_3, 1, Iq_4, 0,1\rangle)$,

$(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, 3, \langle 0,0,1,1,0,1\rangle)$,

$(\langle 0, Iq_3, 1, Iq_4, 0,1\rangle, \langle c, b, \varepsilon, b, c, c\rangle, 2, \langle (0, Iq_5, 1, Iq_6, 1,2)\rangle)$,

$(\langle 0, Iq_5, 1, Iq_6, 1,2\rangle, \langle \varepsilon, b, \varepsilon, b, \varepsilon, \varepsilon\rangle, 1, \langle 0,0,1,1,1,2\rangle)\}$

According to the composed transition relation set, the synchronous composition of the automata in Example 2 (Figure 3.1) is obtained as Figure

3.3. In this figure, in the upper trajectory (T_1), tasks a, c and b are executed sequentially. It can be seen that this trajectory takes 15 time units. While in the lower trajectory (T_2), first a and b are executed simultaneously in 5 time units, then task c is executed in parallel with the remaining part of b in 2 time units. After finishing c, 1 time units remains from execution of b that is done individually.

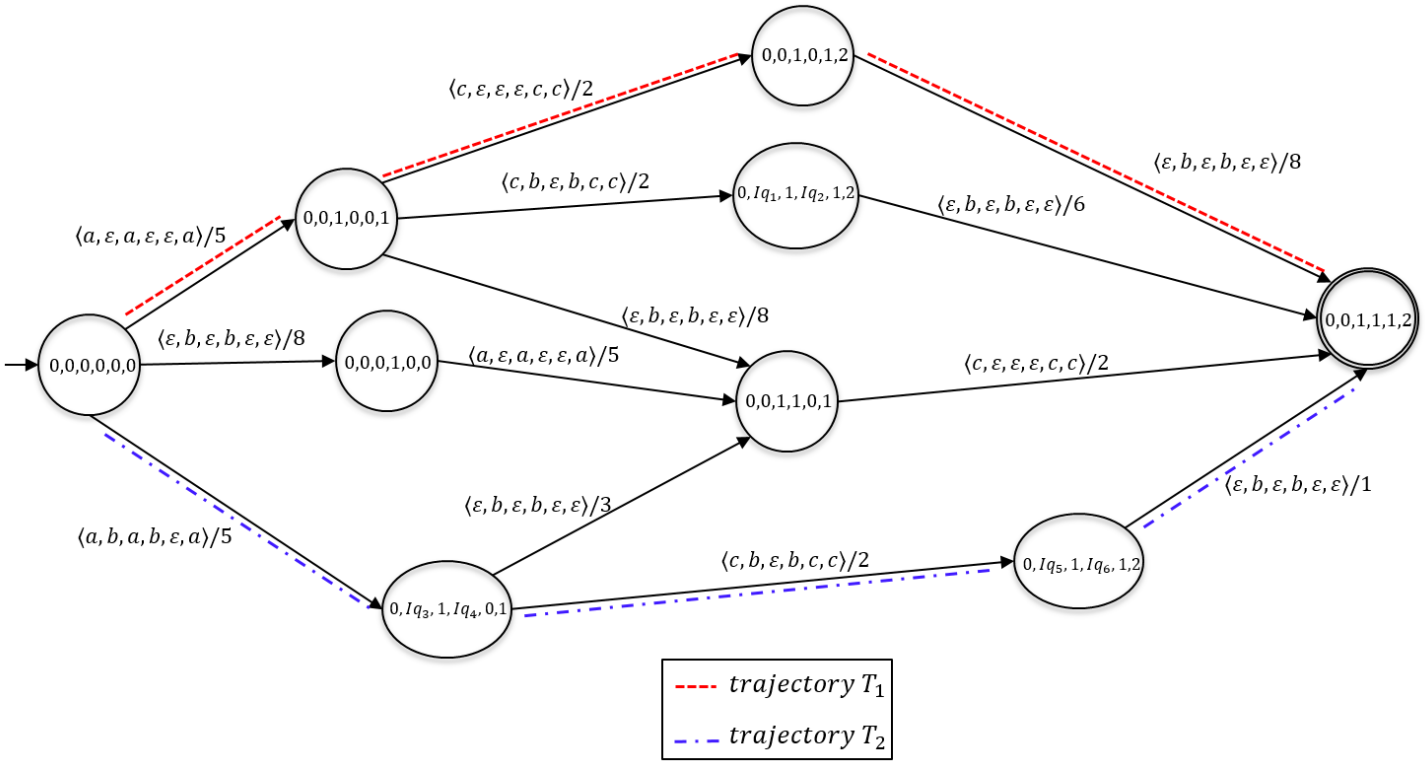

Figure 3.3. Synchronous composition of automata in Example 2

3.3 Finding the optimal schedule

In order to find the minimum makespan through synchronous composition of WA, an optimal time trajectory should be found from the initial state of the synchronous composition to its marked state.

Let $\alpha = q_0 \xrightarrow{A_1/d_1} q_1 \dots \xrightarrow{A_n/d_n} q_n$ be a finite run of a weighted automaton where A_k is a set of actions executing during kth transition and d_k denotes the duration of kth transition of the run. The duration of α, $duration(\alpha)$, is the sum $\sum_{k\in[1,n]} d_k$.

For a state q, the minimal duration for reaching q, $mindur(q)$, is the infimum of the durations of the finite trajectories which end in q (Behrmann, Fehnker, Hune, Larsen, Pettersson, Romijn, et al. 2001):

$$mindur(q) = \inf\{duration(\alpha) | \alpha: a\ run\ ending\ in\ state\ q\} \quad (3.1)$$

Gerd Behrmann et al. (2001) suggest an algorithm for determining the minimum cost for reaching a target state satisfying a property in a priced time automata. Inspired from their algorithm, Algorithm 3.9 is presented that minimizes time of reaching a marked state from the initial state in a WA and yields the fastest trajectory.

In this algorithm, all encountered states are included in two data structures $Passed$ and $Waiting$ that store explored and unexplored states, respectively. Initially, $Passed$ is empty and $Waiting$ includes the initial state. The global variable $makespan$, which is initially set to ∞, stores the lowest duration achieved so far for reaching the marked state. The optimum schedule is a trajectory from the initial state q_0 to the marked state q_m with the minimal duration that is initially empty.

In each iteration, a state q is taken from $Waiting$ and is checked if it corresponds to the marked state q_m. If it was the marked state and $mindur(q)$ was less than $makespan$, it stores $mindur(q)$ as the minimum $makespan$ and its related trajectory α as the optimal schedule $OptSchedule$. Then it adds the state q to $Passed$ and its successors to $Waiting$. In this algorithm, $q \rightarrow q'$ means q' is reachable from q.

Algorithm 3.9: *optimal_schedule*

$makespan \coloneqq \infty$

$OptSchedule \coloneqq \emptyset$

$Waiting \coloneqq q_0$

$Passed \coloneqq \emptyset$

$while\ (Waiting \neq \emptyset)\{$

$select\ q\ from\ Waiting$

$if\ (q = q_m\ and\ \exists \alpha\ s.t. mindur(q) = duration(\alpha)\ and\ mindur(q) < makespan)\{$

$makespan \coloneqq mindure(q)$

$OptSchedule \coloneqq \alpha$

$\}end\ if$

$add\ q\ to\ Passed$

$for\ all\ q's.t. q \rightarrow q' : add\ q'to\ Waiting$

$\}end\ while$

$return\ makespan, OptSchedule$

The algorithm terminates when $Waiting$ is empty and this happens when no more state remains in the state space to be explored. This means that the algorithm searches the entire state space.

3.4 Conclusion

In this chapter, multi-resource sharing scheduling problem is solved through time-optimal reachability analysis on WA. For this purpose, different WA models of the problem should be composed. Whereas the existing synchronous compositions are either not capable of showing simultaneous execution of actions in a scheduled system or the minimum makespan cannot be found through the composition. Hence, a new synchronous composition is presented that shows simultaneous execution of non-conflicting tasks and in addition, contains trajectories through which the minimum makespan can be obtained. In this composition, every task may be needed to be split to two or more sequential parts. Each part may be done in parallel with one action or a part of an action. In that way, a set of tasks can be executed in a shorter time. After building the synchronous composition of the problem, a schedule can be found by doing time-optimal reachability analysis to find the fastest trajectory from the initial to the marked state.

4

Multi-resource sharing scheduling considering uncontrollable environment

4 Multi-resource sharing scheduling considering uncontrollable environment

4.1 Introduction

In Chapters 2 and 3, it is considered that all the actions are controllable which means all the actions can be controlled by the controller. All the information related to the tasks are known in advance and are certain. The same goes for the resources who are assumed to be reliable. In this section, some of the parameters defining the scheduling system are considered to be uncontrollable.

Uncontrollability in the scheduling and industrial problems have been studied by various researchers. Problems considering uncontrollability can be solved through supervisory control theory. Furthermore, as explained in previous chapters, the advantage of using automata modeling approach for modeling problems is obtaining visual and expressive models. Therefore, the intent of this chapter is to solve the problem by means of automata theory and supervisory control. Thus, a literature review is conducted to investigate various kinds of uncontrollable actions in the industry and also different types of automata that are able to model uncontrollability concept.

4.1.1 State of the art

Girault et al. (2003) present a new scheduling heuristic called Fault-Tolerance Based Active Replication to produces distributed fault-tolerant schedules for embedded systems. The heuristic algorithm is implemented in the tool SYNDEX. This method can be applied to critical embedded systems for which software should be fault-tolerant. The aim of the scheduling problem is to optimize the critical path of the obtained schedule.

Abdeddaïm, Asarin, and Maler (2006a) use an extension of timed automata for solving the classical job-shop problem. The authors divide the transitions to controllable and uncontrollable. They propose shortest path algorithms for timed automata to find the optimal schedules. They also investigate non-lazy scheduling with uncertainty in task duration.

Behrmann et al. (2009) develop a tool named TIGA to solve games based on timed game automata with respect to reachability and safety properties. The

tool can output strategies to satisfy a desired condition or reach to the goal state, or let the user play as environment and against the controller. It should be mentioned that timed game automata consists of both controllable and uncontrollable actions.

Abdeddaïm, Asarin, and Sighireanu (2009) present a subclass of timed game automata, called Task TGA which is defined as networks of communicating tasks. In this network, the start time of tasks are deterministic, while their duration are uncertain. This paper presents an approach to solve finite-horizon reachability games on Task TGA by building strategies in the form of Simple Temporal Networks with Uncertainty. They probe that finding this kind of strategy is NP-complete. This study can be used in scheduling problems that consider task-duration uncertainty.

David, Illum, and Larsen (2009) proposed a framework to model and analyze a variety of schedulability scenarios for problems that deal with multiprocessor systems, timing uncertainties in arrival and execution times, possible dependencies of tasks and preemption of resources. The problem is modeled by timed automata.

Komenda, Lahaye, and Boimond (2009) investigate supervisory control of (max, +) automata. To this end, parallel composition of (max, +) automata is presented for which, behavior corresponds to generalized version of Hadamard product. The concept of uncontrollability in this study is such that uncontrollable events can neither be forbidden and can nor be delayed.

Komenda, Lahaye, and Boimond (2010) study behavior of synchronous composition of interval weighted automata resulting in multi-event interval weighted automata. This type of automata is defined as automata with weights in a product dioid. In fact, they are an extension of (max,+) automata since instead of exact durations, temporal constraints are assigned to transitions..

Dumitrescu et al. (2010) propose a framework for multi-criteria optimal controller synthesis to model and optimize fault-tolerant distributed systems considering task execution cost and its service quality. Moreover, to combine criteria, the authors consider three different methods: aggregation, hierarchization and translation. To model the multi-task system, a type of WA called labeled transition system is defined based on input and outputs events.

Marangé et al. (2011) propose a job-shop scheduling model by communicating automata to handle reconfiguration of a manufacturing plant due to resource failure. Following a reconfiguration request, a scheduling is

generated for a set of products that are produced by a set of machines. This schedule can be obtained by means of reachability analysis on the model.

Atto, Martinez, and Amari (2011) provide a (max, +)-based method to supervise discrete event systems subject to tight time constraints. The supervision is designed to guarantee temporal constraints without having an intense effect on the dynamic behavior of the system. The authors model the studied system as timed event graphs represented by linear (max, +) state equations. The method is applied to an example of industrial manufacturing plant subject to strict temporal constraints, the thermal treatment of rubber parts for the automotive industry. In this study, temporal constraints are set to avoid losing parts for the reason of possible failure in the transport device.

Su, Van Schuppen, and Rooda (2012) addresse a minimum-makespan supervisory synthesis job shop problem. They assume also occurrence of uncontrollable events such as malfunction of a component when the system reaches a certain state, being unable to execute a program immediately when the operating system is still retrieving all relevant execution resources for this program according to a pre-specified internal mechanism unknown to the end user, producing imperfect product, etc. The makespan of the problem is computed through theory of heap-of-pieces. A timed supervisory control map is also presented that is capable of implementing the synthesized minimum-makespan sublanguage. The author models the problem by weighted and un-weighted deterministic finite state automata.

Delaval et al. (2013) deal with discrete control of computing systems administration. The authors provide an approach based on a synchronous programming language BZR to solve the control problem. Hierarchical and parallel automata can be built through BZR language. The defined parallel automata is the same as what is defined in the previous chapters of this thesis. While in the case of hierarchy, the sub-automata define the behavior of each state in the upper-level automaton. The BZR compiler is implemented on top of the Heptagon compiler and the SIGALI DCS tool. In this language, uncontrollable events can be given as input variables. While the uncontrollable environment can be modeled through this language, it does not consider timing aspect, which is an indispensable parameter in scheduling.

The idle time between two consecutive tasks may cause significant impact on the quality of wafer in certain wafer fabrication processes needing high temperature and pressure. Therefore, in such process, a cluster tool must facilitate production of wafers through a steady schedule to provides high wafer quality. Kim, Zhou, and Lee (2014) propose a method for steady scheduling of

a single-armed cluster tool based on timed event graph and (max,+) algebra. They concern disruptive events during the fabrication process, such as wafer alignment failure. Therefore, the authors develop strategies that change the system dynamics of a cluster tool by delaying some tasks to stabilize disrupted schedule in a finite time.

Boukra, Lahaye, and Boimond (2015) propose new representations for (max,+) automata in order to describe their extremal behaviors. Consequently, the defined automata is applied to performance evaluation application. Thereby, the worst case and optimum case of behavior of automata are formulated in a form that has polynomial complexity. Furthermore, the authors defines an equation to find the start time of actions in optimal and worst cases. Thus, if the automaton corresponds to the synchronous composition of components of a scheduling problem, the minimum makespan and start time of tasks can be found through the presented formula for the performance evaluation. In this article, the presented formulae are applied to an example of scheduling problem. Whereas, the complexity of finding the schedule is not discussed. Supervisory control of automata and the notion of uncontrollability is studied in this paper. While, uncontrollability of actions is restricted to not delaying.

Fernández Anta et al. (2015) deal with an online system consisting of tasks with different execution times that arrive continuously to be execute on sets of machine which are subject to crashes and restarts. The authors model and investigate the effect of parallelism and failures on the competitiveness of this system.

Lu, Cui, and Han (2015) study a single-machine scheduling problem with resource availability constraints. Unexpected breakdowns may occur for the machine which follow the Weibull failure function. Therefore, preventive maintenances are considered in the scheduling problem. The authors propose a model in which the sequence of jobs, the preventive maintenance times and the planned completion times of jobs are proactively determined simultaneously. The objective of the scheduling problem is to optimize the robustness and stability. To this end, a genetic algorithm is proposed.

Cimatti, Micheli, and Roveri (2015) address the problem of temporal planning considering uncontrollable duration of actions. The authors develop and algorithm to generate robust plans despite possible uncontrollable durations. They conducted an experimental feasibility evaluation by implementing the proposed approach in the planning tool COLIN.

Kundakcı and Kulak (2016) propose efficient hybrid Genetic Algorithm methodologies in order to minimize makespan of a dynamic job shop scheduling problem. In this problem, uncontrollable events such as random job arrivals, machine breakdowns and changes in processing time are considered.

Alves et al. (2016) addresses a supervisory scheduling problem in manufacturing systems in order to maximize parallelism among resources. They model the problem through deterministic finite automata. In addition to predictable uncontrollable events, unpredictable uncontrollable events may occur during the execution of sequence of tasks. The main objective of this study is not to minimize the makespan, but to maximize the parallelism of working resources.

Dorndorf, Jaehn, and Pesch (2017) tackle the problem of assigning flights to airport gates. In this problem, starting and completion times of flight activities are stochastic and the goal is to minimize the number of violations of any kind of constraints like shadow restrictions. In addition, they develop an online decision support system to propose recovery actions in the case of constraint violations.

Rzevski and Skobelev (2017) state that behavior of a railway operation is prone to unpredictable events such as resources unavailability due to failures, weather conditions or human errors. Another challenge in a transportation system is changing trend of demands over time, which cause resource assignment problems. In fact changing transportation resources, such as tracks, is not always possible. Therefore, the aim of this article is to design a railway scheduler capable of allocating resources to demands in real time. Depending on uncontrollable events, the scheduler should rapidly reschedule assignment of resources.

4.1.2 Synthesis of the state of the art

Table 4.1 illustrates the classification of the presented papers that concern uncontrollability aspect. In this table, the reviewed papers are classified based on four criteria: time-optimization scheduling, multi-resource sharing, timed automata and uncontrollable parameter. In the following these criteria are explained.

- Time-optimization scheduling: The discussed problem is schedulability or makespan minimization.
- Multi-resource sharing: This criteria shows that how the multi-resource sharing aspect is considered in the scheduling problems. It

can be taken into account as constraint in the problem or can be modeled using the proposed technics.

- Time-based automata: This criteria determines whether the paper discusses an approach based on a time-based automata, and if yes, on which type of time-based automata.
- Uncontrollable parameter: Different kinds of parameters and events may be assumed uncontrollable in the problems. This criteria enumerates the uncontrollable parameters explained in the paper.

Table 4.1. Classification of literature concerning uncontrollability

Authors	Time-optimization Scheduling	Multi-Resource Sharing	Timed-base automata	Uncontrollable parameter
Girault et al. (2003)	✓			Processor failure
(Yasmina Abdeddaïm, Asarin, and Maler 2006)	✓		Timed automata	Task duration
(David et al. 2009)	✓		Timed automata	Arrival and execution time of task
(Behrmann et al. 2007b)		✓	Timed game automata	Generic event
(Komenda, Lahaye, and Boimond 2009b)		✓	(max,+) automata	Undelayable and unpreventable actions
(Y. Abdeddaïm, Asarin, and Sighireanu 2009)			Task timed automata	Task duration
(Komenda, Lahaye, and Boimond 2010)		✓	Interval (max,+) automata	Action duration
(Dumitrescu et al. 2010)	✓		Labeled transition system	Processor failure
(Marangé et al. 2011)	✓		Timed automata	Resource failure
(Atto, Martinez, and Amari 2011)	✓			failure in the transport device
(Su, Van Schuppen, and Rooda 2012)	✓		Timed-weighted automata	malfunction of a component, being unable to execute a program immediately, producing imperfect product

Authors	Time-optimization Scheduling	Multi-Resource Sharing	Timed-base automata	Uncontrollable parameter
(Delaval et al. 2013)		✓		Uncontrollable variables
(Kim, Zhou, and Lee 2014)	✓			Wafer alignment failure
(Boukra, Lahaye, and Boimond 2015)	✓	✓	(max,+) automata	Undelayable and unpreventable tasks
(Fernández Anta et al. 2015)	✓			Crashes and restart of machines
(Lu, Cui, and Han 2015)	✓			Machine breakdown
(Cimatti, Micheli, and Roveri 2015)	✓			Task duration
(Kundakcı and Kulak 2016)	✓			job arrival, machine breakdowns and processing time
(Alves et al. 2016)	✓			Generic uncontrollable event
(Dorndorf, Jaehn, and Pesch 2017)	✓			Starting and completion time of flights
(Rzevski and Skobelev 2017)	✓			Resources unavailability due to failures, weather conditions or human errors – trend of transportation demand

In Table 4.1 it can be observed that only one study considers scheduling problem taking into account multi-resource sharing and uncontrollable actions during scheduling (Boukra, Lahaye, and Boimond 2015). Whereas, the authors consider the uncontrollable action to be an undelayable and unpreventable task which is a restricted notion of uncontrollability.

The synthesis of this state of the art shows that researchers have investigated various uncontrollable parameters in the scheduling problems such as task duration, start time, completion time, undelayable and unpreventable tasks, resource failure, etc. By adapting uncontrollable actions considered in the related studies to our MRS scheduling problem, three main uncontrollable actions are considered in this chapter: start time, duration of task and failure occurrence in a scheduling problem.

In addition, there are mainly 5 types of timed automata that can show uncontrollable actions; timed automata consisting of controllable and uncontrollable transitions, timed game automata, task timed tame automata,

(max,+) automata and interval weighted automata. It is noteworthy to mention that a group of uncertainties concerned by the researchers (e.g. uncertain duration of task) are considered to be controllable. In other words, time uncertainty in some problems are managed by controllable transitions. While in the similar studies, the same notion of uncertainties are assumed to be completely uncontrollable and are handled by uncontrollable transitions. Hence, due to the interpretation of the researcher from uncertainty, the type of chosen automata for solving the problem changes. For example in the first case, they use timed automata to solve the problem, while in the latter, they use timed game automata for this purpose. In this thesis, due to the constraints causing by these types of uncontrollable actions, the problem is modeled through Timed Game Automata (TGA). The detailed reason will be discussed in section 4.5.

The remainder of this Chapter is as follows: Section 4.2 recalls basic concepts related to timed games and interval weighted automata. Section 4.3 presents the problem description. Different types of uncontrollable parameters is enumerated in Section 4.4. In Section 4.5, the scheduling problem is modeled through TGA considering uncontrollable parameters. In Section 4.6, an example is studied to illustrate how a simple scheduling problem can be formulated through the proposed model. Section 4.7 discussed the solving approach for the scheduling problem. In section 4.8, the illustrative example in section 4.6 is solved through the proposed solving approach. A summary and some remarks conclude the chapter in Section 4.8.

4.2 Background

4.2.1 Timed Game Automata

Definition 4.1 (Timed Game Automata): A TGA is a tuple $G = (L, l_0, Cl, A, E, I, V, V_0)$ where L is a finite set of locations, $l_0 \in L$ is the initial location, Cl is a finite set of clocks, A is a set of actions partitioned into controllable (A_c) and uncontrollable (A_u) action, $E \subseteq L \times B(Cl) \times A \times 2^C \times L$ is a finite set of controllable and uncontrollable transitions, $I: L \to B(Cl)$ associates to each location its invariant, V is the set of integer or Boolean variables or the output of functions that are defined over variables. They can also be updated or incremented on the edges. Furthermore, predicates can be used over these variables as guards on edges of the automaton. V_0 is the initial values of V(Behrmann et al. 2007a).

Definition of state and step are the same as timed automata. Whereas in a TGA, in addition to final states, loosing states may be defined where reaching them should be avoided. Moreover, discrete steps can be made by both controllable and uncontrollable actions.

For analysis of timed automata, its simulation graph should be explored. In a simulation graph, nodes are symbolic states; a symbolic state is defined as a pair (l, Z) where $l \in L$ and Z is a zone of $\mathbb{R}_{\geq 0}^{Cl}$. Referring to the Chapter 2, each state of an automaton consists of a pair $\langle l, cl \rangle$. $Q = L \times \mathbb{R}_{\geq 0}^{Cl}$ is defined as the set of all states in an automaton where $q_0 = (l_0, \vec{0})$. Let $X \subseteq Q$ and $a \in A$. The *a-successor* of X is defined by $Post_a(X) = \{(l', cl') | \exists (l, cl) \in X, (l, cl) \xrightarrow{a} (l', cl')\}$ and the *a-predecessor* of X is defined as $Pred_a(X) = \{(l, cl) | \exists (l', cl') \in X, (l, cl) \xrightarrow{a} (l', cl')\}$. The timed successors and predecessors of X are respectively defined by $X \nearrow = \{(l, cl + d) | (l, cl) \in X \cap [Inv(l)], (l, cl + d) \in [Inv(l)], d \in \mathbb{R}_{\geq 0}\}$ and $X \swarrow = \{(l, cl - d) | (l, cl) \in X, d \in \mathbb{R}_{\geq 0}\}$. Let $\rightarrow$ be the relation defined on symbolic states by: $(l, Z) \xrightarrow{a} (l', Z')$ if $(l, g, a, r, l') \in E$ and $Z' = ((Z \cap [g])[r]) \nearrow$. The simulation graph $SG(TA)$ of timed automaton TA is defined as the transition system $(Z(Q), S_0, \rightarrow)$, where $Z(Q)$ is the set of zones of Q, $S_0 = (\{l_0, \vec{0}\} \nearrow) \cap [Inv(l_0)]$ and $\rightarrow$ defined as above (Cassez et al. 2005).

4.2.2 Safety and reachability games

Given a network of TGA and a formula φ specifying desired conditions in the TGA, rules of a game can be specified. In fact, φ defines the set of states that should be reached/avoided in order to win/lose the game. Networks of TGA consists of both controllable and uncontrollable actions. The player (controller) can trigger controllable actions to win the game and on the contrary, the opponent (environment) can trigger the uncontrollable ones that may cause losing the game. The opponent has priority over the controller. It means that when a state has both outgoing uncontrollable and controllable actions, the opponent will perform the uncontrollable action (De Munter 2010).

Two main control objectives can be defined in a network of TGA:

Definition 4.2 (reachability control problem): Given a TGA G and a set of states $K \subseteq L \times R_{\geq 0}^{Cl}$, a reachability control problem (reachability game) is defined as finding a strategy s such that G supervised by s enforces K.

Definition 4.3 (safety control problem): A safety control problem (safety game) consists of searching a strategy for triggering controllable actions to avoid reaching a set of losing states K.

4.2.3 Winning games

A game can be winning if the last transition leading to the goal state is controllable. For example, the automaton on Figure 4.1.(a) is winning since the transition leading to the goal state is controllable, while the automaton on the Figure 4.1.(b) is losing this transition is uncontrollable. In fact, opponent may decide to stay in the initial state forever without taking the transition that leads to the goal state. Therefore, the controller will never reach the Goal state.

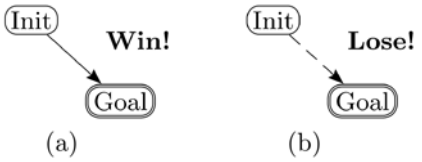

Figure 4.1. Examples of Winning (a) and Losing (b) timed game automata (Behrmann et al. 2007a)

Using an invariant might force the opponent to act. For simulating this force, an invariant could be used in the initial state and an implicit controllable edge can be added with a guard expressing the upper limit of the invariant. This implicit extra transition is called a forced transition. Hence, when the time reaches the upper limit, the automaton moves to the Goal state.

As an example, the automaton on Figure 4.2.(a) shows the original model, the automaton on Figure 4.2.(b) explicitly adds the forced transition from the initial state to the Goal state making it clear why the model is winning[3]. While, in the automaton on Figure 4.2.(c), it can be seen that the forced transition cannot be added since there is already a possible controllable behavior when the automaton hit the invariant. Thus, this model cannot reach the Goal state and is losing.

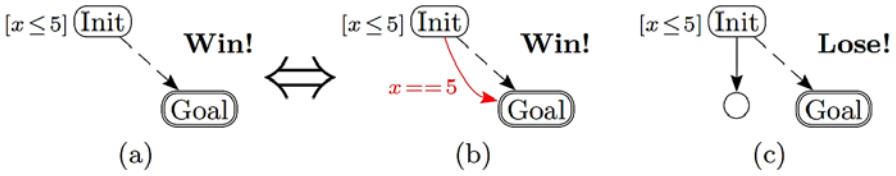

Figure 4.2. Examples of forced transition: winning model (a), equivalent model with the implicit transition made explicit (b) and a losing model (c) (Behrmann et al. 2007a)

[3] Whereas, in version 0.18 of tool TIGA described in section 4.2.5, Fig.(a) is not winning.

When facing with synchronization issue and invariants, some strange behaviors might occur. Forced transitions might become explicit in local automata and be synchronized with others. The rule to avoid the strange behaviors is to locally add forced transitions to each local automata and then to do the composition. In Figure 4.3.(a), the original model of two synchronized automata example is shown. In Figure 4.3.(b), the forced transition is made explicit and finally in Figure 4.3.(c) the full composition of the model is demonstrated (Behrmann et al. 2007a).

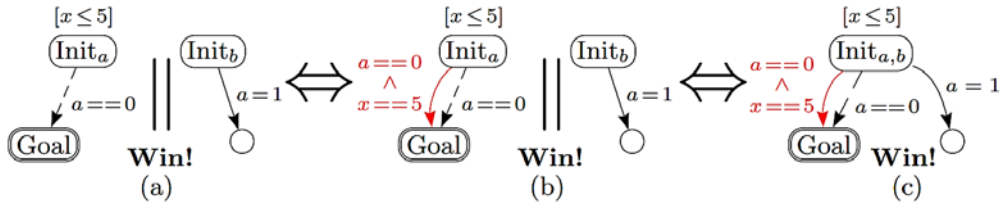

Figure 4.3. Example of synchronization with forced transitions: original model (a), forced transition made explicit (b), complete composition with explicit forced transition (Behrmann et al. 2007a).

4.2.4 Strategy

Definition 4.4 (strategy): A strategy is a function $s: L \times R_{\geq 0}^{Cl} \rightarrow A_c \cup \{\varepsilon\}$ that constantly gives information to the controller to do necessary actions during the course of the game (Behrmann et al. 2006). In each situation, the strategy could suggest the controller a list of possible controllable transitions to either take one of them and so do a controllable action or do nothing at this point in time (which means a silent action ε).

Definition 4.5 (winning strategy): A strategy is said to be a winning strategy if the controller supervised by the strategy always win the game whatever the environment acts.

Definition 4.6 (counter-strategy): Whenever no winning strategy is found, there exists a counter-strategy to either make the controller lose (reach a location that is marked as loose) or just prevent it to win (reach a location that cannot lead to the final location) (Behrmann et al. 2006).

4.2.5 Synthesis tool TIGA

UPPAAL-TIGA is an extension of UPPAAL tool and it implements the first efficient on-the-fly algorithm for solving games based on TGA with respect to reachability and safety properties. This tool implements the on-the-fly algorithm presented by Cassez et al. (2005). Being on-the-fly, the symbolic

algorithm may terminate long before having explored the entire state-space. This algorithm is detailed in section 4.2.10.

TIGA is freely available at http://people.cs.aau.dk/~adavid/tiga/

4.2.6 Winning/losing conditions

In order to win a given timed game, the winning conditions should be known. Given a set of timed automata *A*, a set of goal states (win) and/or a set of losing states (lose) which are defined by UPPAAL state formulae, a total of four different kinds of queries for winning conditions can be obtained. Clocks, locations and discrete variables can be used to specify these conditions. The game is in fact finding a controllable strategy s such that automaton A supervised by s ensures that the controller (Behrmann et al. 2007a):

- *Pure reachability*: "must reach win"

  $$control: A <> win \tag{4.1}$$

  This query searches a strategy that reaches the goal state win.

- *Strict reachability with avoidance (Until):* "must reach win and must avoid $lose$"

  $$control: A[\ not(lose)\ U\ win\ ] \tag{4.2}$$

  This query searches a strategy that will reach the goal state win and avoids losing states $lose$.

- *Weak reachability with avoidance (WeakUntil):* "should reach win and must avoid $lose$"

  $$control: A[\ not(lose)\ W\ win\ ] \tag{4.3}$$

  This query searches a controllable strategy that does not reach losing states $lose$ and maybe reaches the goal state win with the help of the opponent.

- *Pure Safety:* "must avoid $lose$"

  $$control: A[]\ not(lose) \tag{4.4}$$

  This query searches a controllable strategy that never reaches the losing state $lose$.

4.2.7 Partially Cooperative Games

A cooperative strategy can be searched in the case no controllable strategy is found. Within this strategy, the opponent will take uncontrollable actions which

helps avoiding the losing states and reaching the winning states (De Munter 2010; Behrmann et al. 2007a).

The syntax of this formula is as bellow:

- *Partial Cooperation:* "must satisfy φ with the least help from the environment"
  $E <> \ control: \varphi$ (4.5)

4.2.8 Time Optimal Strategy Synthesis

In some problems, it is needed to find not any winning strategy, but a time optimal winning strategy. The syntax of finding this kind of strategy is as follows:

- *Time optimal strict reachability with avoidance (Until):* "must reach win within less than $u - g$ time units and must avoid $lose$"
  $control\_t^*(u, g): A[\ not(lose)\ U\ win\ ]$ (4.6)

  This query searches a controllable strategy that reaches the goal states win and avoids losing states $lose$ within less that $u - g$ time units (Behrmann et al. 2007a).

- *Time optimal pure reachability:* "must reach win within less than $u - g$ time units"
  $control\_t^*(u, g): A <> \ win$ (4.7)
  This query searches a controllable strategy that reaches the goal state win within less that $u - g$ time units.

4.2.9 Example of timed game automata

In this section, an example of timed game automata is presented. Figure 4.4 shows a demo example implemented in TIGA. In this example, there are five locations containing the initial location $L0$, which is marked as a double circle, the goal location $goal$ in green color and the fail location $L4$ in red color. Controllable transition are marked by solid arrows that can be taken by the controller. Uncontrollable transitions are shown by dashed arrows and can be taken by the environment. There is a game between the environment and the controller. In fact, the controller tries to win the game and reach the $goal$ location and the environment should prevent winning by taking uncontrollable actions. The clock in this example is variable x. Clock constraints can be seen as guards on transitions. Invariant constraints on clocks are also put on the locations. In location $L0$, the invariant $x \leq 2$, in purple, doesn't allow the automaton stay more than 2 time units. When x is smaller than 1, the

environment can take the uncontrollable transition $u2$ to $L2$. While the controller can also take the controllable transition $c1$ to $L1$ from time 0 to time 1. If at time 1 the controller doesn't take the transition $c1$ to $L1$, the environment takes the transition $u1$ to $L4$ and the game will be losing. Whereas, since the controller has the choice to take action at time 1, this game is always winning.

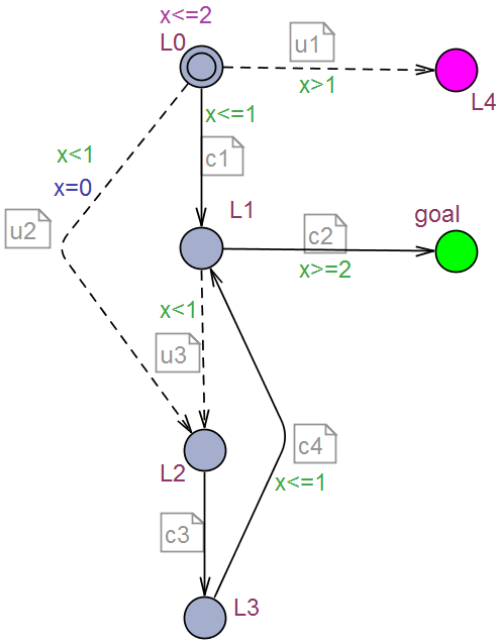

Figure 4.4. Timed game automaton example

4.2.10 On-the-fly algorithm for timed games

On-the-fly algorithm for timed games implemented in TIGA, is the extension of the untimed on-the-fly algorithm proposed by Liu and Smolka (2017).

The $SOFTR$ algorithm (Symbolic On-The-Fly Algorithm for Timed Reachability Games) is presented in Algorithm 4.1 (Cassez et al. 2005). It is based on two main lists: waiting list $Waiting$ for edges in the simulation graph to be explored, and passed list $Passed$ denoting all the symbolic states of the simulation graph that the algorithm have visited. Furthermore, all winning states that are known to be winning, are stored in the set $Win[S] \subseteq S$. The set of predecessors of S that must be added to $Waiting$ is indicated by the set $Depend[S]$. Whenever new information about $Win[S]$ is obtained, this set must be reevaluated. If a symbolic state S' is added to the set $Passed$, its related edge $e = (S, \alpha, S')$ is also added to the dependency set of S' to back-propagate possible future information about additional winning states within S' to S.

Algorithm 4.1

Initialization:

$Passed \leftarrow \{S0\}\ where\ S_0 = \{(l_0, \vec{0})\} \nearrow$

$Waiting \leftarrow \{(S_0, \alpha, S') | S' = Post_\alpha(S_0) \nearrow\}$

$Win[S_0] \leftarrow S_0 \cap (\{Goal\} \times \mathbb{R}^{Cl}_{\geq 0})$

$Depend[S_0] \leftarrow \emptyset$

Main:

$while\ ((Waiting \neq \emptyset) \wedge (s_0 \notin Win[S_0]))\{$

$e = (S, \alpha, S') \leftarrow pop(Waiting)$

$if\ (S' \notin Passed)\{$

$Passed \leftarrow Passed \cup \{S'\}$

$Depend[S'] \leftarrow \{(S, \alpha, S')\}$

$Win[S'] \leftarrow S' \cap (\{Goal\} \times \mathbb{R}^{Cl}_{\geq 0})$

$Waiting \leftarrow Waiting \cup \{(S', \alpha, S'') | S'' = Post_\alpha(S') \nearrow\}$

$if\ (Win[S'] \neq \emptyset)\{$

$Waiting \leftarrow Waiting \cup \{e\}$

$\}end\ if$

$\}else\{\ //reevaluate$[4]

$Win^* \leftarrow Pred_t(Win[S] \cup_{S \xrightarrow{c} T} Pred_c(Win[T]),$

$\cup_{S \xrightarrow{u} T} Pred_u(T \setminus Win[T])) \cap S$

$if\ (Win[S] \subsetneq Win^*)\{$

$Waiting \leftarrow Waiting \cup Depend[S]$

$Win[S] \leftarrow Win^*$

$\}end\ if$

$Depend[S'] \leftarrow Depend[S'] \cup \{e\}$

$\}end\ if$

$\}end\ while$

4.2.11 Interval weighted automata

Interval weighted automata are weighted automata with weights in a suitable interval like semiring.

Definition 4.7. A D-weighted automaton (Komenda, Lahaye, and Boimond 2010) over an alphabet A is a quadruple $G = (Q, \alpha, t, \beta)$, where Q is a finite

[4] When $T \notin Passed, Win[T] = \emptyset$

set of states, $\alpha: Q \to D$, $t: Q \times A \times Q \to D$, and $\beta: Q \to D$, are called input, transition, and output delays, respectively.

To a given state $q \in Q$, a discrete input $a \in A$ and a new state $q' \in Q$, the transition function t assigns an output value such that one of the following condition holds:

- $t(q, a, q') \in D$ corresponds to the discrete step from q to q' for execution of action a.
- $t(q, a, q') = \varepsilon$ if there is no transition form q to q' labeled by a.

Referring to (Komenda, Lahaye, and Boimond 2010), only the best and the worst makespans can be computed using composition of interval weighted automata. Comparing to a timed game, it is equal to makespans where the environment does its worst or best play respectively. While as mentioned earlier, a strategy synthesis on a timed game automaton, yields a strategy through which, several makespans and schedules can be obtained depending on the choice of the environment in taking uncontrollable actions.

4.3 Problem description

The problem treated in Chapter 2 and 3 was completely deterministic. All the information concerning the tasks to be executed was known in advance, including their identity, inter-dependence and duration. Furthermore, the starting time of tasks were deterministic. The same goes for the resources who are assumed to be reliable. Real life is not like that. Certain uncontrollable situations may happen that affect previous expectations for the scheduling. In this thesis, three kinds of uncontrollability are investigated.

For studying these situations, most assumptions for the scheduling problem are the same as Chapter 2, whereas certain uncontrollability may occur affecting the schedule. In such cases, despite the act of the environment, a schedule should be obtained. Hence, problem discerption will be changed as follows:

(1) Duration of tasks may be restricted to be bounded within an interval.
(2) Start time of tasks are subject to bounded uncertainty.
(3) Task preemption is not allowed, whereas due to a failure a task may be canceled.
(4) There may be conflicts for performing tasks at the same time, but when there is no conflict between them, they should be performed simultaneously.

(5) There may exist precedence constraints between tasks.
(6) All tasks are ready to be executed at time zero.
(7) Resources are pre-assigned to tasks.
(8) Resources are reusable (they are not raw materials and by performing maintenances, they can be used in every cycle).
(9) Each resource can be used to execute only one task at a time, but a task may use more than one resource simultaneously.
(10) If resources are not broken down, they are available at time zero.
(11) Resources are not reliable and may fail in two conditions: 1. During performing a task 2. During idle time of resources

4.4 Different types of uncontrollable parameters

In this section, three kinds of uncontrollable situations that are considered in this thesis are discussed.

- *Resource failure while doing a task:* In certain cases, resources are not reliable and for that reason, during execution of tasks, unpredictable failures may occur. Therefore, tasks relating to that resource cannot be executed until its reparation.
- *Resource failure in its idle state:* Not all the failures happen when performing tasks. Sometimes failures may occur when a resource is idle and therefore it won't be able to start execution of any task until it is repaired.
- *Bounded uncertainty in duration of tasks:* In real world, human intervention, incomplete information or uncertain environment may cause uncertainty in duration of tasks.
- *Bounded uncertainty in start time of tasks:* In certain applications, uncertain task performing conditions might cause a delay in start time of tasks from the moment of their release time. By definition, release time is the earliest time when a task can start execution.
  This type of uncertain tasks are divided to two sets; a set of tasks that require resource assignments from their released time and another set to whom resources should be assigned from the start time of their execution.

Inclusion of these types of uncontrollability in the represented model makes it so comprehensive and general to be applied in various industries. Therefore, not all of these uncontrollable parameters may be applicable in a specific

industry, but depending on the industry, one of the cases or a combination of them may be needed.

4.5 Modeling the scheduling problem through TGA considering uncontrollable parameters

As mentioned earlier, in this chapter, uncontrollable situations in the scheduling problem are investigated; it means that the environment affects the new scheduling condition. Therefore, the scheduling problem should be solved with a new point of view.

In this situation, some actions are done by the environment and therefore are uncontrollable and the others are done by the controller. For example, let's assume the case when the start time of the tasks are controllable and there is a possibility of resource failure it their idle state. In this case, tasks can be launched by the controller through taking a controllable transition, while at the same time, the environment may take an uncontrollable transition to cause a failure. Hence, as explained in section 4.2, instead of finding a simple trajectory that reaches the final state, the controller finds a strategy that guarantees reaching the final state whatever the opponent (the environment) is doing.

In Chapter 2 it was argued that through WA, simpler and more abstract models can be obtained for the MRS scheduling problem considering controllable actions rather than timed automata. Whereas, modeling some uncontrollable situations like failure during execution of tasks complicated. Thence, it is not possible to model some uncontrollability by this kind of automata. On the other hand, even if some of them like uncertain duration and start time of tasks are modeled through WA, the model of synchronous composition of the problem considering uncontrollability become much more complicated than before. Since instead of fixed durations and instances, intervals should be engaged in the composition.

For example, for modeling failure while execution of a task, it is necessary to split the task transition and add a failure state between the source and target state of the transition. On the other hand, the exact moment of the failure is not known in advance. Hence, exact duration of split transitions cannot be labeled on the transitions. Therefore, it is not possible to model it with WA. Scheduling problems consisting of tasks with uncertain duration or start time can be modeled by interval weighted automata described in section 4.2 (Komenda, Lahaye, and Boimond 2010). Whereas, there exists no synchronous composition for interval weighted automata through which the minimal

makespan can be obtained and also can present complete behavior of local components. Since such comprehensive composition becomes so complex to be formulated. Therefore, without having the synchronous composition, necessary analyses cannot be done and the schedule cannot be found.

Among uncontrollable cases explained in section 4.4, the only uncontrollable parameter that can be modeled by WA is the failure that occurs while resources are idle. In fact, its WA model would be similar to the model of a task.

With regard to the matters enumerated, a weight-based model is not suitable for stating a MRS scheduling problem considering uncontrollable issues. As mentioned in section 4.2, the clock-based automata for treating uncontrollability is Time Game Automata. Thus, for solving a MRS scheduling problem containing uncontrollable parameters, it can be modeled through Timed Game Automata.

The MRS scheduling problem is firstly modeled by TGA considering possibility of uncontrollable situations. Then through a timed game analysis, the strategy for obtaining optimal schedule and makespan is found. In the sequel, the models of ME automaton, TL automaton and PR automata considering uncontrollability are explained.

4.5.1. ME automaton by considering uncontrollability

As mentioned in section 2.4.2., a ME automaton consists of a set of tasks that are in conflict with each other. To do task t, an automaton waits in the initial location l_0. When receiving a launching signal t_s from the TL automaton of task t, it takes a transition from the initial location to the task location t and resets the clock cl. Afterwards, after elapsing $d(t)$ time units, it takes the transition from location t to the initial location. In fact, $d(t)$ is the duration of task t. In the initial location, the automaton waits for another launching signal to execute a new task (Figure 4.5).

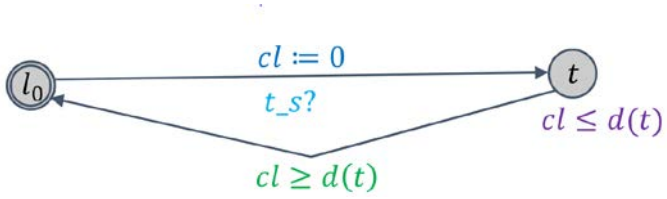

Figure 4.5. Modeling pattern of a task in a ME automaton

Whereas, as explained in section 4.4, some uncontrollable issues should be considered in the model of a task in ME automaton that are detailed in the sequel.

a. Unpredictable resource failure

In each ME automaton, due to using common set of resource, tasks can be in conflict. Therefore, if one of these resources breaks down, tasks related to that ME automaton cannot be executed until the resource is repaired. Reparation duration depends on the resource. In automaton j, this duration is denoted by $d(rep_j)$ and is equal to the maximum time needed for reparation of resources related to this automaton.

Variable b is dedicated to the resource failures, which denotes number of failures in a scheduled cycle. For avoiding schedulability problems, this number is limited. In fact, if no limit is put for this number, there would be a possibility that a scheduled cycle would never end. In this thesis, it is assumed that at most one failure may occur in a cycle. Therefore when b is reached one, no more failure can occur in a ME automaton.

Two kind of resource failures should be modeled in a ME automaton:

(1) *Failure while a resource is idle:* During idle time of resources, a failure may occur. A failure is an uncontrollable action, but it is assume to be limited to occur at most one time. Hence, in the model of failure (Figure 4.6), if b is equal to zero, an uncontrollable transition can be taken by the environment, the clock is reset and the automaton reaches fail location. Then after elapsing $d(rep_j)$ time units, the automaton takes a transition to the initial location and updates variable b to one.

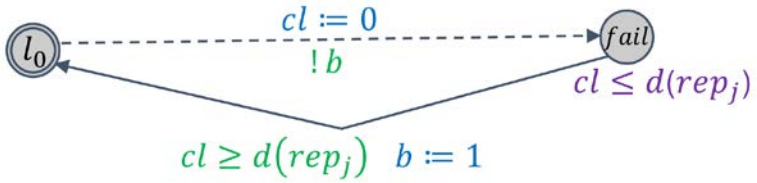

Figure 4.6. Modeling pattern of a resource failure in its idle time in a ME automaton

(2) Failure while doing a task: In automaton j, during execution of task t, a failure may happen. The reason of the failure could be a breakdown in the resources related to the mutual exclusion set j or another mutual exclusion set that share task t.

Execution of task t happens in the location t. Therefore, uncontrollable transitions representing failures should be taken from this location. For modeling a breakdown in the resources related to the automaton j, an uncontrollable transition is added from location t.

If it is the first time that a failure happens in this set of resources, i.e. $b == 1$[5], and if task i is not finished yet, i.e. $cl < d(t)$, this transition can be taken by environment. By taking this transition, clock cl is reset and a signal $stop\_s$ is sent to the other automata that share task t to stop the task. Thereby the automaton reaches location stp where it waits for repairing the failed resource. After the repairing period, i.e. $d(rep_j)$, the resource is ready and all the tasks belonging to this task-conflict set, i.e. TM_j, can start execution. Therefore, the automaton takes a transition to the initial location, updates variable b to one to prevent more failures and sends signal $restart\_s$ to other automata that share the same task to enable restarting t. These automata can be another ME automata, a PR automata or even the TL automaton related to t.

At the same time when a task is interrupted due to a failure in a ME automaton, it should also stop execution in other ME automata that share the same task. Hence, as depicted in Figure 4.7, a transition from location t to the initial location is added to the model of a task. Whenever the automaton is in task location t and receives signal $stop\_s$ from another ME automaton, it stops execution and goes to the initial location.

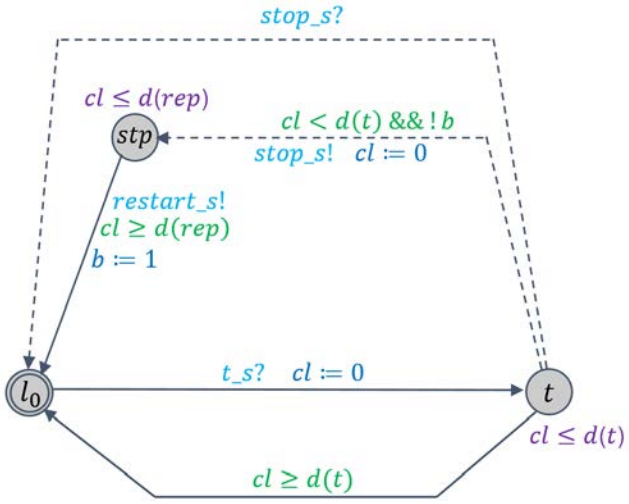

Figure 4.7. Modeling pattern of a task in a ME automaton subject to resource failure

Thereby, with possibility of abovementioned types of failures, the model of a ME automaton will be as Figure 4.8. It can be seen in the figure that this automaton consists of several tasks such as t_i, t_k and t_z.

[5] This number can be changed depending on the decision maker's opinion

For better understanding the model of ME automaton considering resource failure, refer to Section 4.6. This section explains an example of MRS scheduling model consisting of two ME automata with a shared task.

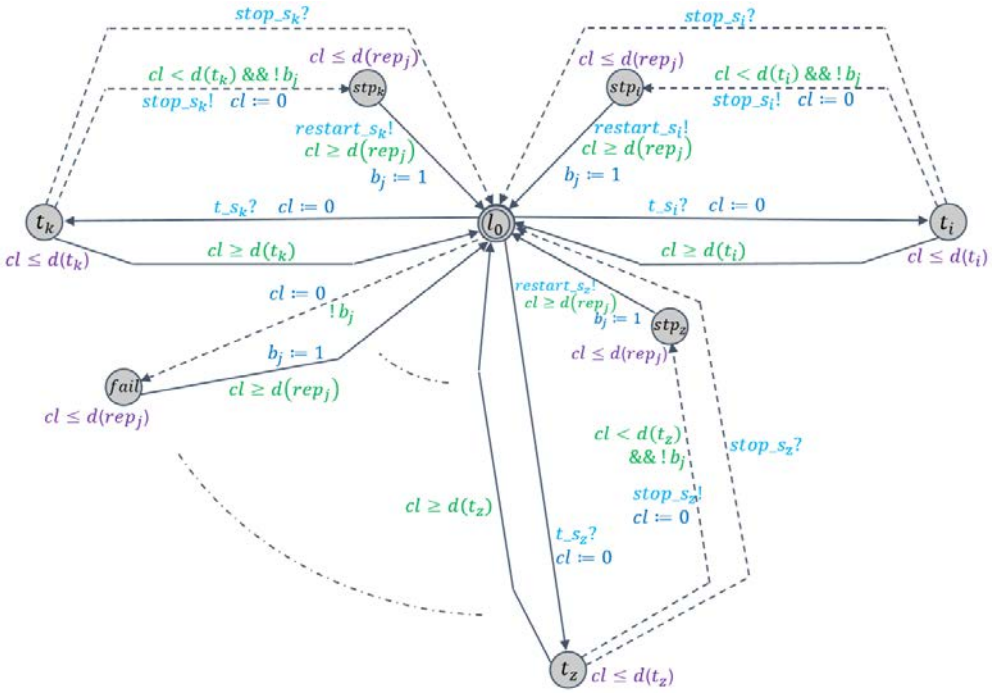

Figure 4.8. Modeling pattern of a ME automaton subject to resource failure

b. Bounded uncertainty in duration of tasks

In some applications, the exact duration of the tasks may not be given, but rather durations are restricted to be bounded within an interval of the form $[d_1(t), d_2(t)]$. Hence, when the ME automata is in the task location t, from the moment the clock reaches $d_1(t)$ till $d_2(t)$, the automata may take a transition to the initial location. Therefore an uncontrollable transition is added to enable this uncertain movement from task location to initial location when $d_1(t) \leq cl < d_2(t)$. But since due to the hypothesis it is certain that the duration of the task is no longer than $d_2(t)$, a controllable transition is added from the task location to the initial location to be taken if the automaton still waits in task location at time $d_2(t)$ (Figure 4.9).

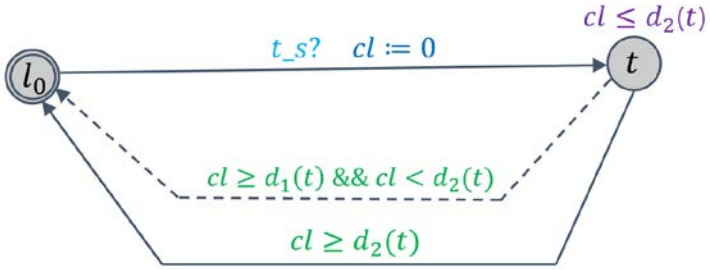

Figure 4.9. Modeling pattern of a task with uncertain duration in a ME automaton

If all the tasks in an automaton have uncertain duration, the model of the ME automaton will be as Figure 4.10. This automaton consists of several tasks such as t_i, t_k and t_z.

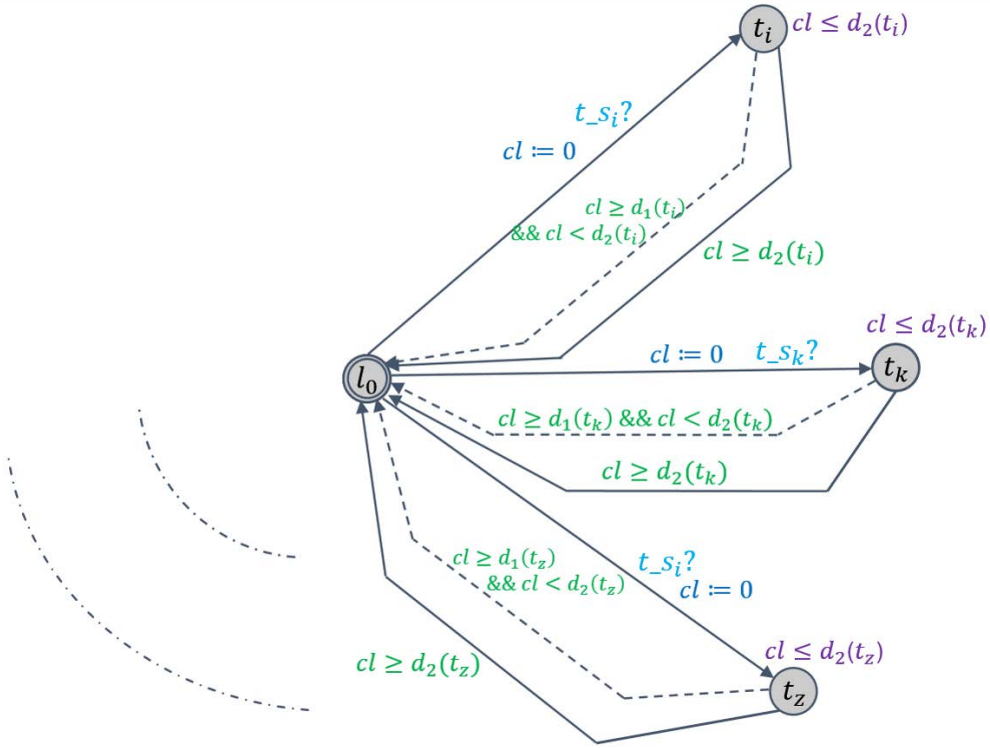

Figure 4.10. Modeling pattern of a ME automaton consisting of tasks with uncertain durations

c. Bounded uncertainty in start time of tasks

Start time of a task may be uncertain. Whereas, as mentioned previously, this uncertainty can be treated with two points of view. The first one is to keep all the necessary resources available from the release time of the task until its start time. The second point of view is to let the environment start the task at a time window whenever the resources are idle. ME automaton model for each of these cases are different:

(1) *Environment starts the task in a time window whenever related resources are idle:* This kind of uncontrollability does not have any impact on the model of the task in ME automaton. Thereby, the model of the task is the ME automaton becomes as Figure 4.5. For considering this type of uncertainty, only the model of TL automaton changes.

(2) *Keeping resources available from the lower bound of the decision time window:* Resources should be reserved from the initial moment of decision time window. Hence, from this moment, the automaton should move from the initial location to another location to prevent starting the other tasks. Although, since in this moment the task is not started yet, the automaton cannot move to the task location. For that reason, in this instant, by receiving the signal $decide\_s$ from the related TL automaton, the ME automaton moves to a new location $decide$. When the environment decides to start the task, it sends signal t_s from the TL automaton to the ME automaton. Thereby, clock is reset and the ME automaton reaches the task location t and starts the task. After waiting $d(t)$ time units in this location and finishing the task, the automaton takes a controllable transition to the initial location and waits for starting a new task (Figure 4.11).

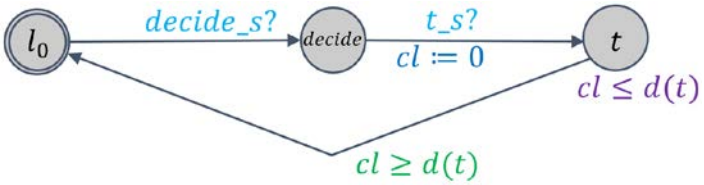

Figure 4.11. Modeling pattern of a task with uncertain start time in a ME automaton

4.5.2. Task launcher automaton by considering uncontrollability

In section 2.4.2 it is explained that for launching every task, a TL automaton is needed and hence number of TL automata are equal to the number of tasks. In this automaton, whenever all the ME automata sharing the task are in their initial location, the output of function $enabled(t)$ becomes true and the task can be launched. Therefore, the automaton sends a signal t_s to the other automata sharing the task t and reaches the location f (Figure 4.12). Thereby, t starts execution in these automata.

It should be recalled that the guard function $enabled(t): \Sigma \rightarrow Bool$, which is associated to the transition of TL automaton, is true if and only if the following condition holds:

$$\forall i \in [1, M]: t \in TM_i \rightarrow \forall t' \in TM_i \backslash \{t\}: \neg pending(t', l_i) \quad (4.8)$$

where $pending: TM_i \times L_{mei} \rightarrow Bool$ is defined as

$$pending(t, l_i) = \begin{cases} false, & l_i = l_0 \\ true, & o.w \end{cases} \quad (4.9)$$

where M is the number of ME automata, t is the name of the task, TM_i is the set of tasks engaged in ith ME automaton. In the ith ME automaton, L_{mei} is the set of locations, l_i is its current location and l_0 is the initial location. In the equation 4.9, pending function verifies if a ME automaton is in a task location where a task is being executed.

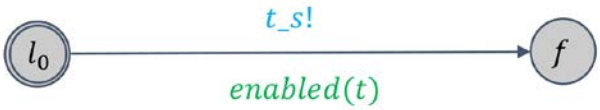

Figure 4.12. Modeling pattern of a task launcher automaton

a. Unpredictable resource failure

(1) *Failure while a resource is idle:* This kind of failure does not interrupts tasks since a TL automaton is only related to tasks and not resources. Therefore, there is no need to change the model of TL automaton for considering failure while a resource is idle.

(2) *Failure while doing a task:* Whenever a failure occurs during execution of a task, the task should be repeated after repairing the resource. Therefore, after this period, TL should return to its initial location to be able to launch the task another time.

To add this feature to the model of TL automaton, one additional location stp is created. When a task is during execution, the automaton is in location f. Therefore, at the instant of resource failure, it receives the signal $stop\_s$ from the ME automaton in which the resource failure was caused and reaches location stp. In this location, the automaton waits until reparation of the related resource. After this period, the automaton receives another signal $restart\_s$ from the ME automaton to which the resource failure was related. Thereby TL automaton goes to the initial location (Figure 4.13).

For better understanding the link between a resource failure in the model of ME automaton and TL automaton, refer to Section 4.6. This section explains the complete model for an example of MRS scheduling problem considering resource failure.

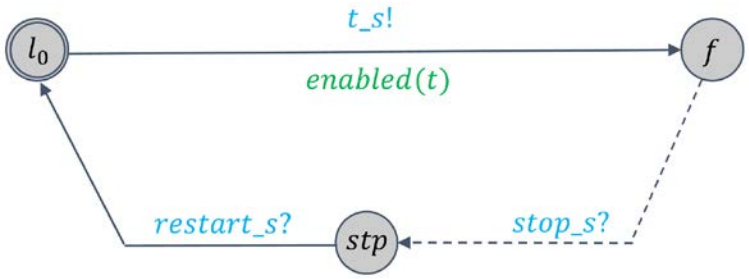

Figure 4.13. Modeling pattern of a launcher automaton for a task subject to failure

b. Bounded uncertainty in duration of tasks

A TL only launches a task and does not concern duration of tasks. Hence, for considering uncertain duration of tasks, there is no need to change the model of TL automaton.

c. Bounded uncertainty in start time of tasks

(1) *Environment starts the task in a time window whenever related resources are idle:* start time of a task may be uncontrollable such that from the instant of releasing a task, its start time vary in an interval of the form $[0, lim]$ where lim represents the maximum delay for starting the task. For modeling this feature, two issues should be considered; the first one is that as before, in addition to the communication signal t_s, the output of $enabled$ function for the task t should be verified. The releasing time of the task is let to be time zero in TL automaton. Hence, the second issue is that from time zero to the moment prior to the time bound lim launching the task t is uncontrollable. If the task is not launched until this time, the automaton should be forced to launch the task at this instant. Furthermore, its launching cannot be delayed more than this threshold.

Therefore, two transitions lead the automaton from the initial location l_0 to the location f. One transition is an uncontrollable transition that can be taken by the environment if $enabled(t)$ is true and the value of the local clock cl is less than lim. Another transition is a controllable transition that can taken by controller if $enable(t)$ is true and the value of cl is equal to lim. When taking one of these transitions, a communication signal t_s will be sent to the other automata that share the task t (Figure 4.14).

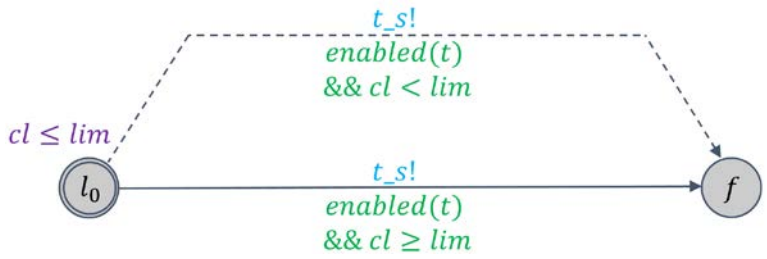

Figure 4.14. Modeling pattern of a task launcher automaton for a task with non-deterministic start time when the resources are occupied from the instant of execution

(2) *Keeping resources available from the lower bound of the decision time window:* In this case, at the instant when all necessary resources for execution of task t are available, i.e. the output of $enable(t)$ is true, the TL automaton of task t reserves resources. To this end, it takes a transition to $decide$ location and sends a signal $decide\_s$ to the ME automata sharing task t. Therefore, the related ME automata reach to their $decide$ location and this issue prevents other tasks in the related ME automata to occupy the necessary resources and to be executed. Furthermore, by taking this transition, the local clock cl is reset. In location $decide$, the automaton waits for the environment to take an uncontrollable transition to the location f until the instant that the clock value reaches lim. If the environment did not took the transition, the supervisor takes a controllable transition to f at time lim. In the both cases, a communication signal t_s is sent to the other automata that share t to start the task.

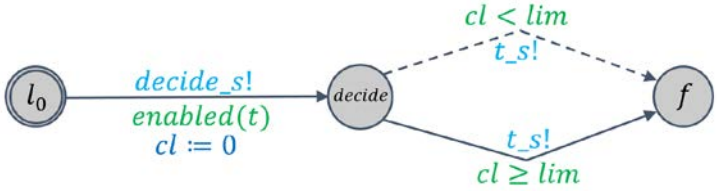

Figure 4.15. Modeling pattern of a task launcher automaton for a task with non-deterministic start time when the resources are occupied from the release time of task

4.5.3. Precedence automata by considering uncontrollability

Uncertainties of duration or start time of tasks does not change the model of PR automata. Because in both types of PR automata, only the starting instant of the task is taken into account. Thus, it is not important if a task starts right after the prior task or it starts some moments later. The same goes for the duration of the task, i.e. the finishing instant of the task is not important. Precedence constraints solely set the *minimum* time distances between the starting times of tasks.

Considering a failure in idle time of tasks also does not need to change the model of PR automata. Since this kind of failure, only cause delay between previsioned start time of tasks and extra delays are not concerned in PR automata models.

The following is the changes made in two types of PR automata to model precedence constraint among tasks that are subject to *failure while being executed*:

(1) *Triggering precedence automaton:* It is intended to put constraints on tasks that are fulfilled successfully. Since when a task fails, it will not be necessarily executed right after reparation of the broken down resource. It may be executed afterwards. Thus, the failed task should be executed anew in the PR automata.
To model this feature in triggering PR automata, it is enough to add uncontrollable transitions that lead the automaton from the locations that tasks are launched to their previous locations where tasks are not launched yet. For example when a TL launches task t_j, the PR automata in Figure 4.16 takes a transition from l_1 to l_2. If t_j fails, the automaton should execute it again. To this aim, a new uncontrollable transition could be added from l_2 to l_1. Thereby, when a failure happens, a signal $stop\_s_j$ is received from the related ME automaton and the PR automata moves to the previous location. This makes it possible to execute t_j for another time. The same goes for all the tasks in the automaton. Thus, in order to consider failure during execution of tasks, the model of triggering PR automata changes to Figure 4.17.

Figure 4.16. Modeling pattern of a triggering PR automata with reliable resources

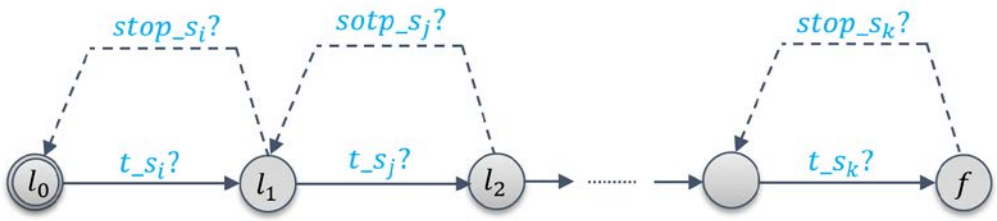

Figure 4.17. Modeling pattern of a triggering PR automata with tasks subject to failure

Delay precedence automaton: Figure 4.18 shows the modeling pattern of a delay PR automata with reliable resources that was explained in Chapter 2. In this figure, it can be seen that apart from the final location f, there exist two types of locations; those from

which a task is launched, i.e. launching locations, and the locations where the automaton waits for a specific period $del(t)$, i.e. delay locations. The process of modeling task failure in this automaton is similar to the triggering PR automata. The only difference is that in this model, there should exist transitions to lead the automata to the launching locations not only from each launching location to their previous launching location, but also from delay locations to their previous launching locations.

Figure 4.18. Modeling pattern of a delay PRautomata with reliable resources

Figure 4.19 represents modeling pattern of a delay PR automata with tasks subject to failure. After launching task t_i, the automaton reaches the delay location l_1. During execution of the task, a resource failure may happen. There is a possibility that this failure occurs during the waiting period in l_1. Whereas, if $del(t_i)$ was smaller than duration of the task $d(t_i)$, this failure may happen after time $del(t_i)$ and when the automaton has moved to the location l_2. Thus, two uncontrollable transitions are added to the automaton. Thereby, whenever a failure signal $stop\_s_i$ is received from a ME automaton, if the PR automata was at location l_1 or l_2, it will go back to the previous launching location, i.e. initial location. Related changes for the other tasks are the same as t_i.

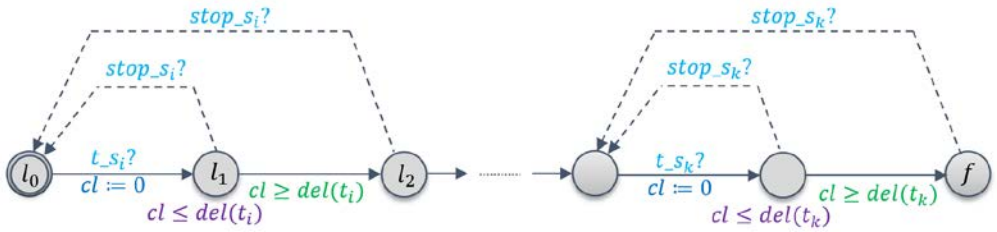

Figure 4.19. Modeling pattern of a delay PR automata with tasks subject to failure

4.6 Example 3

Assume there is a set of tasks $T = \{a, b, c\}$ to be done. Same as Example 1, a set of resources $R = \{R1, \dots, R5\}$ is assigned to tasks with resource association details shown in Table 4.3. Hence, tasks a and b are in conflict with each other, while a and c are not. Therefore, a and c can be performed simultaneously.

Duration of a is restricted to be bounded in interval [7,11] and duration of b and c are 5 and 3 time units respectively. On the other hand, start time of tasks b and c are uncertain, and vary in interval [0,2] from their release time. Despite c needs to assign resources from the first instant of its release time, b requires assignment of its necessary resources from the instant of its start time. These information are shown in Table 4.2. Timing information of tasks in Example 3

Table 4.2. Timing information of tasks in Example 3

task	Time distance between release and start time	Type of engaging resources	duration
a	0	Engage resources from start time	[7,11]
b	[0,2]	Engage resources from start time	5
c	[0,2]	Engage resources from release time	3

Table 4.3. Resource assignment details of Example 3

task	resource				
	R1	R2	R3	R4	R5
a	✓				✓
b	✓	✓	✓		
c		✓	✓	✓	

In addition to the uncertainty in duration and start time of the tasks, resources are not reliable and may fail in their idle time or even during execution of tasks. Maximum repairing time of resources are as Table 4.4. Thus, the maximum duration for repairing resources necessary to execute the set of tasks $\{a, b\}$ and $\{b, c\}$ are 4 and 5 time units respectively.

The precedence constraint among tasks is such that task a should be started after finishing task b.

Table 4.4. Repairing duration of resources

R1	R2	R3	R4	R5
4	3	4	5	4

This example is similar to Example 1. Tasks a and b compose the ME automaton G_1 and tasks b and c compose the ME automaton G_2. Also TL automata for tasks a, b and c are G_3, G_4 and G_5 respectively. Automaton G_6 represents the PR automata.

Related models for this example are depicted in the figures bellow.

Figure 4.20 represents ME automaton G_1. In this figure, $a\_s?$ denotes requesting signal for starting task a. After receiving this signal from the TL automaton G_4, the automaton resets clock cl and reaches task location a. In this location, it executes task a in at least 7 time units. From this instant until time 11, the environment may take an uncontrollable transition to the initial location to terminate the task. At time 11, if the environment hasn't done any action, the controller takes a controllable transition to the initial location l_0. Although, before finishing task and moving to the initial location, a failure may happen. It is assumed that failures may happen in each set of resources only once. This failure can be a breakdown in the set of resources used by the tasks in G_1 or G_2. In the first case, if it is the first time that a failure occurs in G_1, i.e. b_1 is equal to zero and $!\,b_1$ is true, the environment takes a transition to the location stp_a and stops task a. By taking this transition, the clock is reset and sends a signal (broadcasting channel) $stop\_s_a$ to other automata that share a. TL automaton G_3 and PR automata G_6 share a. By receiving $stop\_s_a$, automaton G_3 moves from the location f to the location stp. After 4 time units, automaton G_1 finishes reparation of the failed resource and the automaton reaches the initial location l_0. By taking this controllable transition, variable b_1 is updated to one, and send signal $restart\_s_a$ to the TL automaton of task a to enable restarting it. By receiving this signal by automaton G_3, it returns to l_0 to be able to launch the task another time.

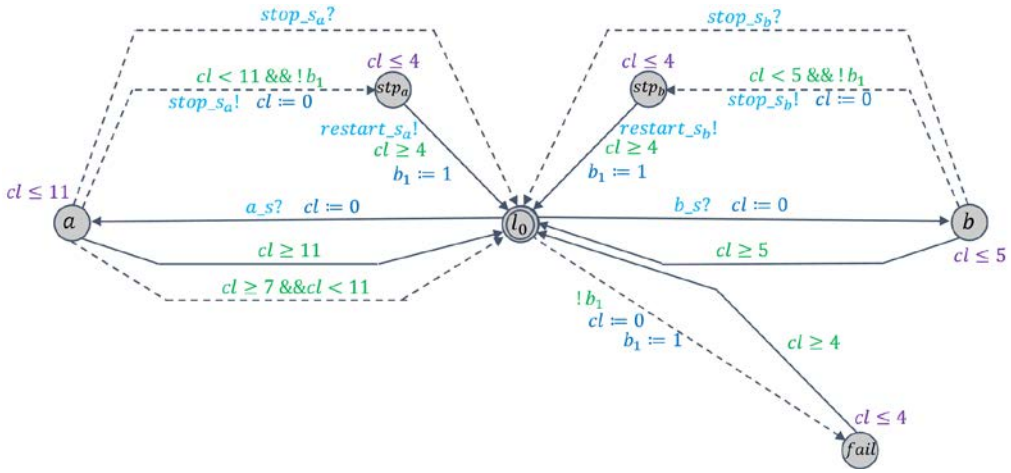

Figure 4.20. ME automaton G_1 in Example 3

If the reason of the fault is a breakdown in another set of resources in another ME automaton, when G_1 is in location a, it receives a signal $stop\_s_a$ (marked as $stop\_s_a$? on the transition) and immediately returns to the location l_0. However, a is not a member of TM_2 and thus doesn't exist in G_2. Thence, this

transition will be never taken and even during failure of resource set 1, other individual tasks in G_2 can be still running.

Performing task b is similar to task a with the difference that its duration is deterministic. In addition, for the reason that b is shared between G_1 and G_2, if any failure happens in any of the resource sets, a signal $stop\_s_b$ will be sent to the other ME automaton, an uncontrollable transition will be taken and thereby both of automata stop doing the task.

If a failure takes place in idle time of the first resource set, an uncontrollable transition from l_0 to $fail$ location will be taken. As a result, b_1 is updated to one and clock is reset. After 4 time units, reparation process terminates and the automaton returns to l_0 and waits to do a task.

In automaton G_2 (Figure 4.21), not only the start time of task c is non-deterministic, but also it needs to reserve resources from its release time. Whenever G_2 receives the signal $decide\_c\_s$ from the automaton G_5, it reaches the location $decide$. In this location, it waits for the action of the environment in automaton G_5 to take the uncontrollable transition from the location $decide$ to f from time 0 to some moments before time 2. If the environment doesn't take any action in automaton G_5, at time 2, the controller takes the controllable transition to f. In either cases, G_5 sends the signal c_s to the other automata. G_2 receives this signal, resets its local clock cl and reaches location c. The remaining process for doing the task is the same as task b.

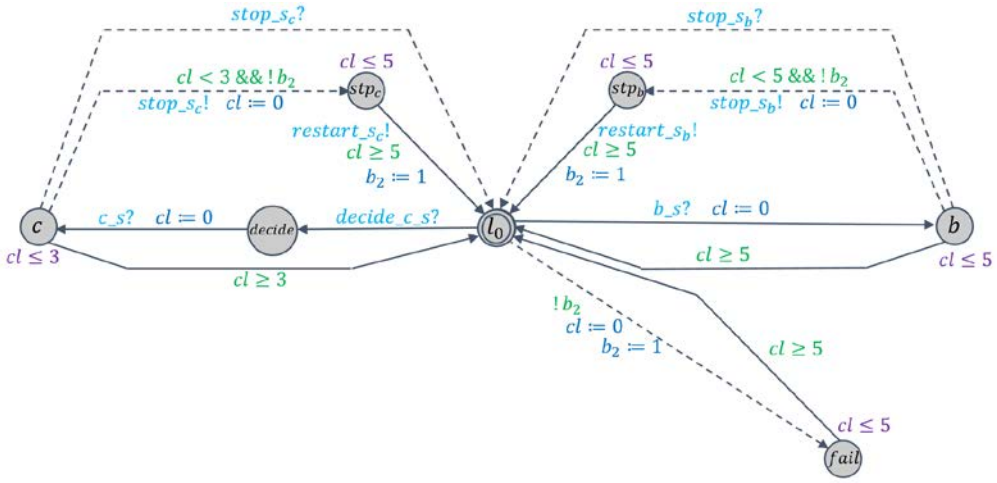

Figure 4.21. ME automaton G_2 in Example 3

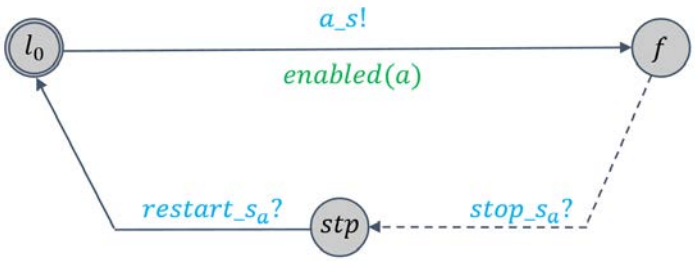

Figure 4.22. TL automaton G_3 for task a in Example 3

In the TL automata of task a, G_3 in Figure 4.22, for taking a transition from l_0 to f, $enabled(a)$ verifies validity of the following predicate and if it was true, the automaton sends signal a_s to G_1 and G_6:

$$a \in TM_1 \rightarrow \neg pending(b, l_1) \tag{4.10}$$

where

$$pending(b, l_1) = \begin{cases} false, & l_1 = l_0 \\ true, & o.w \end{cases} \tag{4.11}$$

where l_1 represents current location of automaton G_1. This predicate means that G_1 is not in task location b and during execution of b.

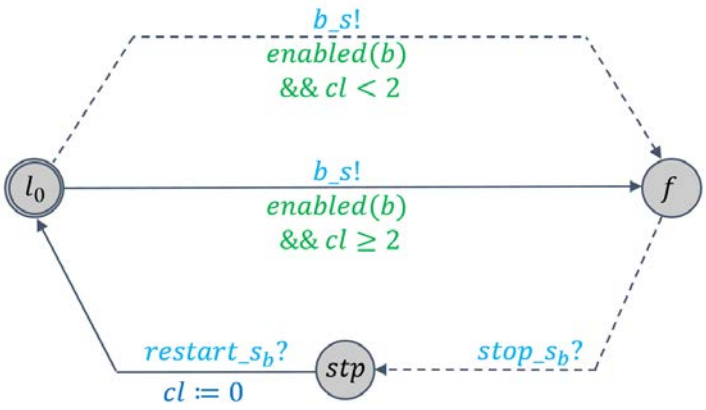

Figure 4.23. TL automaton G_4 for task b in Example 3

In the TL automata of task b, G_4 in Figure 4.23, for taking a transition from l_0 to f, $enabled(b)$ verifies validity of bellow predicate and if it was true, the automaton sends signal b_s to G_1, G_2 and G_6:

$$b \in TM_1 \wedge b \in TM_2 \rightarrow \neg pending(a, l_1) \wedge \neg pending(c, l_2) \tag{4.5}$$

where

$$pending(a, l_1) = \begin{cases} false, & l_1 = l_0 \\ true, & o.w \end{cases} \tag{4.6}$$

$$pending(c, l_2) = \begin{cases} false, & l_2 = l_0 \\ true, & o.w \end{cases} \quad (4.7)$$

which means that ME automata one and two are not in task locations and during execution of tasks a or c respectively.

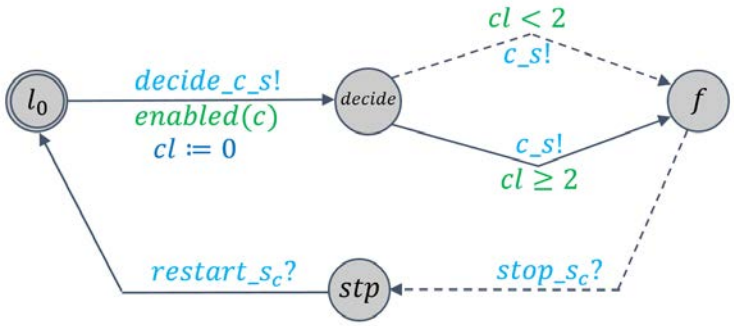

Figure 4.24. TL automaton G_5 for task c in Example 3

In the TL automata of task c, G_5 in Figure 4.24, for taking a transition from l_0 to f, $enabled(c)$ verifies validity of this predicate:

$$b \in TM_2 \rightarrow \neg pending(b, l_2) \quad (4.8)$$

where

$$pending(b, l_2) = \begin{cases} false, & l_2 = l_0 \\ true, & o.w \end{cases} \quad (4.9)$$

which means that G_2 is not in task location b. If it was true, clock is reset and the automaton sends signal $decide\_c\_s$ to G_2 to reserve the related resources.

Figure 4.25 represents PR automata of this example. As depicted in this figure, firstly task b is executed and when it ends, task a can be started. Although, whenever a failure happens during execution of tasks b or a, the automaton returns to locations l_0 and l_2 respectively.

Despite there is a transition from l_2 to l_0, according to the considered delay for task b, in location l_2 the task is already terminated. Therefore, no failure can occur in this location and in order to simplify the automaton, this transition can be omitted. Whereas, if this delay was less than duration of the task, failure could be occurred in this location too. The same is true with the uncontrollable transition from location f to l_2 which is useless in this special example.

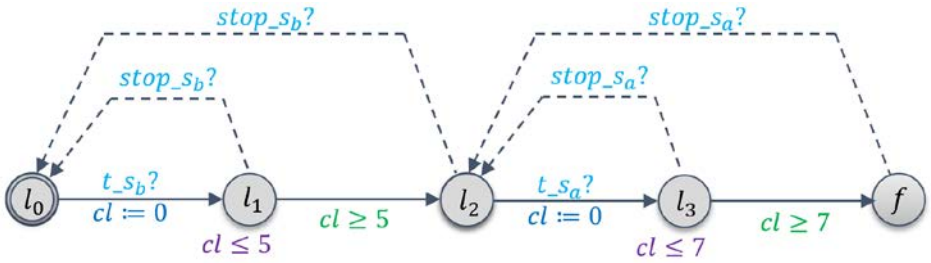

Figure 4.25. PR automata G_6 in Example 3

4.7 Solving approach

In order to find a schedule, the tool TIGA is used as a synthesis tool to explore the state space for determining if there is a winning strategy to reach a location where all the tasks are done and the precedence constraint is respected. For this purpose, a time-optimal control property is verified by performing a reachability game. This game concerns reachability of all the ME automata to their initial locations and all TL and PR automata to their final locations (f) despite uncontrollable actions of the environment. In TCTL language, this property can be formalized as follows:

$$control\_t^*(u,g): A \Diamond ((\bigwedge_{1\le j\le M} ME_j.l_0) \wedge (\bigwedge_{1\le i\le N} task\_launcher_i.f) \wedge (\bigwedge_{1\le k\le p} Precedence_k.f)) \quad (4.10)$$

where l_0 is the initial locations of ME automata, and f are final locations of TL and PR automata. As highlighted previously, there exist M ME automata, N TL automata and P PR automata. Generally, $control\_t^*(u,g): A \Diamond goal$ means that the supervisor must reach a set of goal location within less than $u-g$ time units. In the case of the proposed scheduling problem, goal locations are initial locations l_0 in ME automata and f locations in TL and PR automata. Moreover, u can be set to a very large number and g can have zero value. Thereby, TIGA yields an optimal or sub-optimal value of makespan that can be acquired despite the worst actions of the environment (from the point of view of the supervisor). Furthermore, the strategy to reach the final location can be obtained through this synthesis (Behrmann et al. 2007a).

It is noteworthy to mention that the process of finding a winning strategy is much more expensive than a usual reachability analysis for finding a trajectory to reach the goal location. Thence, the computational time of this solving approach is longer than the previous case and cannot be used for large size MRS scheduling problems.

4.8 Example 3 (continue)

According to the equation 4.10, the following query is verified in TIGA to find the optimal schedule:

$$control\_t^*(100,0): A \Diamond (G_1.l_0 \wedge G_2.l_0 \wedge G_3.f \wedge G_4.f \wedge G_5.f \wedge G_6.f) \qquad (4.11)$$

where l_0 is the initial locations of ME automata G_1 and G_2, and f are final locations of TL automata G_3, G_4 and G_5 and PR automata G_6.

The makespan obtained by TIGA for this example is 45 time units. By implementing the models in this tool, a winning strategy is acquired. For obtaining the strategy, u and g are set to 100 and 0 respectively. In this example, there exist two sets of task conflicts; therefore, there are two sets of resources working simultaneously. It is clear that if a failure occurs in one of the sets, those tasks that do not need both resource sets for execution can be still running.

Figure 4.26 demonstrates the Gantt chart for a sample schedule when the environment does its best play. It means that delays for the start time of tasks are the maximum and failures are happened at moments that can prolong duration of the schedule as much as possible. In this schedule, from time 0 to time 2, the environment doesn't acts and therefore the controller executes task b at time 2. At this instant, b is executed on both sets of resources. Some instances before b ends, a failure happens in the second set of resources. This issue stops b. Maximum reparation time of the second set of resources is 5 time units. Thus, till time 12, resources are repaired and in this moment, ME automaton G_2 allows restarting of task b. Whereas another time, the environment doesn't take action and the execution of b is forced after 2 time units. Therefore, at time 14, b is started at both ME automata and finishes at time 19 when both set of resources are released.

At this instant, automaton G_1 starts execution of task a. At the same moment, TL automaton of task c releases this task and reserves its necessary resources to use them when the environment starts the task. From time 19 to time 21 the environment doesn't acts and therefore the controller executes task c at time 21. At time 24 c finishes.

Duration of task a is non-deterministic and is bounded in interval [7,11]. Until moments before time 30, the environment does not take any action to finish the task. While some instants before time 30, a failure in the first set of resources occurs which stops execution of the task. It takes 4 time units to repair the resources. It is assumed that each set of resources can only fail one time.

Therefore, this set cannot fail anymore. This limit is put to prevent unlimited failures and to bring the problem, closer to reality.

At time 34, the first set of resources are repaired. For the reason that task a was not finished, it should be repeated again. Hence, at time 34, a restarts execution and takes 11 time units to be finished. Therefore all tasks are finished at time 45.

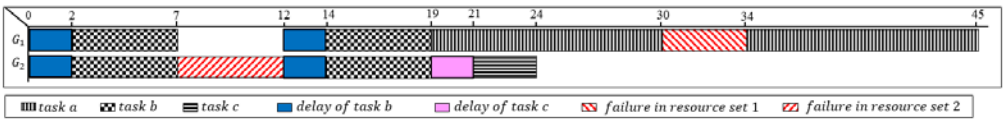

Figure 4.26. Gantt chart for a sample schedule in Example 3- case one

This case is the worst case for occurrence of failures. Failures do not occur when resources are idle. This issue causes an amount of time waste for the reason of repeating tasks. Even this amount is maximum in this schedule; since failures occur at the last moments of execution of the longest tasks. Indeed, the longest tasks of the problem (a and b) took twice as much time as usual.

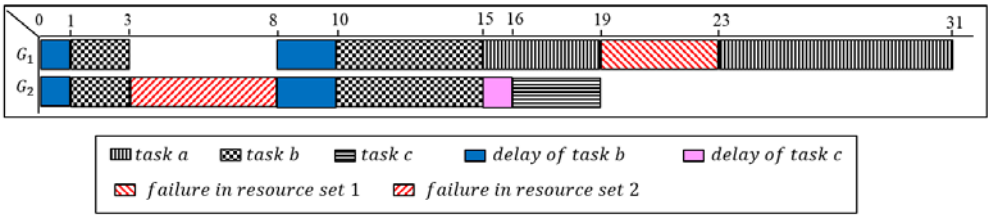

Figure 4.27. Gantt chart for a sample schedule in Example 3- case two

Figure 4.27 shows an alternative schedule that is the result of actions chosen by the environment and the controller for execution of tasks or occurrence of failures and delays. As it can be seen in the figure, in this schedule, task b is executed after 1 time units instead of 2 time units. After 2 time units a failure occurs on the second set of resources which could be happened after 5 time units. Therefore, instead of time 19, task b is done at time 15. Then, task c is executed after 1 time units instead of 2 time units. Furthermore, instead of elapsing 11 time units from execution of a, after 4 time units the failure on the first set of resources happens. They take 4 time units to be repaired. This time, instead of 11 time units, task a ends after 8 time units. Hence, all the tasks are done at time 31. This difference is the result of better choices of the environment (from the point of view of the controller).

4.9 Conclusion

In this chapter, multi-resource sharing scheduling problem is investigated considering uncontrollable parameters that can be happened in the real life. These parameters are uncertain duration of tasks, uncertain start time of tasks or resource failures during a scheduled cycle.

Tasks that are subject to uncertain start time are divided to two sets: those that engage resources from their release time and the ones, which occupy resources only when they are executing.

Furthermore, two types of failures is studied: failure while tasks are being executed and failures during idle time of resources.

In chapter two, it was concluded that WA is a proper means for modeling of MRS problem for which all parameters are controllable. In fact, simple and abstract models can be built for this problem through WA. However, modeling of uncontrollable parameters in the problem is more complicated than controllable parameters and tasks. On the other hand, simplicity of this kind of automaton prevents modeling uncontrollable parameters in the problem. Hence, this kind of problems, is directly modeled by a type of timed automata that considers uncontrollability named timed game automata.

Solving a problem considering uncontrollability needs to perform a supervisory control and extracting a strategy to reach the desired condition despite all actions of the environment. In order to solve the MRS scheduling problem, time-optimal reachability game is performed. Although taking into account uncontrollable parameters yields a schedule closer to the reality, this process is more expensive than simply finding a schedule in which all the tasks are done.

5

Conclusion and perspectives

5 Conclusion and perspectives

5.1 General conclusion

Although wide studies are investigated in scheduling problems by researchers, few studies have used automata and formal verification technics for addressing scheduling problems; yet none of them considered challenging and practical issues such as multi-resource sharing aspect, uncontrollable environment and reaching the optimal schedule in a reasonable time for industrializing the model. This study, focuses on modeling and solving the scheduling problem considering the aforementioned issues.

The MRS scheduling problem is modeled by automata which is expressive and also robust against changes in the parameter setting and the problem hypotheses. Two main types of automata are investigated that can simulate time and task durations for modeling scheduling problems aiming at minimizing makespan: 1. Automata based on weights (i.e. WA) 2. Automata based on clocks (e.g. timed automata). WA is a proper modeling approach for MRS scheduling problem for which all parameters are controllable. It is shown that simpler and more abstract models can be built for this problem through WA. Since all task parameters and events are controllable in Chapter 2 and 3, the problem is modeled by WA in these chapters. In order to find the optimal schedule through the WA model, it is necessary to compose its different components. Hence, an appropriate formalism of synchronous composition is needed to compose components of the model. Whereas there are two issues concerning the existing synchronous compositions in the literature: 1. being unable to show all possible behaviors of the scheduled problem such as simultaneous execution of actions in different automata 2. not containing a trajectory with the minimum makespan for the problem. Consequently, the existing synchronous compositions cannot be used to compose the components of the model. Thus, in order to find the optimal schedule, in Chapter 2, the WA models are translated to timed automata models to be composed through synchronous composition defined for timed automata. Whereas, in Chapter 3, a new synchronous composition is proposed to compose components of the weighed automata model directly. In the sequel, each chapter is explained briefly.

In Chapter 2, an efficient modeling and solving approach is developed for MRS scheduling problem using weighted and timed automata and formal verifications technics. In this chapter, all the task parameters and events are

controllable. Therefore, to model the MRS scheduling problem, a WA model is proposed. In order to obtain the optimal schedule, the proposed WA model is translated to timed automata through some defined rules. In order to find the optimal schedule in a timed automata model, the state space of the model is explored through a reachability analysis. According to the conventional methodology of the reachability analysis, a trajectory should be found in the model in which all the tasks are done. In other words, the scheduling problem becomes a formal verification problem to verify a safety property that guarantees fulfilling all the tasks. In fact, this property is reachability to a set of locations where the condition to reach this set is to complete all the tasks.

To find the optimal schedule, an iterative algorithm is developed. In this algorithm, in each iteration, the aforementioned safety property is verified. By verifying the property, model checker issues a witnessing trajectory that corresponds to one of the possible schedules. The value of clock in the final state of each witnessing trajectory, where all the tasks are done, is equal to the makespan of the schedule. Different schedules could be generated randomly by means of random depth first search method. To find the optimal schedule, several iterations will be done and compared to the optimal schedule.

Two possibilities exist for ending the procedure: 1. To find the optimal makespan in a long time 2. To find the sub-optimal makespan in a period appropriate for decision making depending to the industry. In this thesis, the second approach is followed and therefore, a time instance is determined as criteria for stopping the computation. After reaching this criteria, a suboptimal schedule will be obtained.

The results show that the proposed model and algorithm can be efficiently applied to industrial sized problems with 220 tasks and 20 groups of conflicting task for using various resources. It has been proved that time complexity of the proposed method is polynomial which allows the decision maker to solve an industrial size problem in a period reasonable for making decisions, while time complexity of the problem is NP-hard.

In Chapter 3, a synchronous composition is proposed for WA to enable solving the MRS scheduling problem by performing the developed time-optimal reachability analysis on WA models. In this chapter, a new synchronous composition is presented to compose the WA models proposed in Chapter 2. The advantage of this composition is being enable to execute non-conflicting actions simultaneously. Moreover, due to the approach used in this composition, a schedule with the optimal makespan can be reached using the proposed composition. A generic rule in this composition is such that in each

trajectory, whenever an action is finished in an automaton, intermediate states in all other automata are created. Therefore, the automaton can start another action in that time instant. In that way, the set of tasks can be executed in a shorter time. After building the synchronous composition of the components of the model, a schedule can be found by performing the proposed time-optimal reachability analysis to find the fastest trajectory from the initial state to a state where all the tasks are done.

Finally, in Chapter 4, to bring the problem closer to the real world, the MRS scheduling problem is extended by considering uncontrollable events and uncertain parameters. These parameters are uncertain duration and start time of tasks and resource failures during a scheduled cycle.

Tasks that are subject to uncertain start time are divided to two sets: those that engage resources from their release time and the ones, which occupy resources only when they are executing. Furthermore, two types of failures are studied in this thesis: failure while tasks are being executed and failures during idle time of resources.

However, modeling of uncontrollable parameters in MRS scheduling problem is complicated. Timed game automata is a modeling formalism which support clocks and include uncontrollable transitions and has sufficient features to model the extended problem. Thus, the extended MRS scheduling problem is modeled by timed game automata.

Solving a problem considering uncontrollability needs to perform a supervisory control and extracting a supervisor strategy to reach the desired condition despite all actions of the environment. In order to solve the MRS scheduling problem, time-optimal reachability game is performed. Whereas, this process is more expensive than simply finding a schedule in which all the tasks are done. Thence, although taking into account uncontrollable parameters yields a schedule closer to the reality, this process is much longer to be solved in large sizes.

5.2 General perspectives

In this section, some limitations of this work are presented and possible directions for future researches are given. Particularly four main future research directions can be proposed:

- The defined rules for translating WA models to timed automata models can be implemented in a software to obtain timed automata models automatically.
- The synchronous composition of WA could be implemented in a software to compose WA models automatically. Furthermore, the time-optimal reachability analysis discussed in Chapter 3 could be implemented in a software in order to find the optimal schedule automatically.
- As explained in motivations, task are divided to operation tasks and preventive maintenance tasks. In some application, it may be an advantage to perform operation tasks earlier than maintenance. Whereas, by postponing the maintenance, risk of failure will increase which may need more time to be fixed. Therefore, two cases may happen by postponing the preventive maintenance of resources: 1. Operational tasks may be fulfilled earlier and the makespan for doing them may decrease 2. Failure occurs before performing maintenances and finishing tasks and the makespan increases. Thus, maintenance can be postponed depending on two parameters: 1. the importance of makespan minimization in the specific industry and 2. probability of failure in each instance of the schedule
- A new formalism for WA could be defined in order to model uncontrollable events like occurrence of failure while execution of tasks. In addition, the algorithm of synchronous composition could be extended to compose WA models containing uncontrollability issues.

5.3 Related publication

The results of this thesis is published / submitted as follows:

- M. Rahimi, E. Niel, E. Dumitrescu, “Multi-resource Scheduling by Weighted and Timed Automata”, International Journal of Production Research (ready to submit)
- M. Rahimi, E. Niel, E. Dumitrescu, (2015) “Scheduling by Timed Automata under Resource Conflicts”, Poster presented in: 10ème Colloque sur la Modélisation des Systèmes Réactifs (MSR 2015), Nov. 18-20, Nancy, France

References

Abdeddaïm, Y., E. Asarin, and M. Sighireanu. 2009. “Simple Algorithm for Simple Timed Games.” In *TIME 2009 - 16th International Symposium on Temporal Representation and Reasoning*, 99–106. doi:10.1109/TIME.2009.14.

Abdeddaim, Y, and O Maler. 2001. “Job-Shop Scheduling Using Timed Automata.” In *Computer Aided Verification*, 1:478–92.

Abdeddaïm, Yasmina, Eugene Asarin, and Oded Maler. 2006. “Scheduling with Timed Automata.” In *Theoretical Computer Science*, 354:272–300. doi:10.1016/j.tcs.2005.11.018.

Abdeddaïm, Yasmina, Abdelkarim Kerbaa, and Oded Maler. 2003. “Task Graph Scheduling Using Timed Automata.” In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 8–pp. doi:10.1109/IPDPS.2003.1213431.

Afzalirad, Mojtaba, and Javad Rezaeian. 2016. “Resource-Constrained Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times, Precedence Constraints and Machine Eligibility Restrictions.” *Computers and Industrial Engineering* 98: 40–52. doi:10.1016/j.cie.2016.05.020.

Al-Bataineh, Omar Ibrahim. 2015. “Verifying Worst-Case Execution Time of Timed Automata Models with Cyclic Behaviour.” The University of Western Australia.

Alur, R, C Courcoubetis, and D L Dill. 1993. “Model Checking in Dense Real Time.” *Information and Computation* 104 (1): 2–34. doi:10.1006/inco.1993.1024.

Alur, Rajeev, Salvatore La Torre, and George J. Pappas. 2004. “Optimal Paths in Weighted Timed Automata.” In *Theoretical Computer Science*, 318:297–322. Elsevier. doi:10.1016/j.tcs.2003.10.038.

Alves, Lucas V.R. R., Hugo J. Bravo, Patricia N. Pena, and Ricardo H.C. C. Takahashi. 2016. “Planning on Discrete Events Systems: A Logical Approach.” In *IEEE International Conference on Automation Science and Engineering*, 1055–60. IEEE. doi:10.1109/COASE.2016.7743520.

Atto, Abdourrahmane M., Claude Martinez, and Saïd Amari. 2011. “Control of Discrete Event Systems with Respect to Strict Duration: Supervision of an Industrial Manufacturing Plant.” *Computers and Industrial Engineering* 61 (4). Elsevier Ltd: 1149–59. doi:10.1016/j.cie.2011.07.004.

Baier, Christel, and Joost-Pieter Katoen. 2008. *Principles Of Model Checking*. *MIT Press*. Vol. 950. doi:10.1093/comjnl/bxp025.

Behrmann, Gerd, Ed Brinksma, Martijn Hendriks, and Angelika Mader. 2005. “Production Scheduling by Reachability Analysis - A Case Study.” In *19th IEEE International Parallel and Distributed Processing Symposium*, 140a–140a. IEEE. doi:10.1109/IPDPS.2005.363.

Behrmann, Gerd, A. Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. 2006. “UPPAAL-Tiga: Timed Games for Everyone.” In *Nordic Workshop on Programming Theory (NWPT’06)*.

Behrmann, Gerd, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G Larsen, and Didier Lime. 2007a. "Uppaal Tiga User-Manual." Aalborg University.

Behrmann, Gerd, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Larsen, and Didier Lime. 2007b. "UPPAAL-Tiga : Time for Playing Games." In *Computer Aided Verification*, 121–25.

Behrmann, Gerd, Alexandre David, and Kim G Larsen. 2006. "A Tutorial on Uppaal 4.0."

Behrmann, Gerd, Ansgar Fehnker, Thomas Hune, Kim G Larsen, Paul Pettersson, Judi Romijn, and Frits W Vaandrager. 2001. "Minimum-Cost Reachability for Priced Timed Automata." *HSCC* 1: 147–61.

Behrmann, Gerd, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, and Judi Romijn. 2001. "Efficient Guiding Twoards Cost-Optimality in Uppaal." In *Tools and Algorithms for the Construction and Analysis of Systems*, 174.

Behrmann, Gerd, Kim G. Larsen, and Jacob I. Rasmussen. 2005. "Optimal Scheduling Using Priced Timed Automata." *ACM SIGMETRICS Performance Evaluation Review* 32 (4): 34–40. doi:10.1145/1059816.1059823.

Bengtsson, Johan, Wang Yi, Johan Bengtsson, Wang Yi, and Wang Yi. 2004. "Timed Automata: Semantics, Algorithms and Tools." In *Lecture Notes in Computer Science*, 3098:87–124. Springer Berlin Heidelberg. doi:10.1007/978-3-540-27755-2_3.

Berezin, Sergey, Sérgio Campos, and Edmund M Clarke. 1998. "Compositional Reasoning in Model Checking." In *Lecture Notes in Computer Science*, 1536:81–103. doi:10.1007/3-540-49213-5_4.

Bornot, Sébastien, Gregor Gößler, and Joseph Sifakis. 2001. "On the Construction of Live Timed Systems." In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 1785:109–26. Springer, Berlin, Heidelberg. doi:10.1007/3-540-46419-0_9.

Bouajjani, Ahmed, Stavros Tripakis, and Sergio Yovine. 1997. "On-the-Fly Symbolic Model Checking for Real-Time Systems." In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, 25–34. doi:10.1109/REAL.1997.641266.

Boukra, Rabah, Sébastien Lahaye, and Jean-Louis Boimond. 2015. "New Representations for (Max,+) Automata with Applications to Performance Evaluation and Control of Discrete Event Systems." *Discrete Event Dynamic Systems* 25 (1–2): 295–322. doi:10.1007/s10626-013-0178-y.

Cassez, Franck, Alexandre David, Emmanuel Fleury, Kim G Larsen, and Didier Lime. 2005. "Efficient on-the-Fly Algorithms for the Analysis of Timed Games." In *CONCUR: International Conference on Concurrency Theory*, 5:66–80. doi:10.1007/11539452_9.

Castro, Pedro M., Iiro Harjunkoski, and Ignacio E. Grossmann. 2009. "New Continuous-Time Scheduling Formulation for Continuous Plants under Variable Electricity Cost." *Industrial & Engineering Chemistry Research* 48 (14). American Chemical Society: 6701–14. doi:10.1021/ie900073k.

Cimatti, Alessandro, Andrea Micheli, and Marco Roveri. 2015. “Strong Temporal Planning with Uncontrollable Durations : A State-Space Approach.” In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 3254–60.

Confessore, Giuseppe, Stefano Giordani, and Silvia Rismondo. 2007. “A Market-Based Multi-Agent System Model for Decentralized Multi-Project Scheduling.” *Annals of Operations Research* 150 (1): 115–35. doi:10.1007/s10479-006-0158-9.

David, Alexandre, Jacob Illum, Kim G. Larsen, and Arne Skou. 2009. “Model-Based Framework for Schedulability Analysis Using Uppaal 4.1.” In *Model-Based Design for Embedded Systems*, 1–32.

Delaval, Gwenael, Nol De Palma, Soguy Mak Kare Gueye, Herve Marchand, and Eric Rutten. 2013. “Discrete Control of Computing Systems Administration: A Programming Language Supported Approach.” *2013 European Control Conference, ECC 2013*, 117–24.

Dorndorf, Ulrich, Florian Jaehn, and Erwin Pesch. 2017. “Flight Gate Assignment and Recovery Strategies with Stochastic Arrival and Departure Times.” *OR Spectrum* 39 (1): 65–93. doi:10.1007/s00291-016-0443-1.

Dumitrescu, Emil, Alain Girault, Hervé Marchand, and Eric Rutten. 2010. “Multicriteria Optimal Reconfiguration of Fault-Tolerant Real-Time Tasks.” In *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 10:356–63.

Edis, Emrah B., Ceyda Oguz, and Irem Ozkarahan. 2013. “Parallel Machine Scheduling with Additional Resources: Notation, Classification, Models and Solution Methods.” *European Journal of Operational Research* 230 (3). Elsevier B.V.: 449–63. doi:10.1016/j.ejor.2013.02.042.

Edis, Emrah B., and Irem Ozkarahan. 2011. “A Combined Integer/constraint Programming Approach to a Resource-Constrained Parallel Machine Scheduling Problem with Machine Eligibility Restrictions.” *Engineering Optimization* 43 (2): 135–57. doi:10.1080/03052151003759117.

Fernández Anta, Antonio, Chryssis Georgiou, Dariusz R. Kowalski, and Elli Zavou. 2015. “Online Parallel Scheduling of Non-Uniform Tasks: Trading Failures for Energy.” *Theoretical Computer Science* 590: 129–46. doi:10.1016/j.tcs.2015.01.027.

Gaubert, S, and J Mairesse. 1995. “Task Resource Models and (Max,+) Automata” 11: 133–44.

Gaubert, Stephane. 1995. “Performance Evaluation of (Max,+) Automata.” *IEEE Transactions on Automatic Control* 40 (12): 2014–25. doi:10.1109/9.478227.

Girault, Alain, Hamoudi Kalla, Mihaela Sighireanu, and Yves Sorel. 2003. “An Algorithm for Automatically Obtaining Distributed and Fault-Tolerant Static Schedules.” *Proceedings of the International Conference on Dependable Systems and Networks* 0 (c): 159–68. doi:10.1109/DSN.2003.1209927.

Hartmann, Sönke, and Dirk Briskorn. 2010. “A Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem.” *European Journal of Operational Research* 207 (1): 1–14. doi:10.1016/j.ejor.2009.11.005.

Heimerl, Christian, and Rainer Kolisch. 2010. “Scheduling and Staffing Multiple Projects with a Multi-Skilled Workforce.” *OR Spectrum* 32 (2): 343–68. doi:10.1007/s00291-009-0169-4.

Hune, T., K.G. Larsen, and P. Pettersson. 2001. “Guided Synthesis of Control Programs Using UPPAAL.” *Nordic Journal of Computing* 8 (1): 43–64. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.251&rep=rep1&type=pdf.

Kellerer, Hans, and Vitaly A. Strusevich. 2008. “Scheduling Parallel Dedicated Machines with the Speeding-up Resource.” *Naval Research Logistics* 55 (5): 377–89. doi:10.1002/nav.20292.

Kim, Ja Hee, Mengchu Zhou, and Tae Eog Lee. 2014. “Schedule Restoration for Single-Armed Cluster Tools.” *IEEE Transactions on Semiconductor Manufacturing* 27 (3): 388–99. doi:10.1109/TSM.2014.2315871.

Komenda, Jan, Sebastien Lahaye, and Jean-Louis Boimond. 2010. “Synchronous Composition of Interval Weighted Automata.” *IFAC Proceedings Volumes 43* 43 (12): 318–23.

Komenda, Jan, Sébastien Lahaye, and Jean-louis Boimond. 2009a. “Control of (Max,+) Automata: A Single Step Approach.” In *Control Conference (ECC), 2009 European*, 1985–90.

Komenda, Jan, Sebastien Lahaye, and Jean Louis Boimond. 2009b. “Supervisory Control of (Max,+) Automata: A Behavioral Approach.” *Discrete Event Dynamic Systems: Theory and Applications* 19 (4): 525–49. doi:10.1007/s10626-009-0083-6.

Komenda, Jan, Sébastien Lahaye, and Jean Louis Boimond. 2009c. “Le Produit Synchrone Des Automates (Max,+).” *Journal Europeen Des Systemes Automatises* 43 (7–9): 1033–47. doi:10.3166/jesa.43.1033-1047.

Kundakcı, Nilsen, and Osman Kulak. 2016. “Hybrid Genetic Algorithms for Minimizing Makespan in Dynamic Job Shop Scheduling Problem.” *Computers & Industrial Engineering* 96: 31–51. doi:10.1016/j.cie.2016.03.011.

Lahaye, Sébastien, Jan Komenda, and Jean-louis Boimond. 2012. “Modélisation Modulairea L’aide D’automates (Max,+).” In *Conférence Internationale Francophone d’Automatique-CIFA*, 895–900.

Lahaye, Sébastien, Jan Komenda, and Jean Louis Boimond. 2015. “Compositions of (Max, +) Automata.” *Discrete Event Dynamic Systems: Theory and Applications* 25 (1–2): 323–44. doi:10.1007/s10626-014-0186-6.

Landau, LD, Carla Seatzu, Manuel Silva, Jan H. Van Schuppen, and LD Landau. 2013. *Control of Discrete-Event Systems. Zhurnal Eksperimental’noi I Teoreticheskoi Fiziki*. Vol. 433. doi:10.1007/978-1-4471-4276-8.

Larsen, Kim G., Paul Pettersson, and Wang Yi. 1997. “Uppaal in a Nutshell.” *International Journal on Software Tools for Technology Transfer* 1 (1–2): 134–52. doi:10.1007/s100090050010.

Liu, Xinxin, and Scott A. Smolka. 1998. “Simple Linear-Time Algorithms for Minimal Fixed Points.” *Icalp*, 53–66. doi:10.1007/BFb0055040.

Lu, Zhiqiang, Weiwei Cui, and Xiaole Han. 2015. “Integrated Production and Preventive Maintenance Scheduling for a Single Machine with Failure Uncertainty.” *Computers and Industrial Engineering* 80. Elsevier Ltd: 236–44. doi:10.1016/j.cie.2014.12.017.

Luo, Hao, Bing Du, George Q Huang, Huaping Chen, and Xiaolin Li. 2013. “Hybrid Flow Shop Scheduling Considering Machine Electricity Consumption Cost.” *Intern. Journal of Production Economics* 146: 423–39. doi:10.1016/j.ijpe.2013.01.028.

Marangé, Pascale, Jean François Pétin, Antoine Manceaux, and David Gouyon. 2011. “Contribution À La Reconfiguration Des Systèmes de Production: Ordonnancement Par Recherche D’atteignabilité.” *Journal Europeen Des Systemes Automatises* 45 (1–3): 45–60. doi:10.3166/JESA.45.45-60.

Milner, Robin. 1983. “Calculi for Synchrony and Asynchrony.” *Electronic Notes in Theoretical Computer Science* 1 (C): 66–90. doi:10.1016/S1571-0661(04)80005-7.

Moon, Joon Yung, and Jinwoo Park. 2014. “Smart Production Scheduling with Time-Dependent and Machine-Dependent Electricity Cost by Considering Distributed Energy Resources and Energy Storage.” *International Journal of Production Research* 52 (13): 3922–39. doi:10.1080/00207543.2013.860251.

Munter, R. De. 2010. “A Comparison of Timed Games and Time Optimal Supervisor Synthesis.”

Niebert, P, and S Yovine. 2001. “Computing Efficient Operation Schemes for Chemical Plants in Multi-Batch Mode.” *European Journal of Control: Verification of Hybrid Systems* 7 (4): 440–53. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.5061&rep=rep1&type=pdf.

Niebert, Peter, Stavros Tripakis, and Sergio Yovine. 2000. “Minimum-Time Reachability for Timed Automata.” In *IEEE Mediteranean Control Conference*. https://pdfs.semanticscholar.org/56d3/ae59da80dce64dd7aa2e92e1a760aa09b4ae.pdf.

Nikou, Alexandros, Jana Tumova, and Dimos V. Dimarogonas. 2016. “Cooperative Task Planning of Multi-Agent Systems under Timed Temporal Specifications.” In *Proceedings of the American Control Conference*, 7104–9. IEEE. doi:10.1109/ACC.2016.7526793.

Norström, Christer, Anders Wall, and Wang Yi. 1999. “Timed Automata as Task Models for Event-Driven Systems.” In *Real-Time Computing Systems and Applications, 1999. RTCSA ’99. Sixth International Conference on*, 182–89. doi:10.1109/RTCSA.1999.811218.

Ooshita, Fukuhito, Tomoko Taisuke Izumi, and Tomoko Taisuke Izumi. 2009. “A Generalized Multi-Organization Scheduling on Unrelated Parallel Machines.” *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies*, December. IEEE, 26–33. doi:10.1109/PDCAT.2009.26.

Panek, Sebastian, Sebastian Engell, and Olaf Stursberg. 2006. “Scheduling and

Planning with Timed Automata." *Computer Aided Chemical Engineering* 21 (C): 1973–78. doi:10.1016/S1570-7946(06)80337-8.

Panek, Sebastian, Olaf Stursberg, and Sebastian Engell. 2006. "Efficient Synthesis of Production Schedules by Optimization of Timed Automata." *Control Engineering Practice* 14 (10): 1183–97. doi:10.1016/j.conengprac.2006.02.014.

Quintero, Karla, Eric Niel, and José Aguilar. 2015. "(Max,+) Model for Alignment Selection and Schedule Optimization in a Flow Network." *AIP Conference Proceedings* 1648: 11–14. doi:10.1063/1.4912931.

Quintero, Karla, Eric Niel, Aguilar José, and Laurent Piétrac. 2013. "(Max, +) Optimization Model for Scheduling Maintenance Tasks." In *Proceedings of the World Congress on Engineering and Computer Science*, II:23–25.

Quintero Garcia, Karla Rossa. 2015. "Optimisation D'alignements D'un Réseau de Pipelines Basée Sur Les Algèbres Tropicales et Les Approches Génétiques." Lyon, INSA. http://www.theses.fr/2015ISAL0030.

Quintero Garcia, Karla Rossa, Eric Niel, José Aguilar, and Laurent Piétrac. 2014. "Scheduling Operations in a Flow Network with Flexible Preventive Maintenance: A (Max, +) Approach." *Engineering Letters* 22 (1): 24–33. http://www.engineeringletters.com/issues_v22/issue_1/EL_22_1_04.pdf.

Quintero Garcia, Karla Rossa, Eric Niel, Aguilar José, and Laurent Piétrac. 2014. "A Cost-Criticality Based (Max, +) Optimization Model for Operations Scheduling." *Transactions on Engineering Technologies: Special Issue of the World Congress on Engineering and Computer Science 2013*, 645–60. doi:10.1007/978-94-017-9115-1_47.

Rzevski, G., and P. Skobelev. 2017. "Intelligent Adaptive Schedulers For Railways." *International Journal of Transport Development and Integration* 1 (3). WIT Press: 414–20. doi:10.2495/TDI-V1-N3-414-420.

Sedgewich, Robert, and Kevin Wayne. 2017. "Algorithm." Accessed August 11. http://algs4.cs.princeton.edu/lectures/42DirectedGraphs.pdf.

Shehabinia, Ahmad Reza, Liyong Lin, and Rong Su. 2016. "Timed Supervisory Control for Operational Planning and Scheduling under Multiple Job Deadlines." *arXiv Preprint arXiv:1607.04255*. https://arxiv.org/pdf/1607.04255.pdf.

Su, Rong, Jan H. Van Schuppen, and Jacobus E. Rooda. 2012. "The Synthesis of Time Optimal Supervisors by Using Heaps-of-Pieces." *IEEE Transactions on Automatic Control* 57 (1): 105–18. doi:10.1109/TAC.2011.2157391.

Subbiah, Subanatarajan, and Sebastian Engell. 2010. "Short-Term Scheduling of Multi-Product Batch Plants with Sequence-Dependent Changeovers Using Timed Automata Models." *Computer Aided Chemical Engineering* 28 (C): 1201–6. doi:10.1016/S1570-7946(10)28201-9.

Tripakis, Stavros. 1998. "L'analyse Formelle Des Systèmes Temporisés En Pratique." Université Joseph Fourier-Grenoble 1. https://tel.archives-ouvertes.fr/tel-00004907v2.

Ware, Simonn, and Rong Su. 2017. "Time Optimal Synthesis Based Upon Sequential Abstraction and Its Application to Cluster Tools." *IEEE Transactions on*

Automation Science and Engineering 14 (2): 772–84. doi:10.1109/TASE.2016.2613911.

Wilson, Président, and Cachan Cedex France. 2000. "Comparing Verification with Comparing Veri Cation with HyTech , Kronos and Uppaal on the Railroad Crossing Example."

Xi, Xiaoying, Lili Jiang, and Qiang Zhang. 2009. "Optimization for Multi-Resources-Constrained Job Shop Scheduling Based on Three-Level Heuristic Algorithm." In *Proceedings - 2009 International Asia Conference on Informatics in Control, Automation, and Robotics, CAR 2009*, 296–300. doi:10.1109/CAR.2009.28.

Xian, Changjiu, Yung-Hsiang Lu, and Zhiyuan Li. 2007. "Energy-Aware Scheduling for Real-Time Multiprocessor Systems with Uncertain Task Execution Time." *Proceedings of the 44th Annual Conference on Design Automation - DAC '07*, 664. doi:10.1145/1278480.1278648.

Yusta, J M, F Torres, and H M Khodr. 2010. "Optimal Methodology for a Machining Process Scheduling in Spot Electricity Markets." *Energy Conversion and Management* 51: 2647–54. doi:10.1016/j.enconman.2010.05.030.

Zobolas, G. I., C. D. Tarantilis, and G. Ioannou. 2008. "Exact, Heuristic and Meta-Heuristic Algorithms for Solving Shop Scheduling Problems." In *Studies in Computational Intelligence*, 128:1–40. Springer. doi:10.1007/978-3-540-78985-7_1.

Appendix A: Implementation details of the model

The transition guard in a TL automata ensures that when taking this transition, all ME automata that share the task are in initial location to synchronize their corresponding transitions. Therefore TL automata should be able to access to current locations of ME automata. There is no feature to handle this issue in UPPAAL. Hence, it should be managed indirectly.

From another point of view, this guard guaranteed that when any of the ME automata relating to a task is not in initial location, its TL automaton won't be able to execute the task. To simulate this feature, in each ME automaton, when starting a double-task-transition, function "$disbl\_othr\_t\_expt$" disables launching other tasks belonging to the same automaton. Then, after completing the task, function "$enbl\_othr\_t\_expt$" enables these tasks. More precisely, an initially zero variable $en[i]$, where i is the number of task, is dedicated to each task. Each time taking the first transition of a double-task-transition, the first function increments $en[i]$ variable of other tasks belonging to the automaton to disable their execution. Then at the second transitions, the second function decrements this variable by one. Since a task might be common in more than one specification automata, this variable might be incremented and decremented more than one time. Whereas after execution of all double-task-transitions related to a task, these variables reset to their default value of zero. Hence, whenever this variable is equal to zero, it means that the task is enabled. In TL automata, a guard is defined on the transition such that if $en[id]$ is equal to zero (id denotes number of the task that is defined as id number of automaton), the transition can be taken. Consequently by sending communication signals to specification automata, corresponding transitions will be taken and tasks will be launched. Bellow the aforementioned functions are detailed:

```
void disbl_othr_t_expt(int i) {
  for (k : int[0, N − 1]) {
    if (sel(k) and k != i)
      en[k]++;
  }
}

void enbl_othr_t_expt(int i) {
  for (k : int[0, N − 1]) {
```

```
    if (sel(k) and k != i)
      en[k]--;
  }
}
```

By applying modifications according to the above stated functions, following models are acquired. In these models, $i, j, k, \dots, p$ denote numbers assigned to tasks $a, b, c, \dots, p$.

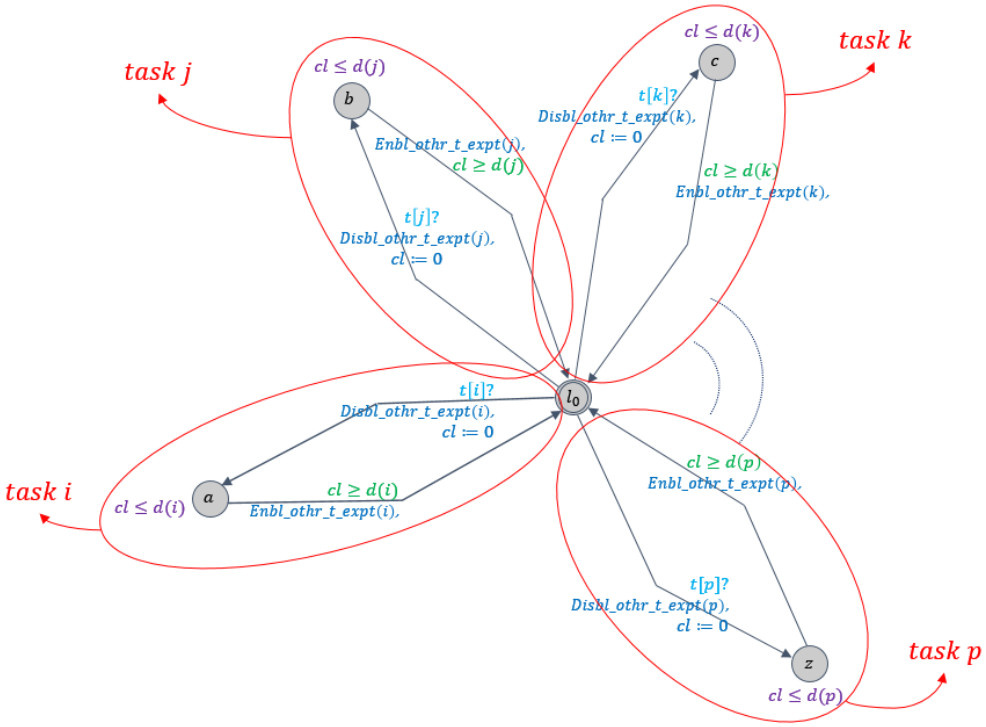

Figure A.1. Implemented modeling pattern of ME automaton

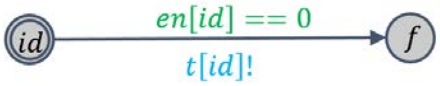

Figure A.2. Implemented modeling pattern of a TL timed automaton

Figure A.3. Implemented modeling pattern of a triggering precedence timed automaton

Figure A.4. Implemented modeling pattern of a delay precedence timed automaton

Appendix B: Java code for depth-first search in a digraph

```
public class DirectedDFS (Sedgewich and Wayne 2017)
{
   private boolean[]marked;  //true if path from s
    public DirectedDFS(Digraph G, int s)
    {
       marked = new boolean[G.V()]; /
                /constructor marks vertices reachable from s
       dfs(G, s);
    }
   private void dfs(Digraph G, int v)  //recursive DFS does the work
   {
       marked[v] = true;
       for (int w : G.adj(v))
           if (!marked[w])dfs(G, w);
   }
   public boolean visited(int v) /
                /client can ask whether any vertex is reachable from s
   { return marked[v]; }
}
```

Appendix C: Abbreviations

BFS	Breadth-First Search
CP	Constraint Programming
DFS	Depth-First Search
DSDA	Deterministic Single-Duration-Action
DVS	Dynamic Voltage Scaling
EDF	Earliest Dead-line First
FIFO	First In First Out
FPS	Fixed Priority Scheduling
IP	Integer Programming
LIFO	Last In First Out
LPTA	Linearly Priced Timed Automata
ME	Mutual Exclusion
MILP	Mixed-Integer Linear Programming
MITL	Metric Interval Temporal Logic
MRS	Multi-Resource Sharing
PR	Precedence
SOM	Sub-Optimal Makespan
TA	Timed Automaton
TCTL	Timed Computation Tree Logic
TGA	Timed Game Automata
TL	Task Launcher
UPTA	Uniformly Priced Timed Automaton
WA	Weighted Automaton